Руководство разработчика

Начало работы

Следующие материалы и технологии используются при разработке и запуске проектов Канцлер RPA:

- Lombok - java-библиотека, которая преобразует аннотации в Java-код.
- Maven - инструмент для управления и сборки проектов.
- Git - распределенная система управления версиями.
- Docker - платформа для разработчиков и системных администраторов, предназначенная для создания, запуска и обмена приложениями с помощью контейнеров.
- IntelliJ IDEA - интегрированная среда разработки на Java, предназначенная для разработки компьютерного программного обеспечения.
- Selenium -инструмент для автоматизации действий веб-браузера.

Настройка среды разработки

Установка JDK

Так как Канцлер RPA является проектом Java, требуется установить JDK. В настоящее время мы поддерживаем минимальную версию Java 8. Мы рекомендуем использовать Oracle JDK 8. Убедитесь, что переменная среды `JAVA_HOME` установлена и указывает на ваш JDK после установки.

Установка Maven

Канцлер RPA использует Maven для сборки проектов. Вы можете скачать Apache Maven с официального сайта. Извлеките архив дистрибутива в любой каталог и добавьте каталог bin распакованного дистрибутива в переменную среды `PATH` вашего пользователя. Дополнительные сведения см. в документации Установка Apache Maven.

Использование Maven с репозиториями HTTP

Внешние небезопасные URL-адреса HTTP блокируются по умолчанию, начиная с версии 3.8.1.

Это может оказать негативное влияние, если ваш сервер Канцлер RPA Nexus имеет небезопасный URL-адрес HTTP.

Вы можете установить более раннюю версию Maven 3.6.3: https://archive.apache.org/dist/maven/maven-3/3.6.3/binaries/

Или ознакомиться с более подробной информацией в разделе "How to fix when I get a HTTP repository blocked?": https://maven.apache.org/docs/3.8.1/release-notes.html#how-to-fix-when-i-get-a-http-repository-blocked

Установка Git

Git — отличный инструмент для управления версиями. Получить установщик для вашей операционной системы можно здесь.

После этого необходимо настроить локально установленный Git для использования учетных данных GitHub. Запустите командную строку и выполните команды:

```
git config --global user.name “github_username”
```

```
git config --global user.email “email_address”
```

Замените **github_username** и **email_address** своими учетными данными GitHub.

Среда **IDEA** по умолчанию содержит плагин Git, этой функцией можно воспользоваться для разработки RPA вместо загрузки Git с сайта.

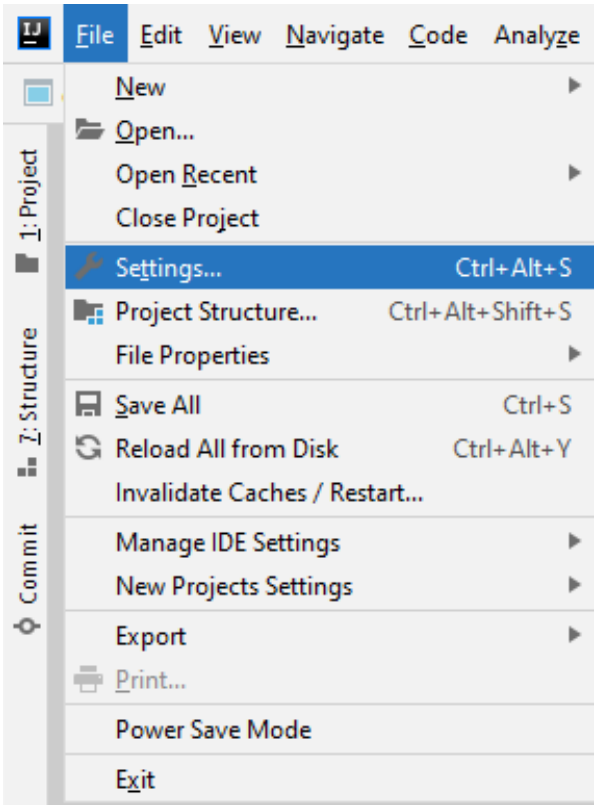
Установка IDE

Несмотря на то, что с Канцлер RPA можно работать и в других интегрированных средах разработки, мы рекомендуем начать с IntelliJ IDEA Community Edition IDE. Скачать установщик можно здесь.

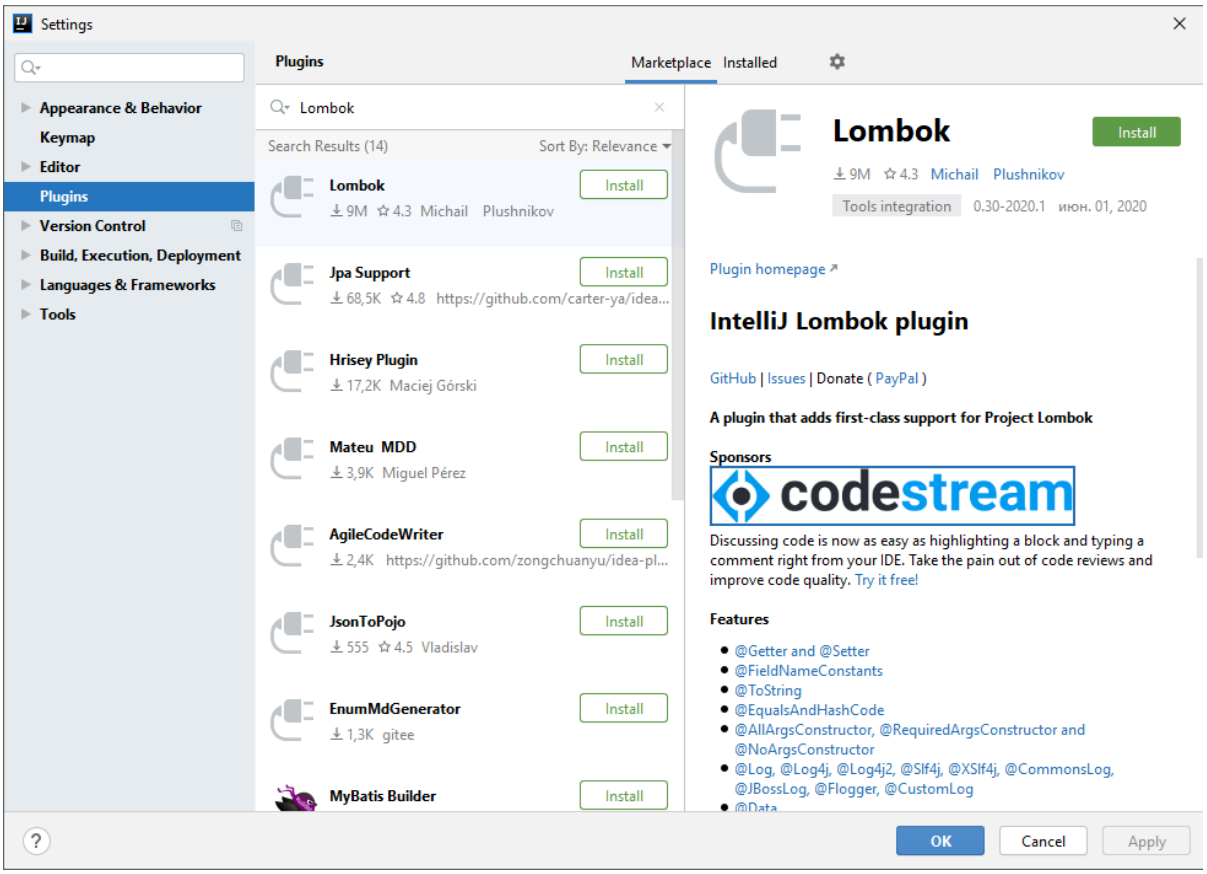
Установка плагина Lombok

Плагин Lombok для IDEA делает написание кода более удобным. Ознакомиться с его возможностями можно на официальном сайте.

Запустите IntelliJ IDEA IDE. Перейдите в меню **Файл** **Настройки**:

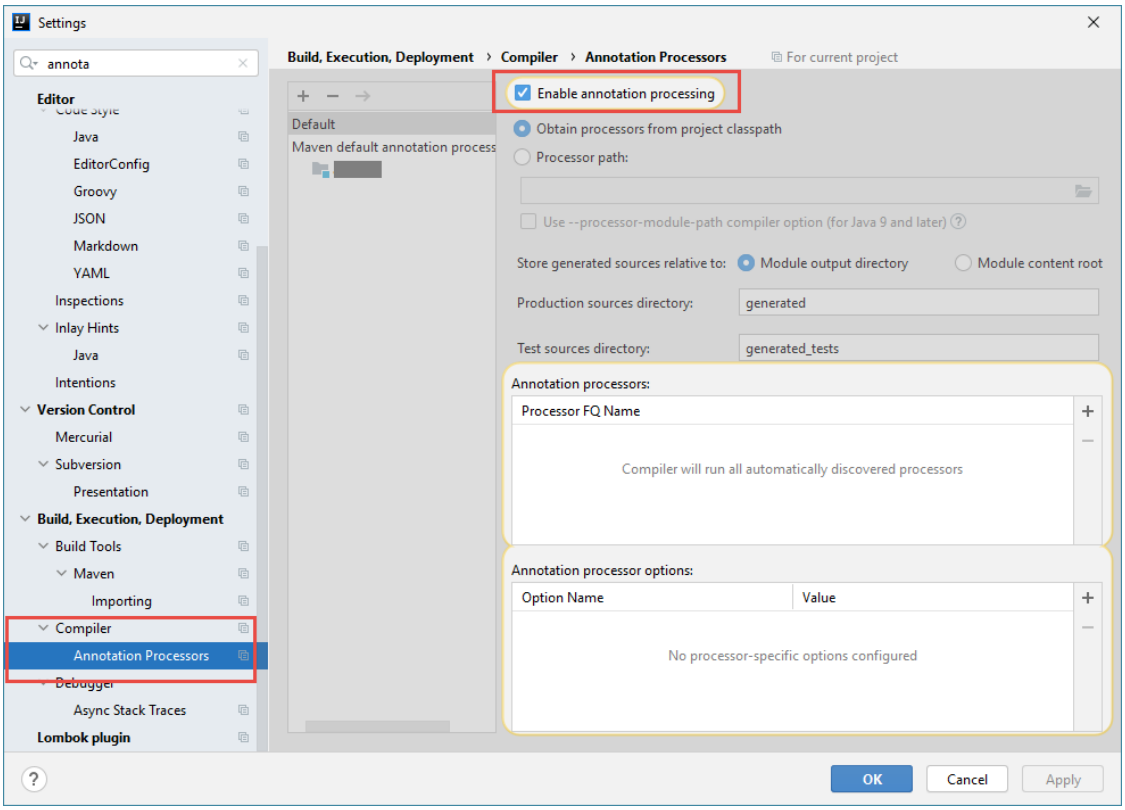


На левой панели выберите **Плагины**, введите *lombok* в поле поиска и нажмите **Ввод**.



Нажмите кнопку **Установить** для *Lombok* и перезапустите IDE.

Включите обработку аннотаций:



Добавление сертификата в хранилище доверенных сертификатов

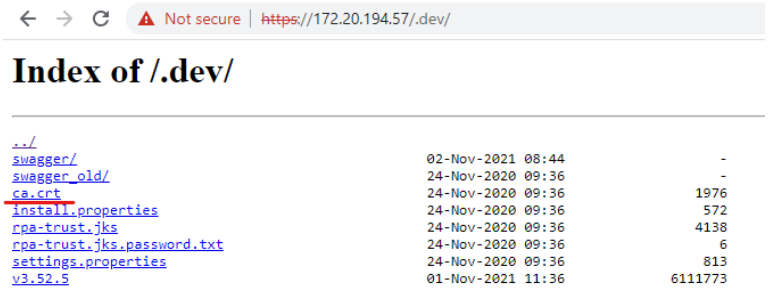
Для правильной связи среды разработки с Канцлер RPA может понадобиться добавить сертификат сервера Канцлер RPA в ваш Java.

Если URL сервера управления, который вы собираетесь использовать, имеет протокол HTTPS и самозаверяющий сертификат (вы можете заметить красное предупреждение “Небезопасно” (“Not Secure”) в браузере Chrome), необходимо выполнить следующие действия:

1. Откройте следующий URL-адрес сервера управления:
   https://{control_server_url}/.dev/

   Обратите внимание, что он доступен только в том случае, если сервер управления был установлен с опцией "dev=true".
   В противном случае этот сервер не должен быть связан со средой разработки и вам необходимо обратиться к системному администратору, который отвечает за установку сервера, чтобы получить этот сертификат.

2. На открывшейся странице скачайте файл с расширением "**.crt**":



3. Загрузите сертификат в хранилище доверенных сертификатов.

```
<JAVA_HOME>/java/bin/keytool -keystore <JAVA_HOME>/java/jre/lib/security/cacerts -import -file ca.crt
-alias kanclerRPA
```

4. Вам будет предложено ввести пароль хранилища ключей.
   Java: начальный пароль файла хранилища ключей "cacerts" - "**changeit**".

5. Если ключ уже существует и/или неверен, вам необходимо удалить этот **псевдоним** (или ввести другой псевдоним на шаге 3):

```
<JAVA_HOME>/java/bin/keytool -delete -alias kanclerRPA -keystore <JAVA_HOME>/java/jre/lib/security
/cacerts
```

Использование образа Канцлер RPA Dev для разработки

1. Установите VirtualBox Install VirtualBox https://www.virtualbox.org/
2. Получите последний образ Канцлер RPA VBox и загрузите его в Virualbox.
3. Запустите только что созданную виртуальную машину и подождите, пока она загрузит приложение.
4. Откройте среду разработки Канцлер RPA Dev по адресу https://127.0.0.1:9443

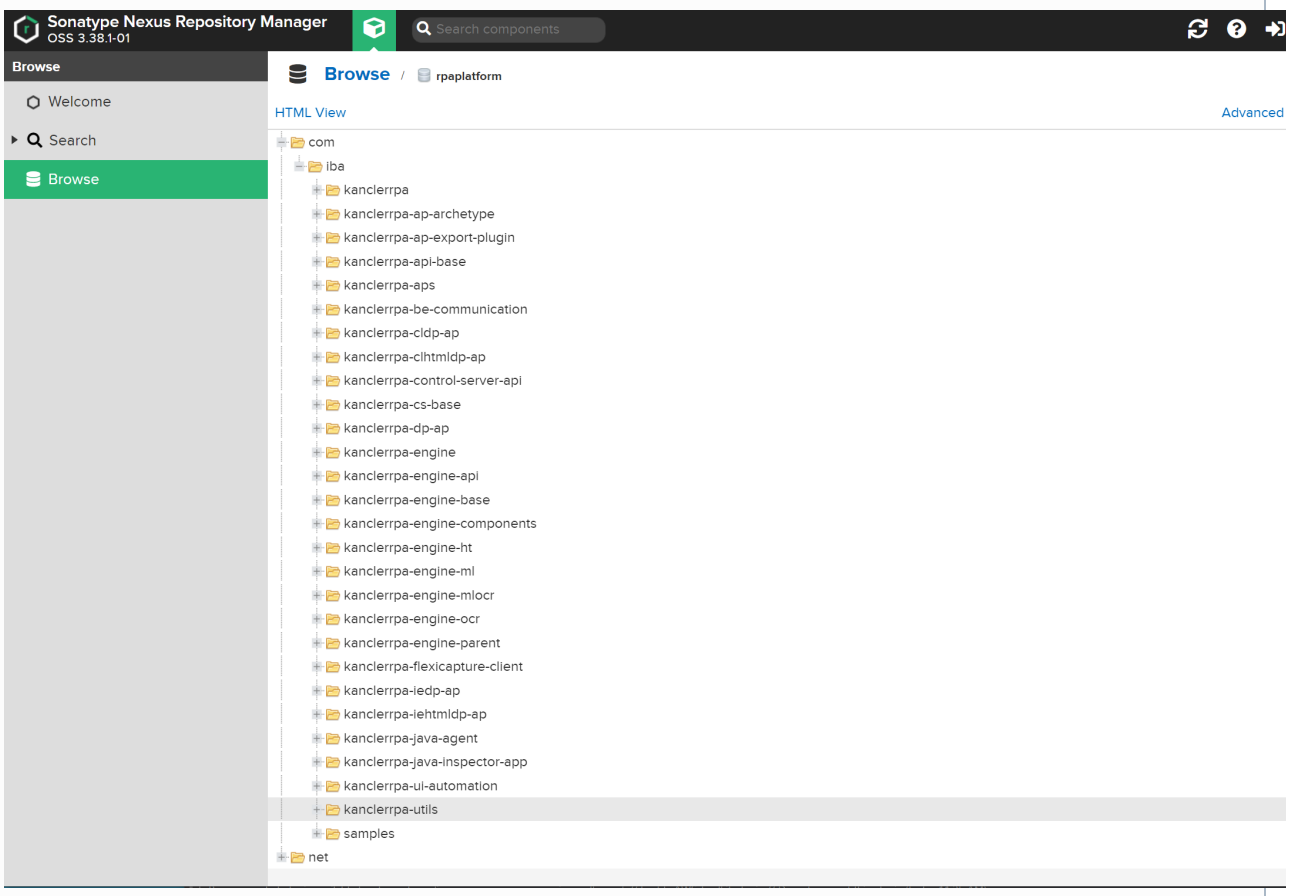
Краткое руководство

Экземпляр сервера управления Канцлер RPA

Все, что необходимо использовать в процессе разработки, поставляется вместе с установкой сервера управления на встроенном Nexus:

URL-адрес Nexus

https://{CS Host}/nexus/#browse/browse:rpaplatform



Обратитесь к Руководству по установке чтобы получить экземпляр сервера управления для разработки или используйте существующий экземпляр, к которому у вас есть доступ.

Настройка Maven для использования репозитория Nexus Канцлер RPA

Откройте файл Maven **settings.xml** (в папке */.m2 в Linux или C:/Users/*ваш пользователь>/.m2 в Windows) и добавьте репозиторий Maven, используя предоставленные шаблоны.

Чтобы развернуть jar-код автоматизированного процесса на Nexus севера управления, необходимо добавить следующий артефакт аутентификации:

```
        <servers>

                . . . .
                <server>
                        <id>rpaplatform-nexus</id>
                        <username>deployment</username>
                        <password>deployment</password>
                </server>

        </servers>
```

Репозиторий Maven сервера управления для доступа к артефактам:

```
        . . . . .
        <profiles>
                <profile>
                        <id>rpaplatform-nexus</id>
                        <repositories>
                                <repository>
                                        <id>rpaplatform-nexus</id>
                                        <url>https://{CS host}/nexus/repository/rpaplatform/</url>
                                </repository>
                        </repositories>
                        <pluginRepositories>
                                <pluginRepository>
                                        <id>rpaplatform-nexus</id>
                                                <url>https://{CS host}/nexus/repository/rpaplatform/</url>
                                </pluginRepository>
                        </pluginRepositories>
                </profile>
        </profiles>
        <activeProfiles>
                <activeProfile>rpaplatform-nexus</activeProfile>
        </activeProfiles>

        . . . .
```

Создание проекта по шаблону

Используя Maven **kanclerrpa-ap-archetype**, можно создать новый проект из шаблона. Запустите мастер с помощью следующей команды:

```
mvn archetype:generate -DarchetypeGroupId=com.iba -DarchetypeArtifactId=kanclerrpa-ap-archetype -
DarchetypeVersion={  RPA} -DgroupId={   } -DartifactId={   } -Dversion={  }
```

Убедитесь, что вы добавили корневой сертификат сервера управления в хранилище доверенных сертификатов Java.

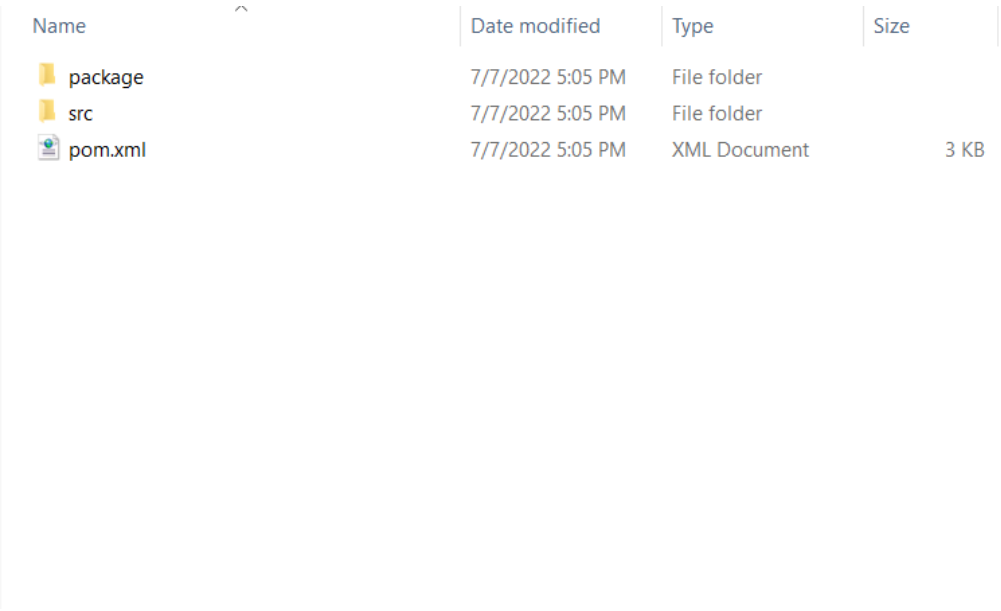
Генератор шаблонов предоставит вам предварительно заполненные значения для генерации проекта:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.2.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[WARNING] Archetype not found in any catalog. Falling back to central repository.
[WARNING] Add a repository with id 'archetype' in your settings.xml if archetype's repository is elsewhere.
[INFO] Using property: automationProcessName = Sample Process
[INFO] Using property: automationProcessDescription = This process automates a lot of work...
[INFO] Using property: addDataStore = true
[INFO] Using property: addDocumentSet = true
[INFO] Using property: addDocumentType = true
[INFO] Using property: addHumanTaskType = true
[INFO] Using property: addModel = true
[INFO] Using property: addNotificationTemplate = true
[INFO] Using property: addSecretVault = true
[INFO] Using property: addStorage = true
[INFO] Using property: addDashboard = true
[INFO] Using property: rpaplatformHost = http://localhost:8081
[INFO] Using property: groupId = in-archetype
[INFO] Using property: artifactId = test
[INFO] Using property: version = 1.0
[INFO] Using property: package = in-archetype
Confirm properties configuration:
automationProcessName: Sample Process
automationProcessDescription: This process automates a lot of work...
addDataStore: true
addDocumentSet: true
addDocumentType: true
addHumanTaskType: true
addModel: true
addNotificationTemplate: true
addSecretVault: true
addStorage: true
addDashboard: true
rpaplatformHost: http://localhost:8081
groupId: in-archetype
artifactId: test
version: 1.0
package: in-archetype
 Y: : Y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: kanclerrpa-ap-archetype:2.5.0-SNAPSHOT
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: in-archetype
[INFO] Parameter: artifactId, Value: test
[INFO] Parameter: version, Value: 1.0
[INFO] Parameter: package, Value: in-archetype
[INFO] Parameter: packageInPathFormat, Value: in-archetype
[INFO] Parameter: rpaplatformHost, Value: http://localhost:8081
[INFO] Parameter: automationProcessName, Value: Sample Process
[INFO] Parameter: addHumanTaskType, Value: true
[INFO] Parameter: addSecretVault, Value: true
[INFO] Parameter: groupId, Value: in-archetype
[INFO] Parameter: addStorage, Value: true
[INFO] Parameter: addDocumentType, Value: true
[INFO] Parameter: addDocumentSet, Value: true
[INFO] Parameter: version, Value: 1.0
[INFO] Parameter: addDashboard, Value: true
[INFO] Parameter: addModel, Value: true
[INFO] Parameter: package, Value: in-archetype
[INFO] Parameter: addNotificationTemplate, Value: true
[INFO] Parameter: artifactId, Value: test
[INFO] Parameter: automationProcessDescription, Value: This process automates a lot of work...
[INFO] Parameter: addDataStore, Value: true
[WARNING] Don't override file C:\Users\Nestsiarchuk_IV\test\src\main\java\in-archetype
[WARNING] Don't override file C:\Users\Nestsiarchuk_IV\test\src\test\java\in-archetype
[WARNING] Don't override file C:\Users\Nestsiarchuk_IV\test\src\test\resources
[WARNING] CP Don't override file C:\Users\Nestsiarchuk_IV\test\package
[WARNING] CP Don't override file C:\Users\Nestsiarchuk_IV\test\package\automationProcess
[INFO] Executing META-INF/archetype-post-generate.groovy post-generation script
[INFO] Project created from Archetype in dir: C:\Users\Nestsiarchuk_IV\test
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  13.074 s
[INFO] Finished at: 2022-07-07T17:05:25+03:00
[INFO] ------------------------------------------------------------------------
```

Вы можете использовать значение по умолчанию или ввести N, чтобы запустить мастер в интерактивном режиме.

Результат будет следующим:



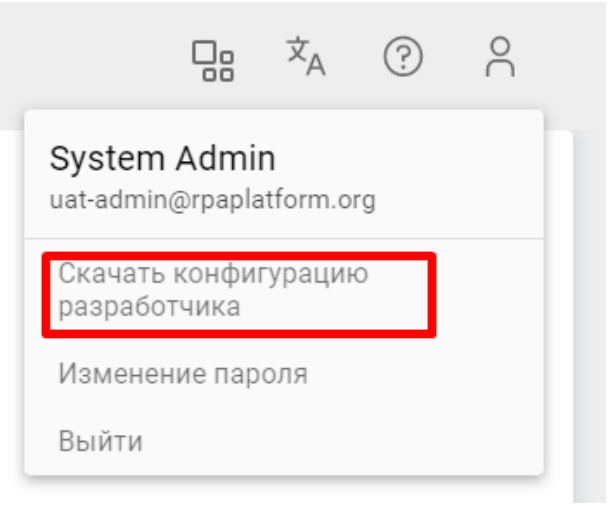
DevelopmentConfigurationModule

Данная конфигурация предназначена для запуска автоматизированных процессов в среде сервера управления. Её основные характеристики:

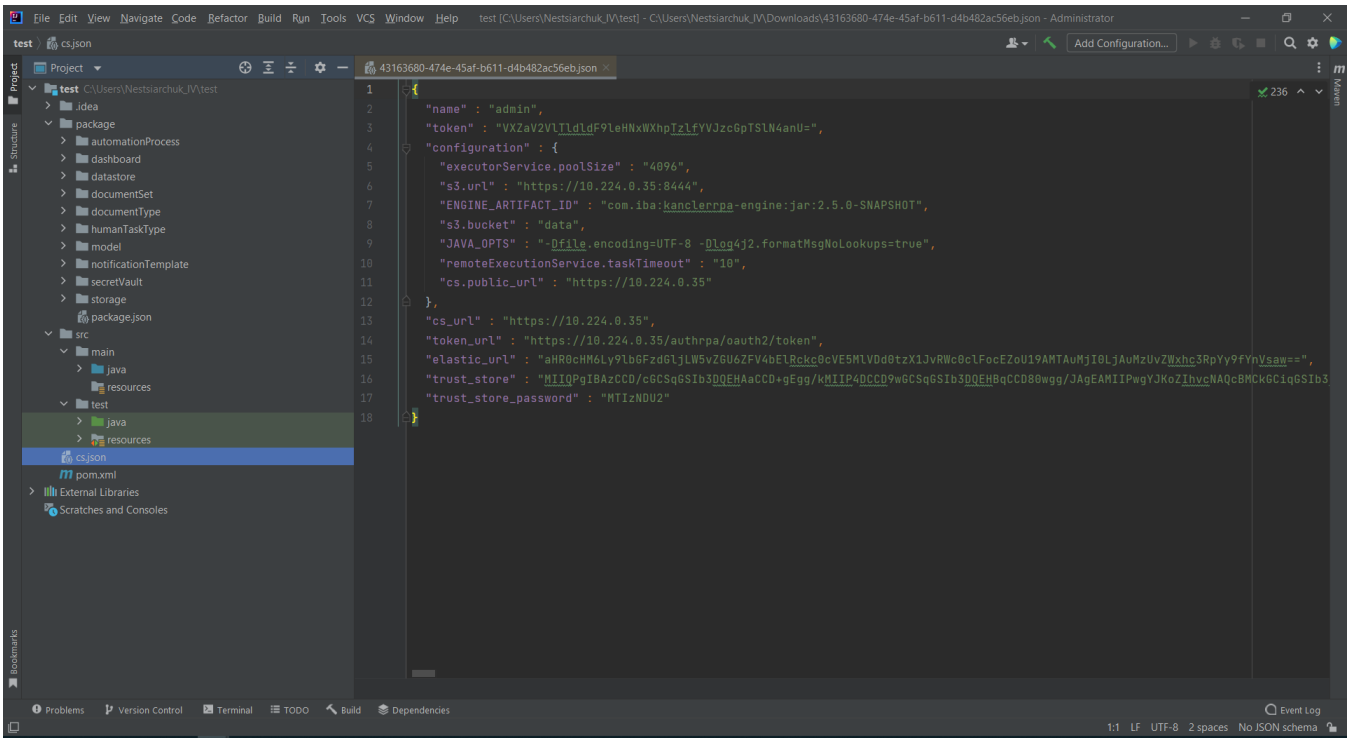
- история задач хранится в файловой системе;
- только локальные журналы;
- локальное хранилище из файла свойств;
- хранилище данных размещено на сервере управления.

Чтобы использовать данную конфигурацию, необходимо связать запуск с экземпляром сервера управления.

Загрузите JSON-файл конфигурации разработки, поместите его в рабочий каталог запуска. Обратите внимание, что пользователи могут загружать конфигурацию разработки только в том случае, если они являются членами группы с правами доступа Разработка-ВЫПОЛНЕНИЕ.



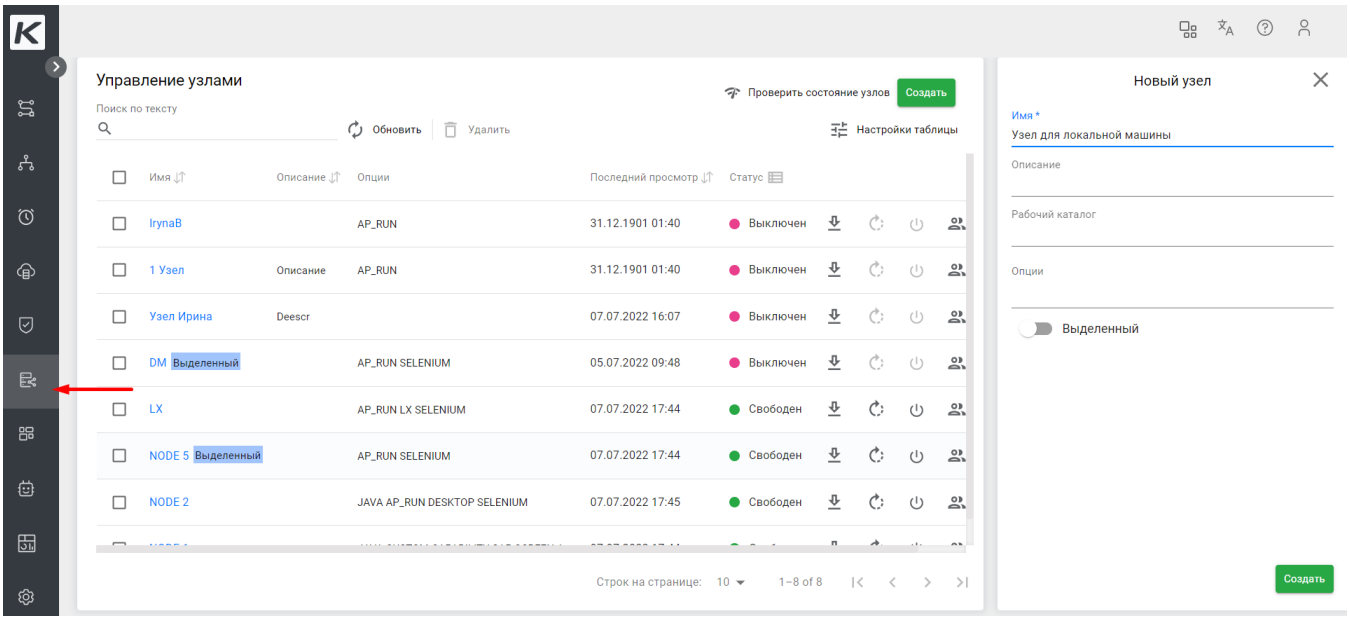
Переименуйте файл в **cs.json** и поместите его в рабочий каталог модуля:



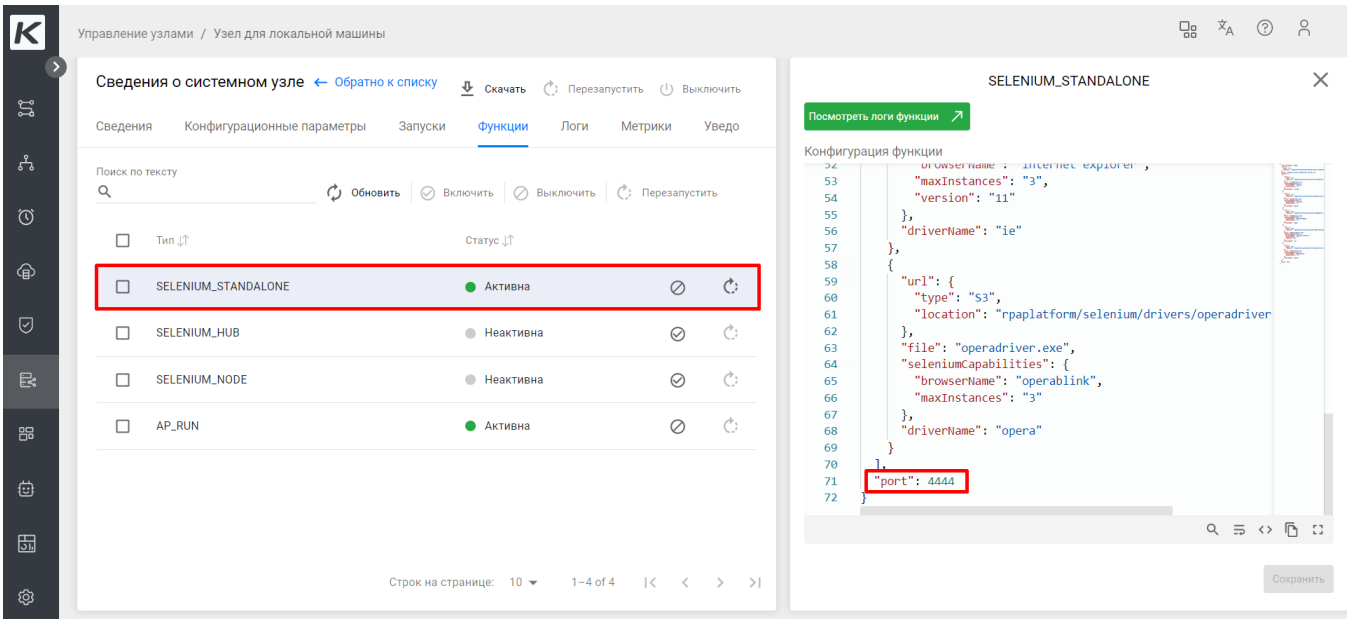
Настройка узла

Вы можете настроить сервер Selenium с помощью Документации по Selenium, либо установить и настроить агент узла Канцлер RPA, что является более простым способом.

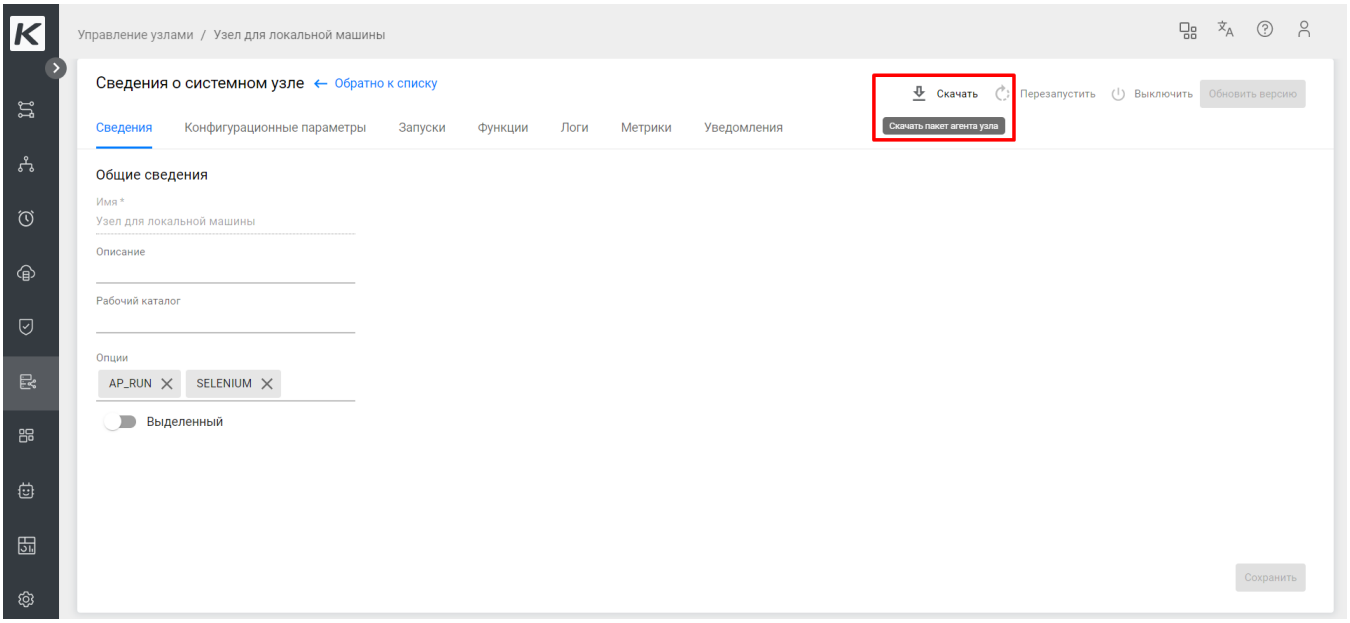
Перейдите на сервер управления, откройте модуль **Управление узлами** и создайте новый узел для своей машины:



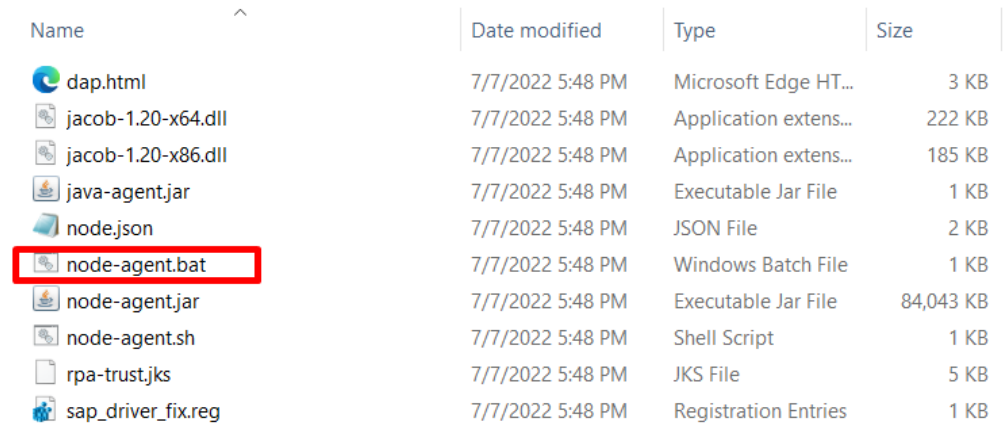
Перейдите на вкладку **Функции узла** и включите SELENIUM_STANDALONE с портом 4444, сделайте его выделенным для защиты вашей машины от запусков сервера управления:



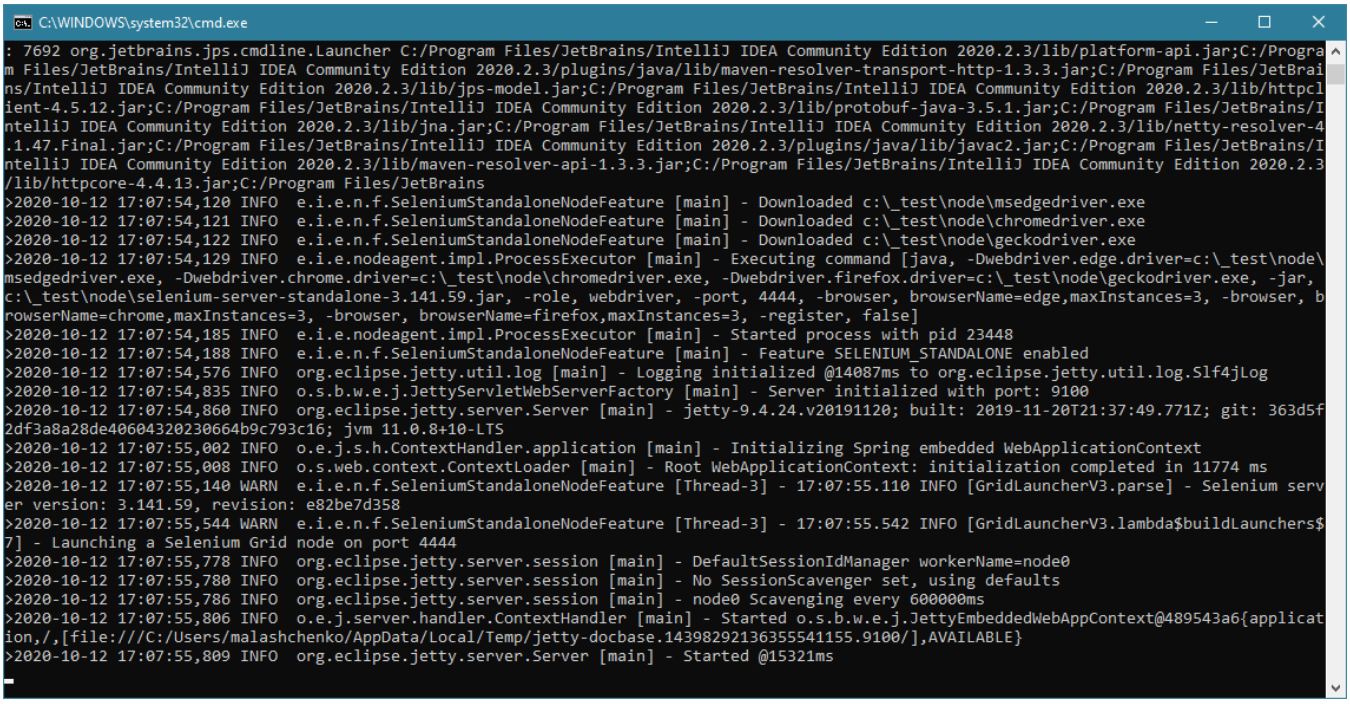
Скачайте пакет агента узла на свой компьютер.



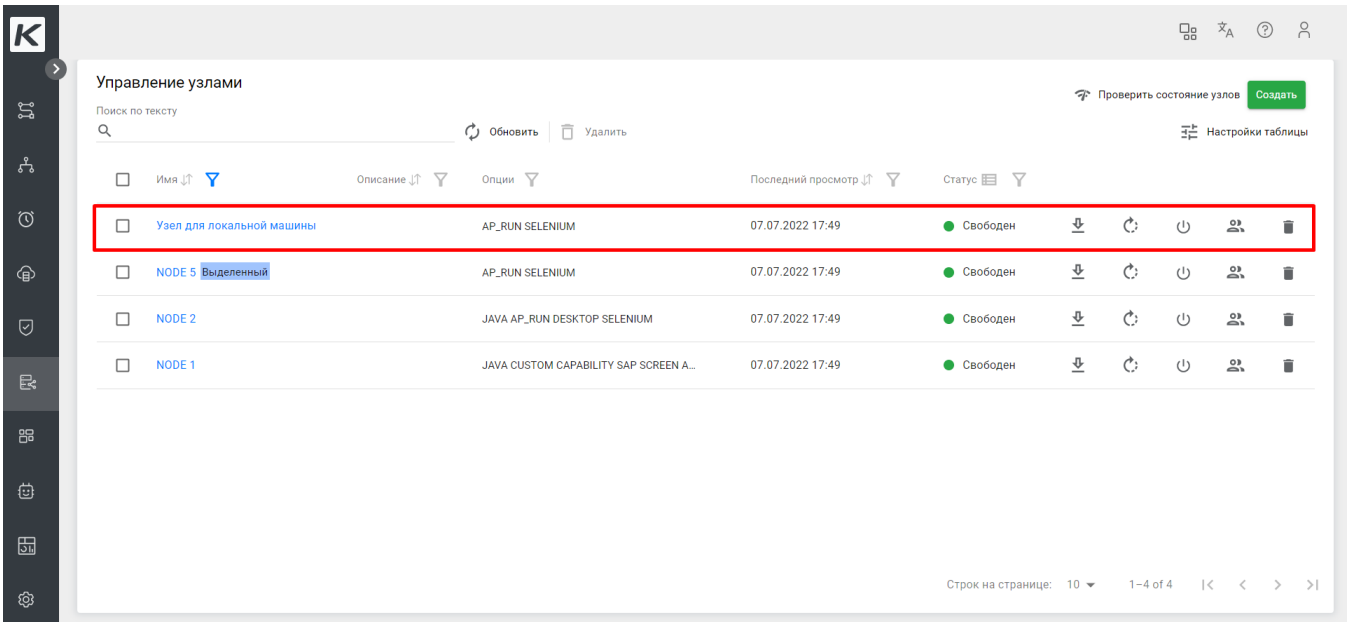
Разархивируйте его и запустите файл **node-agent.bat**:



Name	Date modified	Type	Size
dap.html	7/7/2022 5:48 PM	Microsoft Edge HT...	3 KB
jacob-1.20-x64.dll	7/7/2022 5:48 PM	Application extens...	222 KB
jacob-1.20-x86.dll	7/7/2022 5:48 PM	Application extens...	185 KB
java-agent.jar	7/7/2022 5:48 PM	Executable Jar File	1 KB
node.json	7/7/2022 5:48 PM	JSON File	2 KB
node-agent.bat	7/7/2022 5:48 PM	Windows Batch File	1 KB
node-agent.jar	7/7/2022 5:48 PM	Executable Jar File	84,043 KB
node-agent.sh	7/7/2022 5:48 PM	Shell Script	1 KB
rpa-trust.jks	7/7/2022 5:48 PM	JKS File	5 KB
sap_driver_fix.reg	7/7/2022 5:48 PM	Registration Entries	1 KB



Узел станет доступным для сервера управления:



Теперь вы можете запустить автоматизированный процесс, используя локальное соединение Selenium с уже созданным узлом Канцлер RPA. Робот будет выполнять задачи RPA на компьютере разработчика.

Настройка конфигураций запуска модуля

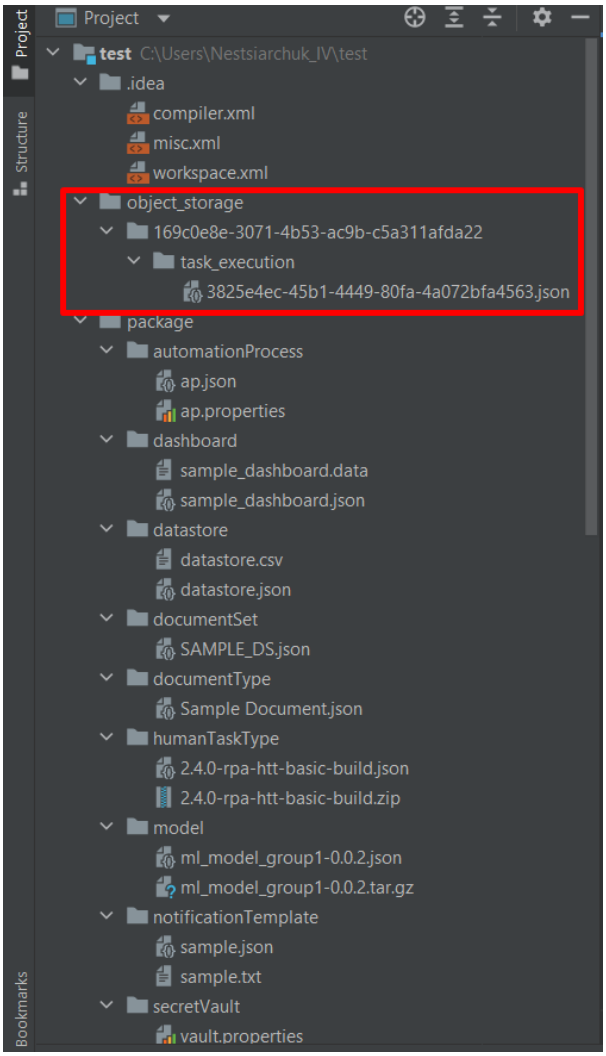
Для разработки доступны 2 возможные конфигурации запуска автоматизированного процесса:

- StandaloneConfigurationModule
- DevelopmentConfigurationModule

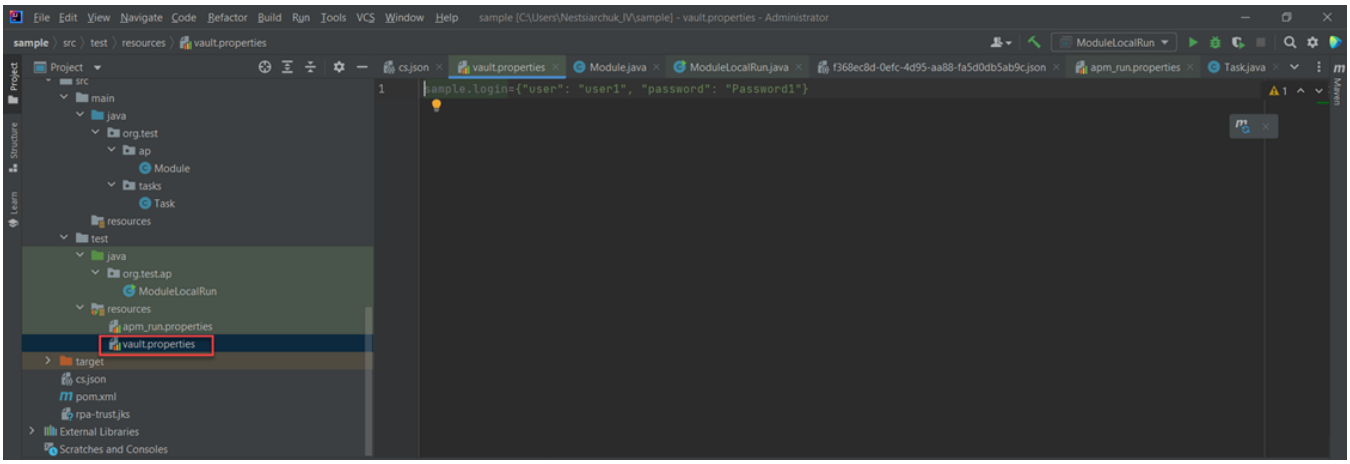
StandaloneConfigurationModule

Это конфигурация по умолчанию. Её основные характеристики:

- история задач хранится в файловой системе;



- только локальные журналы;
- локальное хранилище из файла свойств;



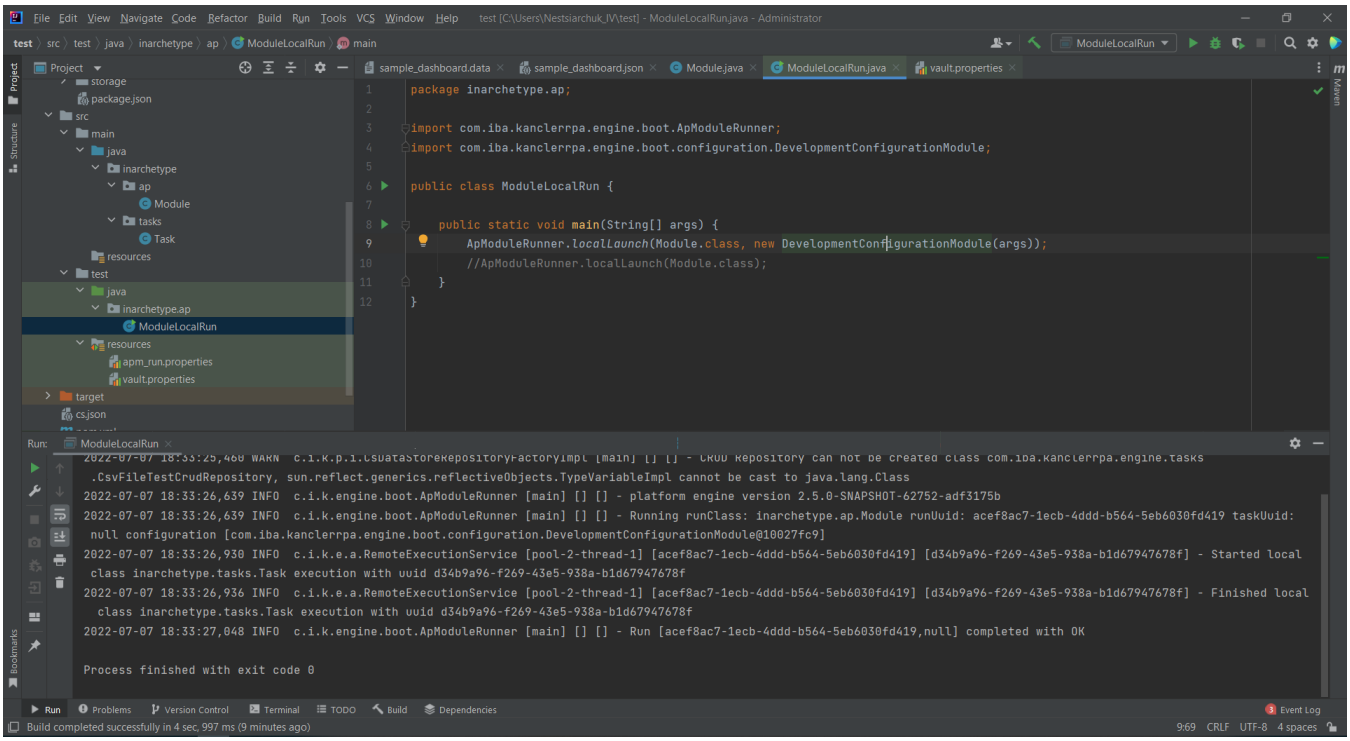
⚠️ Хранилище данных эмулируется в памяти и не поддерживает запросы SQL

Запуск автоматизированного процесса совместно с сервером управления

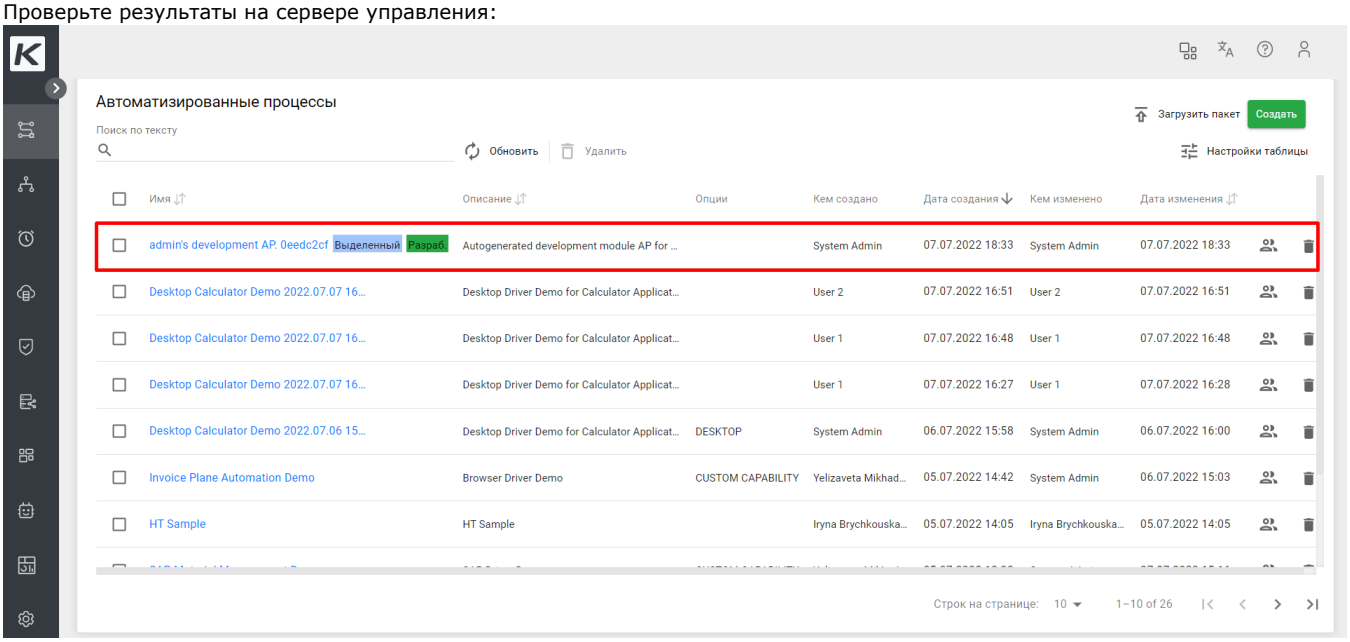
Отмените преобразование в комментарий следующей строки:

Строка для раскомментирования
`ApModuleRunner.localLaunch(Module.class, new DevelopmentConfigurationModule(args));`

и запустите ModuleLocalRun:



Проверьте результаты на сервере управления:



Развертывание сервере управления

Существует два способа развернуть автоматизированный процесс на сервере управления:

1. Развернуть на сервере управления вручную
2. Загрузить zip-архив автоматизированного процесса
3. Использовать плагин экспорта Maven

Развертывание на сервере управления вручную

Разверните jar автоматизированного процесса на Nexus Канцлер RPA.

Пропишите учетные данные репозитория **kanclerrpa-nexus** в файле Maven **settings.xml** , см. выше.

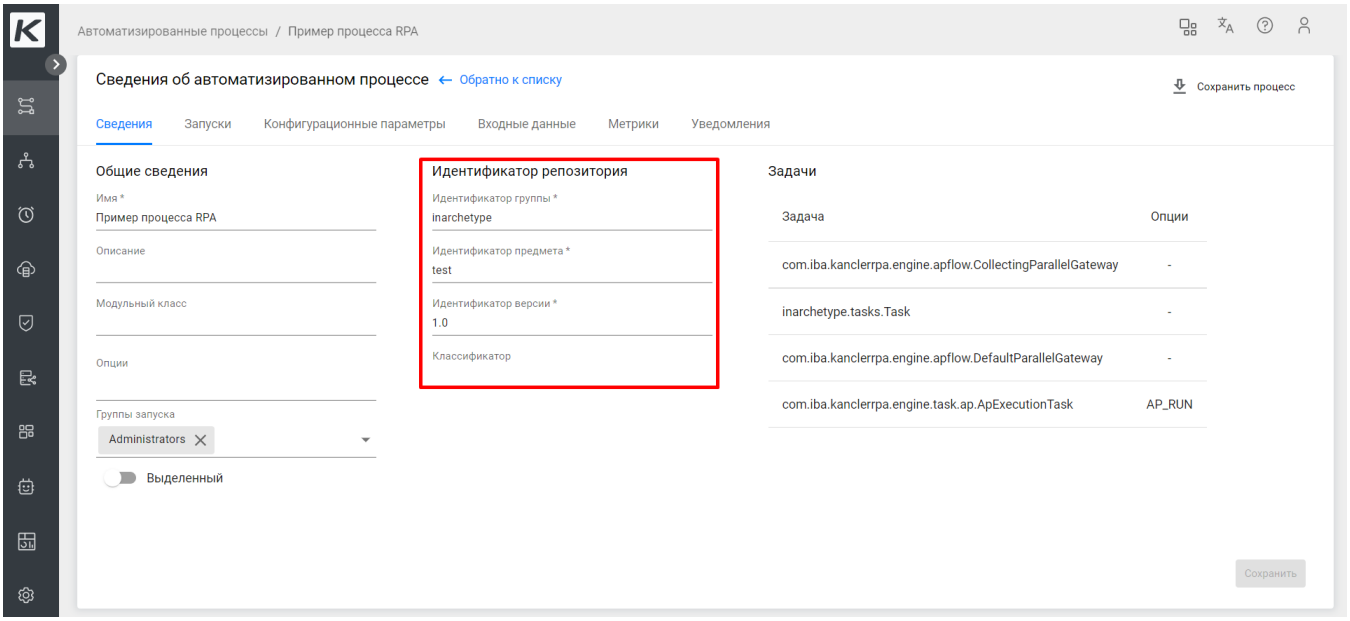
Настройте URL-адрес репозитория Nexus в разделе **DistributionManagement** файла .pom вашего проекта:

```
<distributionManagement>
        <repository>
                <id>rpaplatform-nexus</id>
                <url>https://<kanclerrpa-host>/nexus/repository/aps/</url>
        </repository>
        <snapshotRepository>
                <id>rpaplatform-nexus</id>
                <url>https://<kanclerrpa-host>/nexus/repository/aps/</url>
        </snapshotRepository>
</distributionManagement>
```

Запустите Maven deploy с профилем **nexus-deploy**:

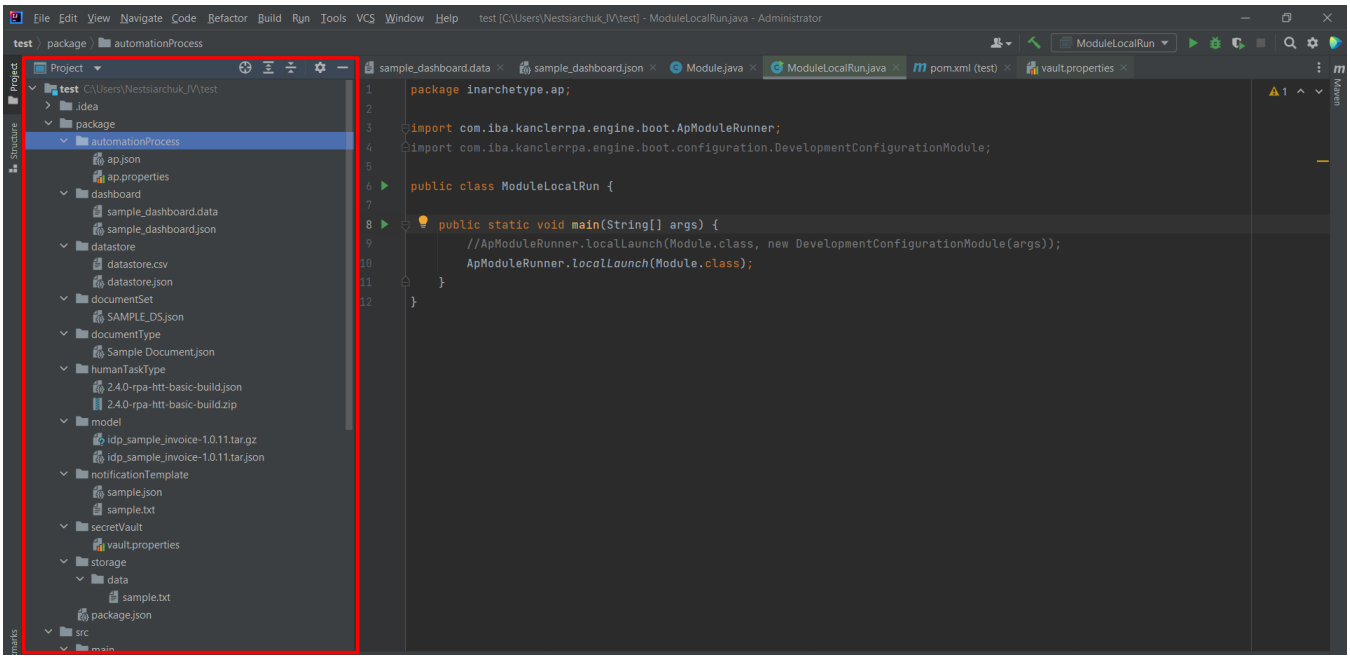
```
mvn clean deploy
```

Перейдите на сервер управления и задайте детали автоматизированого процесса и нажмите кнопку **Создать**.



Загрузка zip-архива автоматизированного процесса

Автоматизированные процессы, хранилища данных, а также секреты можно импортировать из с помощью zip-файла пакета. Его структуру необходимо определить в папке проекта:

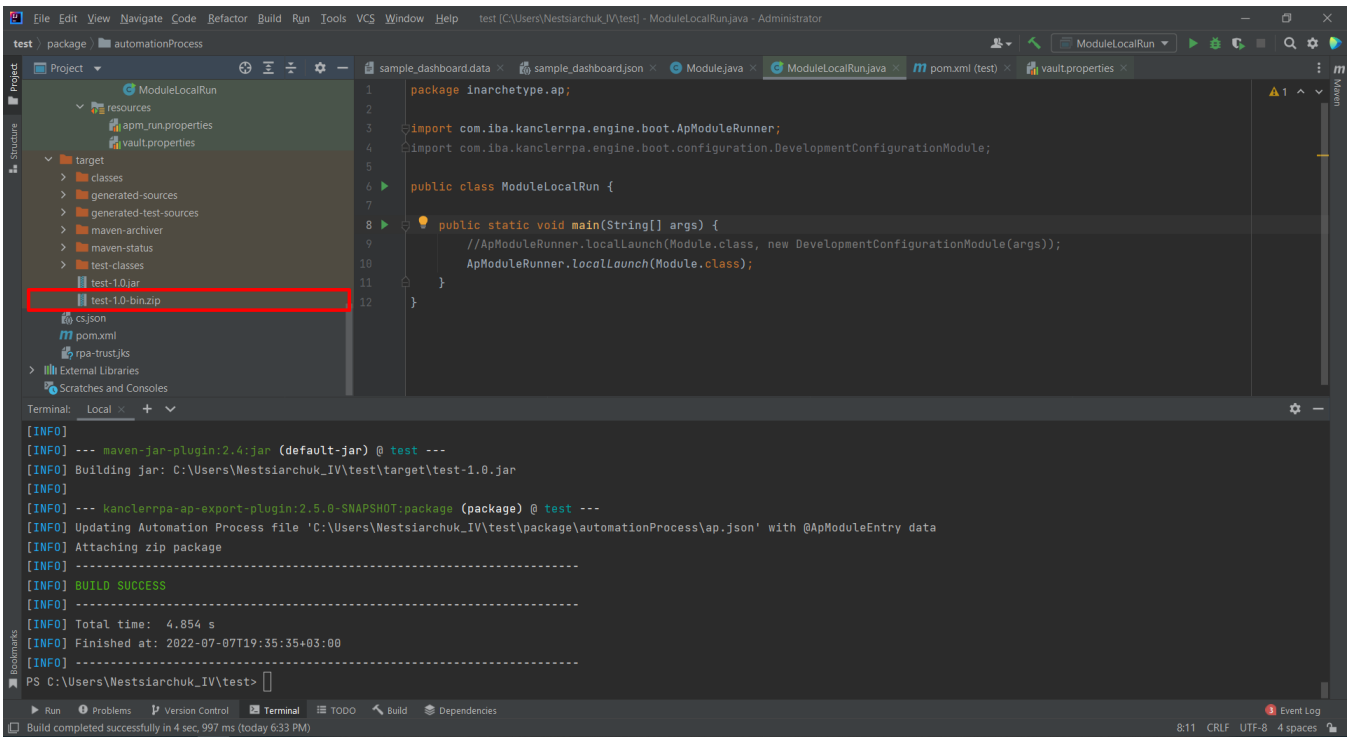


Дополнительные сведения см. в Структура проекта автоматизированного процесса.

Профиль **local-package** предоставляет zip-файл автоматизированного процесса для загрузки на сервер управления, его можно сгенерировать с помощью команды:

```
mvn clean package
```

После запуска команды в **"target"** пакете появится .zip-архив:



Плагин экспорта Maven

Профиль cs-upload можно использовать для сборок CI/CD. Он содержит **maven-assembly-plugin** , который загружает сгенерированный пакет на сервер управления. Более подробную информацию см. в разделе Структура проекта автоматизированного процесса.

Структура проекта автоматизированного процесса

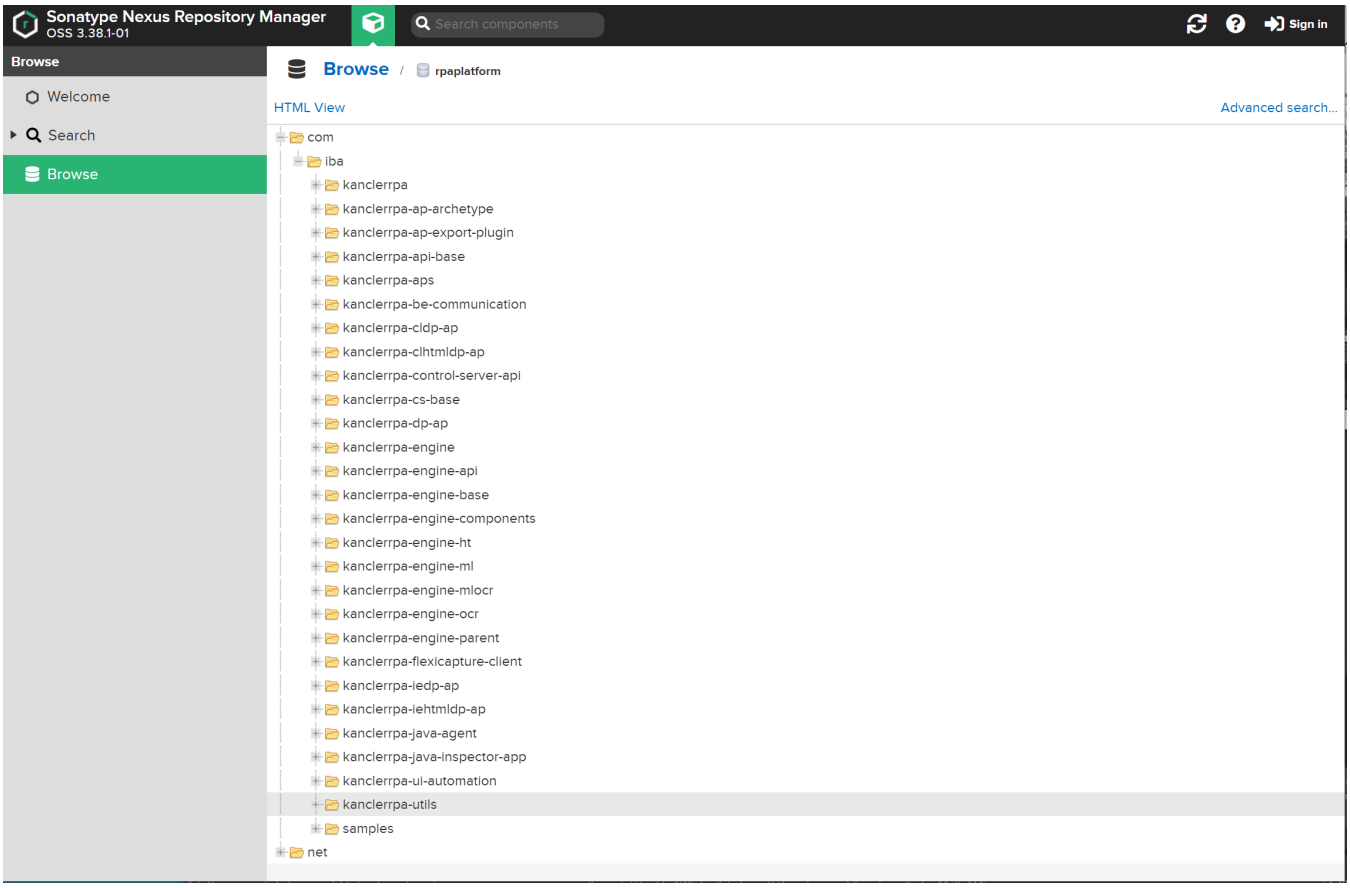
Content

Зависимости

Все, что необходимо использовать в процессе разработки, поставляется вместе с установкой сервера управления на встроенном Nexus:

URL-адрес Nexus

https://{CS Host}/nexus/#browse/browse:rpaplatform



Генерация проекта

Проект может быть сгенерирован из архетипа с использованием интерфейса командной строки или интегрированной среды разработки.

Интерфейс командной строки

Следующая команда создает проект из архетипа:

```
mvn archetype:generate -DarchetypeGroupId=com.iba -DarchetypeArtifactId=kanclerrpa-ap-archetype -
DarchetypeVersion={  RPA} -DgroupId={   } -DartifactId={   } -Dversion={  }
```

В результате будет сгенерирован проект-заглушка.

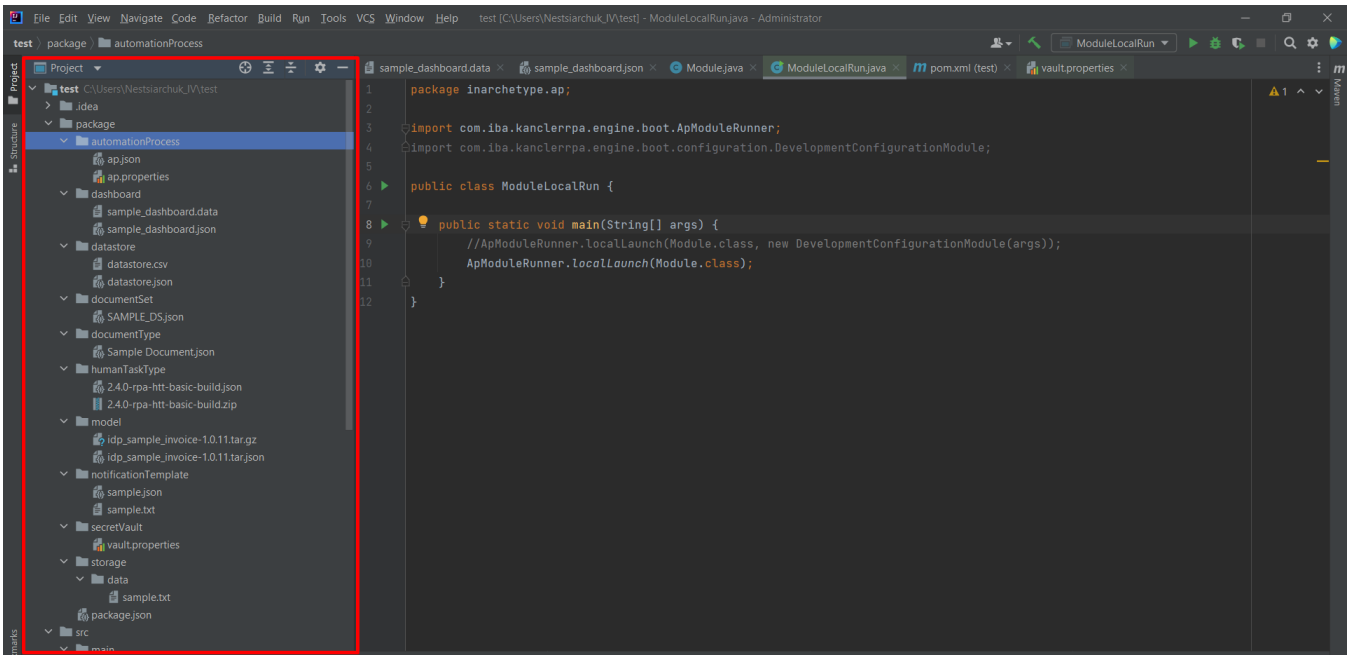
Интегрированная среда разработки

Сначала добавьте удаленный каталог с архетипом *kanclerrpa-ap-archetype* в вашу IDE .

После этого используйте последовательность "Новый проект Проект Maven" ("New project Maven project") и выберите *kancler rpa-ap-archetype* в качестве базового архетипа для создания проекта.

Структура проекта

Сгенерированный проект для автоматизированного процесса представляет собой обычный проект Maven с дополнительной папкой:



Папка **Package** содержит определение артефактов, которые необходимо добавить в пакет автоматизированного процесса — результат развертывания проекта Maven. Сгенерированный пакет можно использовать для импорта на сервер управления Канцлер RPA.

Папка	Содержание	Описание
automationProcess	Автоматизированные процесы для добавления в итоговый пакет.	Содержит следующие файлы для каждого автоматизированного процесса. Файлы одного и того же автоматизированного процесса должны иметь одинаковые имена, но разные расширения.

Файл	Описание	Параметры

*.json
 	
обязательный
	com.iba.kanclerrpa.cs.controller.dto.AutomationProcessImportDto - JSON	Типовой шаблон параметров:
		
		`{`
		`"importStrategy": "OVERRIDE",`
		`"jarImportStrategy": "SKIP",`
		`"name" : "Sample AP",`
		`"description" : "Sample AP",`
		`"repositoryId" : "test.org:sample:jar:0.1`
		`SNAPSHOT",`
		`"moduleClass": "com.ap.Module",`
		`"input": {`
		`"variables": {}`
		`}`
		`}`
		
		где:
		
		• importStrategy - ADD, OVERRIDE, SKIP. Стра
		слияния для автоматического слияния.
		Необязательно. ADD если пропущено.
		• jarImportStrategy - OVERRIDE, SKIP. Страте
		слияния библиотек автоматизированных пр
		при автоматическом слиянии. Необязатель
		пропущено, автоматическое слияние не раз
		• moduleClass - имя класса автоматизированн
		процесса, обязательно.
		• name - имя автоматизированного процесса,
		обазательно. Его можно игнорировать, если
		аннотации @ApModuleEntry автоматизирова
		процесса есть имя, плагин устанавливает ег
		автоматически.
		• description - описание автоматизированного
		автоматизации, необязательно. Его можно
		игнорировать, если в аннотации @ApModule
		автоматизированного процесса автоматизац
		описание, плагин устанавливает его автома
		• repositoryId - обязательный идентификатор
		артефактов автоматизированного процесса
		загрузки с Nexus. Его можно игнорировать,
		автоматически устанавливает его из текуще
		проекта Maven. Для автоматизированных пр
		определенных в разных библиотеках, его н
		установить вручную.
		• input - определяет json входных данных
		автоматизированного процеса. Допустимо т
		переменное поле. Необязательно.

*.properties *необязателен*	файл свойств для параметров автоматизированного процесса соответствующего файла *.json. Параметры могут быть определены в файле JSON, но данный файл добавляет и перезаписывает параметры, определенные в json.	`key=value`

***** Для каждого определенного процесса автоматизации плагин экспорта пытается добавить соответст
файл в папку **lib** итогового пакета с определением класса автоматизированного процесса. Если все
автоматизированные процессы в одном проекте, будет только один jar. Если вы хотите добавить
дополнительные jar-файлы в пакет, вы можете указать, что необходимо добавить в .pom проекта в р
plugins:

```
<build>
        <plugins>
                <plugin>
                        <groupId>com.iba</groupId>
                        <artifactId>kanclerrpa-ap-export-plugin</artifactId>
                        <configuration>
                                <groupId>${project.groupId}|com.iba|com.iba.samples.syst
iba.samples</groupId>
                                <artifactId>kanclerrpa-dp-ap|kanclerrpa-cldp-ap|kanclerr
ap|kanclerrpa-aps|kanclerrpa-idp-ap|kanclerrpa-invoiceplane-system|kanclerrpa-invoicepla
/artifactId>
                        </configuration>
                </plugin>
        </plugins>
</build>
```

где groupId/artifactId — это строка регулярного выражения. Плагин добавляет дополнительные библи
области зависимостей проекта для всех артефактов, которые имеют соответствующий groupId/artifac

dashboard

Панель мониторинга для добавления в итоговый пакет.

Содержит следующие файлы для каждого хранилища данных. Файлы одного и того же хранилища да
должны иметь одинаковое имя, но разные расширения.

Файл	Описание	Параметры

*.json
 	
обязателен
	com.iba.kanclerrpa.cs.controller.dto.DashboardImportDto - JSON	Типовой шаблон параметров:
		
		dashboard.json
		
		`{`
		`  "importStrategy": "OVERRIDE",`
		`  "name": "Sample Dashboard",`
		`  "description": "Sample",`
		`  "settings": {`
		`  "SAMPLE_DS": "${SAMPLE_DS}"`
		`  },`
		`  "dashboardId": "unknown"`
		`}`
		
		где:
		
		• importStrategy - ADD, OVERRIDE, SKIP. Стратегия сли автоматического слияния. Необязательно. ADD, если пропущено.
		• name - имя панели мониторинга, обязательно.
		• description - описание, необязательно.
		• settings - настройки панели мониторинга, необязател
		• dashboardId - ID панели мониторинга в Grafana, обяз

*.data		См. Панели мониторинга платформы.

<table>
<tr>
<td>dataStore</td>
<td>Хранилища данных для добавления в итоговый пакет.</td>
<td>Содержит следующие файлы для каждого хранилища данных. Файлы одного и того же хранилища да должны иметь одинаковое имя, но разные расширения.
<table>
<tr><th>Файл</th><th>Описание</th><th>Параметры</th></tr>
<tr>
<td>*.json

обязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.DataStoreImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
"importStrategy": "OVERRIDE",
"description": "Sample",
"name": "SAMPLE_DS"
}</pre>
где:

• importStrategy - ADD, OVERRIDE, SKIP. Стратег слияния для автоматического слияния. Необяза ADD, если пропущено.
• name - имя хранилища данных, обязательно.
• description - описание, необязательно.</td>
</tr>
<tr>
<td>*.csv

необязателен</td>
<td>содержание хранилища данных</td>
<td>Файл формата CSV:

<pre>index,name,author
1,Book 1,Author 1
2,Book 2,Author 2</pre></td>
</tr>
<tr>
<td>*.format

необязателен</td>
<td>Файл формата CSV, JSON com.iba.kanclerrpa.cs.controller.dto.TableDataFormatDto</td>
<td>Пример кастомизированного шаблона:

<pre>{
"charset": "UTF-8",
"delimiter": ",",
"ignoreEmptyLines": true,
"ignoreSurroundingSpaces": false,
"nullString": null,
"quote": "\"",
"recordSeparator": "\r\n"
}</pre></td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>documentSet</td>
<td>Пакеты документов для добавления в итоговый пакет.</td>
<td>Содержит следующие файлы для каждого пакета документов, файлы одного и того же пакета докуме должны иметь одинаковое имя, но разные расширения.
<table>
<tr><th>Имя</th><th>Описание</th><th>Параметры</th></tr>
<tr>
<td>*.json

обязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.DocumentSetImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
 "importStrategy": "OVERRIDE",
 "name": "SAMPLE_DS",
 "description": "Sample Document Set",
 "documentType": {
 "name": "Sample Document"
  },
 "model": {
 "name": "idp_sample_invoice",
 "version": "1.0.11"
  },
 "settings": {}

}</pre>
где:

• importStrategy - OVERRIDE, SKIP. Стратегия слияния д автоматического слияния. Необязательно, если пропу автоматическое слияние не разрешено.
• name - имя пакета документов, обязательно.
• description - описание, необязательно.
• documentType - ссылка на тип документов пакета доку имени, обязательно.
• model - ссылка на модель, необязательно.
• settings - настройки JSON.</td>
</tr>
<tr>
<td>*.zip

необязателен</td>
<td>Файл архива данных пакета документов - содержимое пакета документов.</td>
<td>Используйте экспорт данных пакета документов для получ файла. См. Скачать пакет данных.</td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>documentType</td>
<td>Типы документов для добавления в итоговый пакет.</td>
<td>Содержит отдельные файлы для каждого типа документов.
<table>
<tr><th>file</th><th>description</th><th>definition</th></tr>
<tr>
<td>*.json

обязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.DocumentTypeImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
 "importStrategy": "OVERRIDE",
 "name": "SAMPLE_DT",
 "description": "Sample Document Type",
 "humanTaskTypeName": "Basic Sample Type",
 "settings": {}
}</pre>
где:

• importStrategy - ADD, OVERRIDE, SKIP. Стратегия сли
автоматического слияния. Необязательно. ADD, если
пропущено.
• name - имя типа документов, обязательно.
• description - описание, необязательно.
• humanTaskTypeName- ссылка на тип пользовательск
по имени, обязательно.
• settings - настройки JSON.</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>model</td>
<td>Модели для добавления в итоговый пакет.</td>
<td>Содержит следующие файлы для каждой модели. Файлы одной и той же модели должны иметь одина
но разные расширения.
<table>
<tr><th>Файл</th><th>Описание</th><th>Параметры</th></tr>
<tr>
<td><имя-версия>.tar.json

необязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.MlModelImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
 "importStrategy": "OVERRIDE",
 "description": "Sample Model"
}</pre>
где:

• importStrategy - OVERRIDE, SKIP. Стратегия слияния для автомати
слияния. Необязательно, если пропущено, автоматическое слияни
разрешено.
• description - описание, необязательно.

Если файл отсутствует, система импортирует модель с gz-файлом без с
стратегии автоматического слияния, их нужно указать вручную в
пользовательском интерфейсе выполнить автоматический импорт в пл
экспорта будет невозможно.</td>
</tr>
<tr>
<td><имя-версия>.tar.gz

обязателен</td>
<td>Файл пакета модели Python.</td>
<td>Используйте опцию экспорта модели для получения файла.</td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>humanTaskType</td>
<td>Типы пользовательских задач для добавления в итоговый пакет.</td>
<td>Содержит следующие файлы для каждого типа пользовательской задачи. Файлы одного и того же ти пользовательской задачи должны иметь одинаковое имя, но разные расширения.
<table>
<tr><th>Файл</th><th>Описание</th><th>Параметры</th></tr>
<tr>
<td>*.json

обязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.HumanTaskTypeImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
 "importStrategy": "OVERRIDE",
 "name": "SAMPLE_DT",
 "description": "Sample Document Type"
}</pre>
где:

• importStrategy - ADD, OVERRIDE, SKIP. Стратегия с для автоматического слияния. Необязательно. ADD пропущено.
• name - имя типа пользовательской задачи, обязате
• description - описание, необязательно.</td>
</tr>
<tr>
<td>*.zip

обязателен</td>
<td>Файл пакета типа пользовательской задачи.</td>
<td>ZIP-архив типа пользовательской задачи.</td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>secretVault</td>
<td>Записи хранилища секретов для добавления в итоговый пакет.</td>
<td>Содержит один файл vault.properties для всех секретов или отдельный файл для каждой записи хран секретов. Если файл vault.properties существует, файлы json будут проигнорированы.
<table>
<tr><th>Файл</th><th>Описание</th><th>Параметры</th></tr>
<tr>
<td>vault.properties

необязателен</td>
<td>Содержит характеристики секретов в формате файла свойств.</td>
<td>CSV-файл:

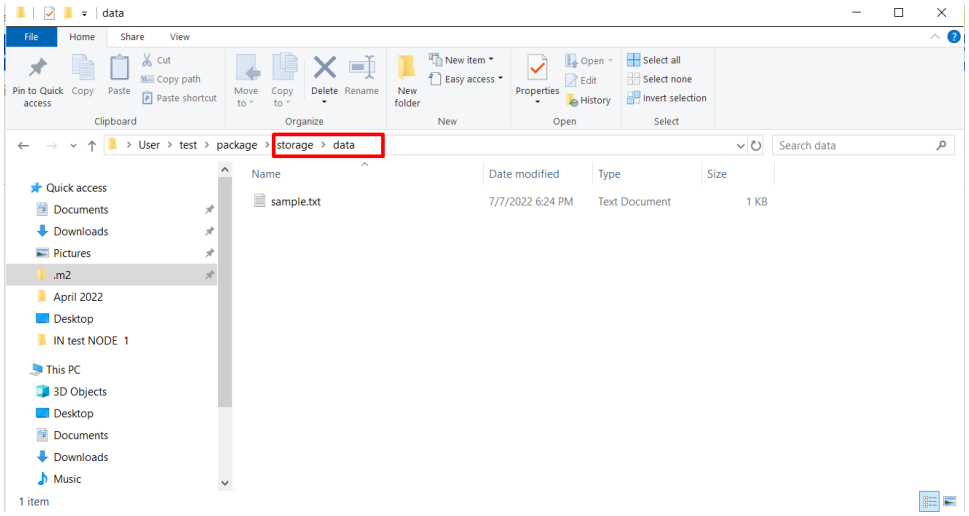
<pre>importStrategy=OVERRIDE
abcLogin={"user": "user1", "password": "Password1"}
key=value</pre>
где:

• importStrategy - ключ зарезервирован - OVERRIDE, SKIP. Ст слияния для автоматического слияния. Необязательно, если пропущено, автоматическое слияние не разрешено.
• значение может быть com.iba.kanclerrpa.engine.model.Secret json в случае использования секрета в качестве учетных да</td>
</tr>
<tr>
<td>*.json

необязателен</td>
<td>com.iba.kanclerrpa.cs.controller.dto.SecretVaultImportDto - JSON</td>
<td>Типовой шаблон параметров:

<pre>{
 "importStrategy": "OVERRIDE",
 "alias": "key",
 "value": "value"
}</pre>
где:

• importStrategy - OVERRIDE, SKIP. Стратегия слияния для автоматического слияния. Необязательно, если пропущено, автоматическое слияние не разрешено.
• alias - алиас секрета, обязательно.
• value - значение секрета, обязательно.</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>storage</td>
<td>Файлы для добавления в хранилище Канцлер RPA.</td>
<td>Каталог должен содержать по крайней мере одну папку уровня, сопоставленную с сегментом. т.е. сл структура добавит файл в папку data данных сегмента:

 
</td>
</tr>
</table>

package.json	Файл метки формата пакета автоматизации.	Пустой JSON

 **Обратите внимание, что только стратегии OVERRIDE и SKIP могут применяться к сущностям .jar, пакетам документов, моделям и записям хранилища секретов. Будьте осторожны при использовании стратегии OVERRIDE для ранее упомянутых сущностей. Даже если в процессе импорта произойдет ошибка, откат изменений будет невозможен, и данные в сегменте будут потеряны для .jar, пакета документов, модели и хранилища секретов, если к ним была применена стратегия OVERRIDE.**

Сборка пакетов

Следующая команда соберет проект и создаст zip-архив со всеми необходимыми файлами, включая jar.

```
mvn clean package
```

Zip-файл проекта будет сгенерирован и помещен в целевой каталог.

Импорт пакета

Есть два способа импортировать пакет на сервер управления: через интерпретатор командной строки (CLI) с помощью встроенного плагина и более традиционный - через пользовательский интерфейс.

Интерфейс командной строки

Чтобы использовать плагин для автоматической загрузки пакетов приложений на сервер управления, необходимо указать путь к файлу конфигурации разработки - cs.json:

```
<build>
        <plugins>
                <plugin>
                        <groupId>com.iba</groupId>
                        <artifactId>kanclerrpa-ap-export-plugin</artifactId>
                        <version>2.5.0-SNAPSHOT</version>
                        <configuration>
                                <config>c:\work\cs.json</config>
                        </configuration>
                </plugin>
        </plugins>
</build>
```

После этого выполните следующую команду, которая соберет проект, создаст zip-архив и загрузит его на сервер управления с указанной вами конфигурацией разработки:

```
mvn clean package deploy -Pcs-upload
```

Обратите внимание, что плагин использует importStrategy, предназначенный для автоматического разрешения конфликтов.

Пользовательский интерфейс

1. Перейдите на страницу **Автоматизированные процессы** и нажмите **Загрузить пакет**.
2. Выберите zip-файл для загрузки.
3. Нажмите **Загрузить**.

Сервер управления ответит отчетом о выполненных им операциях и их статусе. Пользовательский интерфейс для загрузки пакетов по умолчанию использует режим ADD, что обеспечивает создание новых записей.

Возможны два сценария:

1) Ошибок нет, все компоненты успешно созданы (дальнейших действий не требуется)

2) Есть ошибки - некоторые компоненты вышли из строя из-за конфликтов с существующими одноименными. В этом случае просмотрите ошибки и выберите другую стратегию для объединения этих компонентов, например, OVERRIDE или SKIP. Нажмите кнопку **Загрузить**, чтобы выполнить операцию загрузки еще раз.

Обратите внимание, что операция загрузки внутренне состоит из двух независимых транзакций - обновления сервера управления и загрузки jar-файла в Nexus. Загрузка в Nexus не откатывается из-за ограничений Nexus.

Работа с проектом автоматизированного процесса

Использование модуля запуска Канцлер RPA

Запуск процесса с сервера управления

Создайте общий класс, который будет содержать всю структуру бизнес-процесса. Отмет ьте его аннотацией *ApModuleEntry* и сделайте так, чтобы он расширял класс *ApModule*. Добавьте шаги процесса (создание шагов будет описано ниже).

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;

@ApModuleEntry(name = "Hello World")
public class HelloWorld extends ApModule {
    public TaskOutput run() throws Exception {
        return execute(getInput(), HelloStep.class).get();
    }
}
```

Запуск процесса из интегрированной среды разработки

При разработке бизнес-процесса, возникает необходимость протестировать его прямо в среде разработки без создания JAR-файла. Для использования модуля запуска Канцлер RPA, внедрения зависимостей и других функций на этапе отладки, импортируйте класс *ApModuleRunner*. Создайте основной класс и используйте вышеупомянутые классы для выполнения метода запуска в том же классе.

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;

public class ApTestModuleLocalRun {
    public static void main(String[] args) {
        ApModuleRunner.localLaunch(HelloWorld.class);
    }
}
```

Профили запуска

См. Настройка конфигураций запуска модуля.

Добавление логики в бизнес-процесс

Создание шагов процесса

Создайте класс, расширяющий *ApTask.* Отметьте его с помощью аннотации *ApTaskEntry* и добавьте имя и описание задачи в качестве параметров. Ваш класс должен выглядеть примерно следующим образом:

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;

@ApTaskEntry(name = "Step 1", description = "Task that logs Hello World")
public class HelloStep extends ApTask {

    }
}
```

Добавление логики

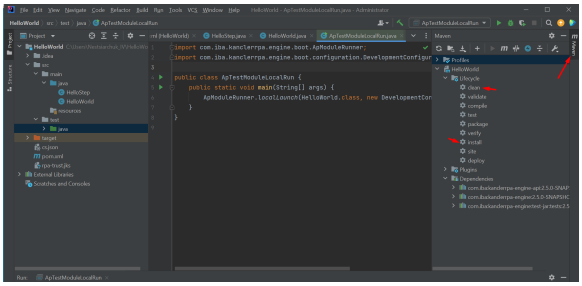
Теперь необходимо добавить логику. Переопределим метод execute(). Это самая простая задача без ввода-вывода, которую можно создать. Сделаем это с целью демонстрации.
Поскольку мы используем lombok, давайте воспользуемся им! Добавьте аннотацию *Slf4j* в свой класс. Это даст нам доступ к регистратору. Запишем сообщение *«Hello World»* в метод *execute* . Наш класс должен выглядеть сейчас следующим образом:

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;

@ApTaskEntry(name = "Step 1", description = "Task that logs Hello World")
@Slf4j
public class HelloStep extends ApTask {
    @Override
    public void execute() {
        log.info("Hello World!");
    }
}
```

Сборка Maven

Для сборки Maven и установления зависимостей необходимо выполнить следующие шаги:

1. Перейдите в представление Maven (справа), выберите свой проект и разверните под ним жизненный цикл.
2. Выполните очистку.
3. Запустите установку.



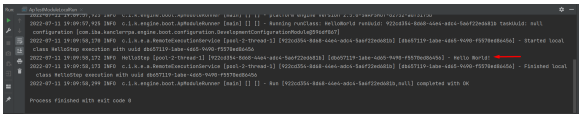
Запуск процесса

⚠ Не забудьте добавить файл *apm_run.properties* в свой рабочий каталог!

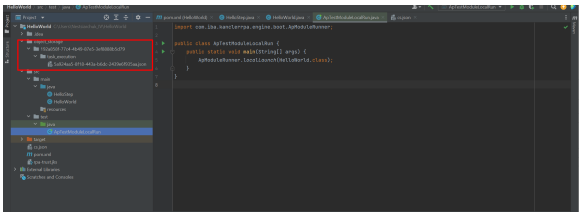
После первого запуска процесса будет создана конфигурация запуска. Выберите **Запуск** **Редактировать конфигурации** (RunEdit Configurations) для изменения настроек. Здесь можно изменить рабочий каталог.

Unable to render {include} The included page could not be found.

Если все правильно, вы должны получить сообщение "Hello World" в окне консоли.



Папка *object_store* будет создана в структуре вашего проекта после первого запуска. Дочерняя папка UUID в *object_store* содержит историю запуска *ExampleProcess*, где UUID - идентификатор процесса. Канцлер RPA проверит, существует ли папка UUID перед запуском, и пропустит запуск процесса, если папка найдена. Вы должны удалить эту папку, чтобы снова запустить процесс *ExampleProcess.*



⚠ Вы можете изменить строку идентификатора UUID на случайно сгенерированное значение с помощью *UUID.randomUUID().toString()*, чтобы избежать пропуска класса *HelloStep.*

Архитектура автоматизированного процесса

В этом разделе описываются компоненты ядра, на которых может быть построен автоматизированный процесс.

- API выполнения задач
- Использование входных и полученных данных
- Аннотации
- Обработка исключений
- Lombok

API выполнения задач

Каждый бизнес-процесс, который вы автоматизируете, содержит одну или несколько задач. **Task** - это специальный модуль, обычно это атомарная бизнес-операция, такая как "Читать почту из почтового ящика" или "Сохранить файл в базу данных". Логика конкретного бизнес-процесса должна быть описана внутри специального модуля, который называется **AP** (автоматизиров анный процесс). **AP** манипулирует задачами, их входными и выхоными данными и управляет потоком.

AP (Автоматизированный процесс)

Модуль **AP** - это специальный класс, который расширяет **ApModule** . Точкой входа класса AP является метод **run** , который содержит высокоуровневую логику бизнес-процесса, управляет задачами, их входными и выходными данными и описывает поток выполнения процесса.

Класс AP также должен быть аннотирован **@ApModuleEntry ,** чтобы предоставить имя автоматизированному процессу, которое будет отображаться на сервере управления.

Ниже приведен пример, как выглядит класс AP:

MyDemoAp.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@ApModuleEntry(name = "My First Demo AP")
public class MyDemoAp extends ApModule {

    public TaskOutput run(){
        //automation process logic here
    }

}
```

Task (Задача)

Модуль **Task** — это специальный класс, расширяющий **ApTask** . Точкой входа класса Task является метод **execute**. Внутри данного метода существует доступ к объектам входных и выходных данных задачи. Как правило Task содержит некоторую атомарную логику бизнес-операций, например "Читать почту из почтового ящика" или "Сохранить файл в базу данных". Также внутри Task вы можете создать экземпляр объекта драйвера для работы с некоторым приложением, которое требует манипуляций с пользовательским интерфейсом.

Классы Task также должны быть аннотированы **@ApTaskEntry ,** чтобы предоставить имя вашей задаче, которое будет отображаться на сервере управления.

Пример класса Task:

MyFirstTask.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;

@ApTaskEntry(name = "My First Task")
@Slf4j
public class MyFirstTask extends ApTask {

    @Override
    public void execute() {
        //task logic here
    }

}
```

Локальное средство запуска

Чтобы протестировать и запустить класс **AP** локально, вам нужно создать **средство запуска (runner)** и предоставить первоначальные входные данные.

Ниже приведен пример класса со статическим методом **main**, который является точкой входа для запуска класса AP:

LocalRunner.java

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import lombok.extern.slf4j.Slf4j;

import java.util.UUID;

@Slf4j
public class LocalRunner {

        public static void main(String[] args) {
                ApModuleRunner.localLaunch(ExecuteAllAsyncDemoAp.class);
        }
}
```

Класс runner должен быть объявлен в test scope, т.к. реализация engine присутствует только там.

Выполнение задач

В следующих примерах мы будем использовать **четыре задачи** для демонстрации различных способов выполнения задач. Они почти одинаковы, разница состоит в том, что первая задача бездействует дольше, чем вторая, вторая бездействует дольше, чем третья, а третья бездействует дольше, чем четвёртая:

Task1.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;


@ApTaskEntry(name = "Task 1")
@Slf4j
public class Task1 extends ApTask {

    private static final int SLEEP_MILLIS = 4000;

    @Override
    public void execute() throws InterruptedException {
        Thread.sleep(SLEEP_MILLIS);
        log.info("Task 1 finished. Provided input: {}. It was sleeping {} millis", getInput().get("test_var"),
SLEEP_MILLIS);
        getOutput().put("test_var", "Output from Task1");
    }
}
```

Task2.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;


@ApTaskEntry(name = "Task 2")
@Slf4j
public class Task2 extends ApTask {

    private static final int SLEEP_MILLIS = 3000;

    @Override
    public void execute() throws InterruptedException {
        Thread.sleep(SLEEP_MILLIS);
        log.info("Task 2 finished. Provided input: {}. It was sleeping {} millis", getInput().get("test_var"),
SLEEP_MILLIS);
        getOutput().put("test_var", "Output from Task2");
    }
}
```

Task3.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;


@ApTaskEntry(name = "Task 3")
@Slf4j
public class Task3 extends ApTask {

    private static final int SLEEP_MILLIS = 2000;

    @Override
    public void execute() throws InterruptedException {
        Thread.sleep(SLEEP_MILLIS);
        log.info("Task 3 finished. Provided input: {}. It was sleeping {} millis", getInput().get("test_var"),
SLEEP_MILLIS);
        getOutput().put("test_var", "Output from Task3");
    }
}
```

Task4.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;


@ApTaskEntry(name = "Task 4")
@Slf4j
public class Task4 extends ApTask {

    private static final int SLEEP_MILLIS = 1000;

    @Override
    public void execute() throws InterruptedException {
        Thread.sleep(SLEEP_MILLIS);
        log.info("Task 4 finished. Provided input: {}. It was sleeping {} millis", getInput().get("test_var"),
SLEEP_MILLIS);
        getOutput().put("test_var", "Output from Task4");
    }
}
```

execute

Все методы **execute** запускают выполнение задач асинхронно, поэтому выполнение задачи будет запущено, но основной поток AP не будет ждать результата и продолжит обработку следующей строки кода. Это может быть полезно в том случае, когда нет необходимости в немедленном получении результатов задачи, и автоматизированный процесс может продолжать выполнение последующих задач.

Вместо **TaskOutput** этот метод возвращает объект **CompletableFuture<TaskOutput>** . Чтобы получить выходные данные данного объекта, необходимо вызвать метод **get()**. Метод **get()** немедленно **вернет TaskOutput** , если **задача уже выполнена**. Он **заблокирует ваш основной поток автоматизированного процесса** и **будет ждать завершения задачи,** если **задача еще не завершена.**

execute

```
CompletableFuture<TaskOutput> execute(TaskInput input, Class<? extends ApTask>... task)
CompletableFuture<TaskOutput> execute(TaskInput input, ExecutionService.MergeStrategy mergeStrategy, Class<?
extends ApTask>... task)
CompletableFuture<TaskOutput> execute(TaskOutput output, Class<? extends ApTask>... task)
CompletableFuture<TaskOutput> execute(TaskOutput prevTaskOutput, ExecutionService.MergeStrategy mergeStrategy,
Class<? extends ApTask>... classes)
Function<TaskOutput, CompletableFuture<TaskOutput>> execute(Class<? extends ApTask>... classes)
Function<TaskOutput, CompletableFuture<TaskOutput>> execute(ExecutionService.MergeStrategy mergeStrategy,
Class<? extends ApTask>... classes)
```

- **input** - объект входных данных, полученный от локального средства запуска в случае локального выполнения, или объект входных данных, предоставленный на сервере управления в случае *executiomergeStrategyn* на сервере.
- **task** - класс задачи, которую нужно выполнить, например **MyFirstTask.class**.
- **mergeStrategy** - объект BiConsumer (обычно лямбда-выражение), в котором можно определить стратегию слияния.
- **output** - объект **TaskOutput**, который позволяет использовать выходные данные ранее выполненной задачи в качестве входных данных для следующей задачи.

Синхронное выполнение задач

Это простой способ выполнить одну задачу с некоторыми входными данными.

execute

```
CompletableFuture<TaskOutput> execute(TaskInput input, Class<? extends ApTask>... task)
```

где:

- **input** - объект входных данных, полученный от локального средства запуска в случае локального выполнения, или объект входных данных, предоставленный на сервере управления в случае *executiomergeStrategyn* на сервере.
- **task** - класс задачи, которую нужно выполнить, например **MyFirstTask.class**.

Чтобы получить выходные данные из этого объекта, необходимо вызвать метод **get().**

Также вместо объекта **TaskInput** этот метод может принимать объект **TaskOutput** , который позволяет использовать выходные данные ранее выполненной задачи в качестве входных данных для следующей задачи:

```
CompletableFuture execute(TaskOutput prevTaskOutput, Class<? extends ApTask>... task)
```

Ниже приводится пример автоматизированного процесса, который выполняет четыре задачи, и в котором все выходные данные используются в качестве входных данных для следующих задач:

ExecuteDemoAp.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@ApModuleEntry(name = "My First Demo AP")
public class MyDemoAp extends ApModule {
    public TaskOutput run() throws Exception {
        log.info("execute method test started");
        TaskOutput previousOutput = execute(getInput(), Task1.class).get();
        previousOutput = execute(previousOutput, Task2.class).get();
        previousOutput = execute(previousOutput, Task3.class).get();
        previousOutput = execute(previousOutput, Task4.class).get();

        log.info("execute method test finished. Final output: {}", previousOutput.get("test_var"));
        return previousOutput;
    }
}
```

Логи по завершению выпонения выглядят следующим образом:

```
2022-07-11 19:28:29,028 INFO  c.i.k.taskexecution.ExecuteDemoAp [main] [] [] - execute method test started
2022-07-11 19:28:29,105 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [d12d2cf7-833c-4743-9028-0e26ea25ffa7] - Started local class com.iba.kanclerrpa.taskexecution.
task.Task1 execution with uuid d12d2cf7-833c-4743-9028-0e26ea25ffa7
2022-07-11 19:28:33,107 INFO  c.i.k.taskexecution.task.Task1 [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [d12d2cf7-833c-4743-9028-0e26ea25ffa7] - Task 1 finished. Provided input: null. It was sleeping
4000 millis
2022-07-11 19:28:33,115 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [d12d2cf7-833c-4743-9028-0e26ea25ffa7] - Finished local class com.iba.kanclerrpa.taskexecution.
task.Task1 execution with uuid d12d2cf7-833c-4743-9028-0e26ea25ffa7
2022-07-11 19:28:33,192 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [8b547340-8cd2-4714-bee3-c9f48e002ddd] - Started local class com.iba.kanclerrpa.taskexecution.
task.Task2 execution with uuid 8b547340-8cd2-4714-bee3-c9f48e002ddd
2022-07-11 19:28:36,193 INFO  c.i.k.taskexecution.task.Task2 [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [8b547340-8cd2-4714-bee3-c9f48e002ddd] - Task 2 finished. Provided input: Output from Task1. It
was sleeping 3000 millis
2022-07-11 19:28:36,194 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [8b547340-8cd2-4714-bee3-c9f48e002ddd] - Finished local class com.iba.kanclerrpa.taskexecution.
task.Task2 execution with uuid 8b547340-8cd2-4714-bee3-c9f48e002ddd
2022-07-11 19:28:36,204 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-3] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [49820ffc-84d3-4d3a-9d94-e5348d31e048] - Started local class com.iba.kanclerrpa.taskexecution.
task.Task3 execution with uuid 49820ffc-84d3-4d3a-9d94-e5348d31e048
2022-07-11 19:28:38,206 INFO  c.i.k.taskexecution.task.Task3 [pool-1-thread-3] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [49820ffc-84d3-4d3a-9d94-e5348d31e048] - Task 3 finished. Provided input: Output from Task2. It
was sleeping 2000 millis
2022-07-11 19:28:38,207 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-3] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [49820ffc-84d3-4d3a-9d94-e5348d31e048] - Finished local class com.iba.kanclerrpa.taskexecution.
task.Task3 execution with uuid 49820ffc-84d3-4d3a-9d94-e5348d31e048
2022-07-11 19:28:38,215 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [ce7b6487-e367-44aa-81c8-5756c6866a64] - Started local class com.iba.kanclerrpa.taskexecution.
task.Task4 execution with uuid ce7b6487-e367-44aa-81c8-5756c6866a64
2022-07-11 19:28:39,217 INFO  c.i.k.taskexecution.task.Task4 [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [ce7b6487-e367-44aa-81c8-5756c6866a64] - Task 4 finished. Provided input: Output from Task3. It
was sleeping 1000 millis
2022-07-11 19:28:39,217 INFO  c.i.k.e.a.RemoteExecutionService [pool-1-thread-1] [ec4116d6-9929-4f1c-95ee-
d4df9f5d83eb] [ce7b6487-e367-44aa-81c8-5756c6866a64] - Finished local class com.iba.kanclerrpa.taskexecution.
task.Task4 execution with uuid ce7b6487-e367-44aa-81c8-5756c6866a64
2022-07-11 19:28:39,221 INFO  c.i.k.taskexecution.ExecuteDemoAp [main] [] [] - execute method test finished.
Final output: Output from Task4
```

Асинхронное выполнение задач

Ниже приводится пример автоматизированного процесса, который выполняет четыре задачи асинхронно и выводит результаты каждой из них по отдельности:

ExecuteAsyncDemoAp.java

```
package com.iba.kanclerrpa.taskexecution;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.taskexecution.task.Task1;
import com.iba.kanclerrpa.taskexecution.task.Task2;
import com.iba.kanclerrpa.taskexecution.task.Task3;
import com.iba.kanclerrpa.taskexecution.task.Task4;
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.CompletableFuture;

@Slf4j
@ApModuleEntry(name = "Execute Async Demo")
public class ExecuteAsyncDemoAp extends ApModule {

        public static void main(String[] args) {
                ApModuleRunner.localLaunch(ExecuteAsyncDemoAp.class);
        }

        public TaskOutput run() throws Exception {
                log.info("executeAsync method test started");

                CompletableFuture<TaskOutput> completableFuture1 = execute(getInput(), Task1.class);
                CompletableFuture<TaskOutput> completableFuture2 = execute(getInput(), Task2.class);
                CompletableFuture<TaskOutput> completableFuture3 = execute(getInput(), Task3.class);
                CompletableFuture<TaskOutput> completableFuture4 = execute(getInput(), Task4.class);

                log.info("executeAsync method test finished. Final output Task1: {}; Task2: {}; Task3: {};
Task4: {}", completableFuture1.get().get("test_var"),
                completableFuture2.get().get("test_var"), completableFuture3.get().get("test_var"),
completableFuture4.get().get("test_var"));

                return completableFuture4.get();
        }
}
```

Логи по завершению выпонения выглядят следующим образом:

```
2020-08-26 19:37:55,503 INFO e.i.e.t.ExecuteAsyncDemoAp [main] - executeAsync method test started
2020-08-26 19:37:56,574 INFO  e.i.kanclerrpa.taskexecution.task.Task4 [pool-1-thread-4] - Task 4 finished.
Provided input: null. It was sleeping 1000 millis
2020-08-26 19:37:57,574 INFO  e.i.kanclerrpa.taskexecution.task.Task3 [pool-1-thread-3] - Task 3 finished.
Provided input: Output from Task4. It was sleeping 2000 millis
2020-08-26 19:37:58,574 INFO  e.i.kanclerrpa.taskexecution.task.Task2 [pool-1-thread-2] - Task 2 finished.
Provided input: Output from Task3. It was sleeping 3000 millis
2020-08-26 19:37:59,573 INFO  e.i.kanclerrpa.taskexecution.task.Task1 [pool-1-thread-1] - Task 1 finished.
Provided input: Output from Task2. It was sleeping 4000 millis
2020-08-26 19:37:59,578 INFO e.i.e.t.ExecuteAsyncDemoAp [main] - executeAsync method test finished. Final
output Task1: Output from Task1; Task2: Output from Task2; Task3: Output from Task3; Task4: Output from Task4
```

Помимо этого, **CompletableFuture** API предоставляет возможность объединять запуски в цепочки, где выходные данные предыдущей задачи будут автоматически использоваться в качестве входных данных для следующей задачи. Чтобы использовать это, нужно вызвать метод **thenCompose** из объекта **CompletableFuture** и предоставить методы **executeAsync**, которые возвращают объект **java.util.function.Function .** Для этого необходимо использовать следующие сигнатуры методов:

```
Function<TaskOutput, CompletableFuture<TaskOutput>> execute(Class<? extends ApTask>... classes)
```

Как видите, единственное отличие состоит в том, что эти методы не принимают input в качестве аргумента, потому что он будет предоставлен автоматически.

Пример его использования:

```
 CompletableFuture<TaskOutput> finalCompletableFuture = execute(getInput(), Task1.class)
                .thenCompose(execute(Task2.class))
                .thenCompose(execute(Task3.class))
                .thenCompose(execute(Task4.class));
```

Ниже приводится пример автоматизированного процесса, который выполняет четыре задачи, где выходные данные предыдущей задачи автоматически используются в качестве входных данных для следующей задачи. В этом случае поведение выполнения будет таким же, как и в методе **execute**, в котором цепочку входов/выходов необходимо проставить вручную:

ExecuteAsyncChainDemoAp.java

```
@Slf4j
@ApModuleEntry(name = "Execute Async Chain Demo")
public class ExecuteAsyncChainDemoAp extends ApModule {

        public static void main(String[] args) {
                ApModuleRunner.localLaunch(ExecuteAsyncChainDemoAp.class);
        }

        public TaskOutput run() throws Exception {
                log.error("executeAsync method chain test started");

                CompletableFuture<TaskOutput> finalCompletableFuture = execute(getInput(), Task1.class)
                .thenCompose(execute(Task2.class))
                .thenCompose(execute(Task3.class))
                .thenCompose(execute(Task4.class));

                TaskOutput finalOutput = finalCompletableFuture.get();

                log.error("executeAsync method chain test finished. Final output: {}; ", finalOutput.get
("test_var"));

                return finalOutput;
        }
}
```

Логи по завершению выпонения выглядят следующим образом:

```
2020-08-26 19:51:07,212 INFO e.i.e.t.ExecuteAsyncChainDemoAp [main] - executeAsync method chain test started
2020-08-26 19:51:11,268 INFO  e.i.kanclerrpa.taskexecution.task.Task1 [pool-1-thread-1] - Task 1 finished.
Provided input: null. It was sleeping 4000 millis
2020-08-26 19:51:14,284 INFO  e.i.kanclerrpa.taskexecution.task.Task2 [pool-1-thread-2] - Task 2 finished.
Provided input: Output from Task1. It was sleeping 3000 millis
2020-08-26 19:51:16,298 INFO  e.i.kanclerrpa.taskexecution.task.Task3 [pool-1-thread-1] - Task 3 finished.
Provided input: Output from Task2. It was sleeping 2000 millis
2020-08-26 19:51:17,312 INFO  e.i.kanclerrpa.taskexecution.task.Task4 [pool-1-thread-2] - Task 4 finished.
Provided input: Output from Task3. It was sleeping 1000 millis
2020-08-26 19:51:17,317 INFO e.i.e.t.ExecuteAsyncChainDemoAp [main] - executeAsync method chain test finished.
Final output: Output from Task4;
```

Асинхронное выполнение нескольких задач

Этот метод можно использовать для запуска асинхронного выполнения нескольких задач с одним и тем же input. Он имеет следующую сигнатуру метода:

```
CompletableFuture<TaskOutput> execute(TaskOutput output, Class<? extends ApTask>... task)
```

где:

- **input** - объект входных данных, полученный от локального средства запуска в случае локального выполнения, или объект входных данных, предоставленный на сервере управления в случае выполнения на сервере.
- **task**- классы задач, которые нужно выполнить, например **MyFirstTask.class, MySecondTask.class**.

Возвращает: объект **CompletableFuture** для которого вы можете вызвать метод **get(),** чтобы получить объект **TaskOutput**.

Пример использования:

```
CompletableFuture<TaskOutput> completableFutureCombined = execute(getInput(), Task1.class, Task2.class, Task3.
class, Task4.class);
```

Окончательный **вывод** будет результатом **слияния всех выходных данных** . По умолчанию существует стратегия слияния, которая в случае конфликта (выходная переменная с одинаковым именем) - последняя задача, предусмотренная для этого метода, перезапишет продублированную переменную.

Рассмотрим следующий пример автоматизированного процесса:

ExecuteAllAsyncDemoAp.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModuleBase;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.taskexecution.task.Task1;
import com.iba.kanclerrpa.taskexecution.task.Task2;
import com.iba.kanclerrpa.taskexecution.task.Task3;
import com.iba.kanclerrpa.taskexecution.task.Task4;
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.CompletableFuture;

@Slf4j
@ApModuleEntry(name = "Execute All Async Demo")
public class ExecuteAllAsyncDemoAp extends ApModuleBase {

    public TaskOutput run() throws Exception {
        log.info("executeSubtasks method test started");
        CompletableFuture<TaskOutput> completableFutureCombined = executeSubtasks(getInput(), Task1.class,
Task2.class, Task3.class, Task4.class);
        TaskOutput output = completableFutureCombined.get();
        log.info("executeSubtasks method test finished. Final output: {};", output.get("test_var"));
        return output;
    }

}
```

Логи по завершению выпонения выглядят следующим образом:

```
2020-08-26 20:07:43,537 INFO  e.i.e.t.ExecuteAllAsyncDemoAp [main] - executeSubtasks method test started
2020-08-26 20:07:44,644 INFO  e.i.kanclerrpa.taskexecution.task.Task4 [pool-1-thread-5] - Task 4 finished.
Provided input: null. It was sleeping 1000 millis
2020-08-26 20:07:45,634 INFO  e.i.kanclerrpa.taskexecution.task.Task3 [pool-1-thread-4] - Task 3 finished.
Provided input: null. It was sleeping 2000 millis
2020-08-26 20:07:46,638 INFO  e.i.kanclerrpa.taskexecution.task.Task2 [pool-1-thread-3] - Task 2 finished.
Provided input: null. It was sleeping 3000 millis
2020-08-26 20:07:47,627 INFO  e.i.kanclerrpa.taskexecution.task.Task1 [pool-1-thread-2] - Task 1 finished.
Provided input: null. It was sleeping 4000 millis
2020-08-26 20:07:47,636 INFO  e.i.e.t.ExecuteAllAsyncDemoAp [main] - executeSubtasks method test finished.
Final output: Output from Task4;
```

Можно определить **собственную стратегию слияния вывода** . Для этого необходимо использовать следующую сигнатуру метода:

```
CompletableFuture<TaskOutput> execute(TaskInput input, ExecutionService.MergeStrategy mergeStrategy, Class<? extends ApTask>... task)
```

где:

- **input** - объект входных данных, полученный от локального средства запуска в случае локального выполнения, или объект входных данных, предоставленный на сервере управления в случае выполнения на сервере.
- **mergeStrategy** - объект BiConsumer (обычно лямбда-выражение), в котором можно определить стратегию слияния.
- **classes** - классы задач, которые нужно выполнить, например **MyFirstTask.class, MySecondTask.class**.

Возвращает: объект **CompletableFuture** для которого вы можете вызвать метод **get(),** чтобы получить объект **TaskOutput**.

Рассмотрим пример использования, в котором мы определяем собственную стратегию слияния. В случае конфликта (выходная переменная с одинаковым именем) значения объединяются с помощью символа вертикальной черты ("|"):

ExecuteAllAsyncWithMergeStrategyDemoAp.java

```
package com.iba.kanclerrpa.taskexecution;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.service.ExecutionService;
import com.iba.kanclerrpa.taskexecution.task.Task1;
import com.iba.kanclerrpa.taskexecution.task.Task2;
import com.iba.kanclerrpa.taskexecution.task.Task3;
import com.iba.kanclerrpa.taskexecution.task.Task4;
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.CompletableFuture;

@Slf4j
@ApModuleEntry(name = "Execute All Async with merge strategy Demo")
public class ExecuteAllAsyncWithMergeStrategyDemoAp extends ApModule {

        public static void main(String[] args) {
                ApModuleRunner.localLaunch(ExecuteAllAsyncWithMergeStrategyDemoAp.class);
        }

        public TaskOutput run() throws Exception {
                log.info("executeAllAsync with merge strategy method test started");

                ExecutionService.MergeStrategy combineWithPipeMergeStrategy = (finalOutput,
finishedTaskVariables) -> {
                        for (String key : finishedTaskVariables.keySet()) {
                                if (finalOutput.containsKey(key)) {
                                        String currentFinalValue = finalOutput.get(key);
                                        currentFinalValue = currentFinalValue + "|" + finishedTaskVariables.get
(key);
                                        finalOutput.replace(key, currentFinalValue);
                                } else {
                                        finalOutput.put(key, finishedTaskVariables.get(key));
                                }
                        }
                };

                CompletableFuture<TaskOutput> completableFutureCombined = execute(getInput(),
combineWithPipeMergeStrategy, Task1.class, Task2.class, Task3.class, Task4.class);
                TaskOutput output = completableFutureCombined.get();
                log.info("executeAllAsync with merge strategy method test finished. Final output: {};", output.
get("test_var"));
                return output;
        }
}
```

Логи по завершению выпонения выглядят следующим образом:

```
2020-08-26 20:17:53,658 INFO  e.i.e.t.ExecuteAllAsyncWithMergeStrategyDemoAp [main] - executeSubtasks with
merge strategy method test started
2020-08-26 20:17:54,747 INFO  e.i.kanclerrpa.taskexecution.task.Task4 [pool-1-thread-5] - Task 4 finished.
Provided input: null. It was sleeping 1000 millis
2020-08-26 20:17:55,753 INFO  e.i.kanclerrpa.taskexecution.task.Task3 [pool-1-thread-4] - Task 3 finished.
Provided input: null. It was sleeping 2000 millis
2020-08-26 20:17:56,758 INFO  e.i.kanclerrpa.taskexecution.task.Task2 [pool-1-thread-3] - Task 2 finished.
Provided input: null. It was sleeping 3000 millis
2020-08-26 20:17:57,743 INFO  e.i.kanclerrpa.taskexecution.task.Task1 [pool-1-thread-2] - Task 1 finished.
Provided input: null. It was sleeping 4000 millis
2020-08-26 20:17:57,754 INFO  e.i.e.t.ExecuteAllAsyncWithMergeStrategyDemoAp [main] - executeSubtasks with
merge strategy method test finished. Final output: Output from Task1|Output from Task2|Output from Task3|Output
from Task4;
```

Использование входных и полученных данных

Что такое входные/полученные данные?

Входные и полученные данные необходимы для передачи данных в задачи процесса и получения результата. Зачастую они используются для передачи идентификаторов данных для обработки, для передачи данных между шагами (обычно данные хранятся в базе данных и только ключи передаются в следующие шаги), и для получения выходных данных (в качестве результата работы). Это могут быть бизнес-данные или информация об успешном запуске.

Структура входных/полученных данных

Вы можете сохранить любой сериализуемый объект в качестве входного или полученного объекта. Входные/полученные объекты хранят данные в переменных типа *HashMap<String, String>*. Ключ - это имя переменной, а значение - это данные, хранящиеся в json формате. При просмотре/сохранении во входящих/полученных данных, добавленный объект автоматически сериализуется/десериализуется в json строку.

Использование

Для загрузки данных для в качестве входных данных для процесса, перейдите в **Сведения об Автоматизированном Процессе** **Входные данные**. См. Входные данные.

После этого, для использования данных, загруженных через сервер управления или созданных в коде, вам необходимо объявить переменные с аннотацией *@Input* с соответствующим именем и типом переменной.

```
public class SampleTask extends ApTask {

    @Input(key = "newKey")
    private String myInputString;

        @Input
        private MyTransferClass myTransferObject;
...
```

В примере выше мы ожидаем две переменные в качестве входных данных:

- Переменная типа *String*. *@Input(key = "newKey")* обозначает что переменная была передана процессу с ключом *newKey*.
- Переменная типа *MyTransferClass* type была передана с ключом *myTransferObject* без дополнительных опций. Мы просто объявили переменную с соответствующим типом и именем и определили аннотацию *@Input*, затем мы можем использовать данные.

Для передачи данных для вывода используется аннотация *@Output* при объявлении переменной. Если вы хотите получить данные, внесите изменения и передайте их дальше, возможно использовать

обе аннотации одновременно.

```
public class SampleTask extends ApTask {

        @Output
        @Input
        private MyTransferClass myTransferObject;

    @Output
    private String myOutputString;
...
```

В данном примере мы получаем входной объект *myTransferObject* и отправляем его в качестве выходных данных. Мы также создаем дополнительную переменную *myOutputString,* которая также передается в следующий шаг.

Пример

В данном примере мы работаем с классом Book. Пример состоит из двух шагов. В первом шаге мы получаем данные и передаем их во второй шаг. Во втором шаге данные берутся из первого, производятся изменения и данные отправляются для вывода.

Класс Book - это объект, который сохраняется в хранилище данных и затем будет передаваться между шагами.

Book.java

```
import com.iba.kanclerrpa.persistence.annotation.Column;
import com.iba.kanclerrpa.persistence.annotation.Entity;
import com.iba.kanclerrpa.persistence.annotation.Id;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity(value = "io_example_books")
@NoArgsConstructor
@Data
public class Book {

    @Id
    @Column("index")
    private String index;

    @Column("name")
    private String name;

    @Column("author")
    private String author;

    @Column("count")
    private String count;

    public Book(String index, String name, String author) {
        this.index = index;
        this.name = name;
        this.author = author;
    }

    public Book(String index, String name, String author, String count) {
        this.index = index;
        this.name = name;
        this.author = author;
        this.count = count;
    }
}
```

Передача объекта целиком может потребовать много ресурсов. Поэтому, правильный подход - это сохранить объекты в хранилище данных и передавать их идентификаторы между шагами. Для передачи информации, был создан специальный объект DTO, который содержит идентификатор процесса и передаваемых данных.

BooksResultDto.java

```
import lombok.Getter;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;

@ToString
public class BooksResultDto {

    @Getter
    private final String uuid;

    @Getter
    private final List<String> bookIds;

    public BooksResultDto(String uuid) {
        this.bookIds = new ArrayList<>();
        this.uuid = uuid;
    }

    public void add(String id) {
        bookIds.add(id);
    }
}
```

Ниже мы добавим методы для работы с хранилищем данных в интерфейс BookRepository.

BookRepository.java

```
import com.iba.kanclerrpa.persistence.CrudRepository;
import org.example.domain.Book;

import java.util.ArrayList;
import java.util.List;

public interface BookRepository extends CrudRepository<Book, String> {
    default List<Book> getBookByName(String name) {

        List<Book> result = new ArrayList<>();
        for (Book book : findAll()) {
            if (book.getName().equals(name)) {
                result.add(book);
            }
        }
        return result;
    }

    default void saveBooks(List<Book> books) {

        for (Book book : books) {
            save(book);
        }
    }
}
```

SampleTaskOne получает books, сохраняет их в хранилище данных и передает DTO объект с идентификаторами в следующий шаг.

SampleTaskOne.java

```
import java.util.Arrays;
import java.util.List;
import java.util.UUID;

import javax.inject.Inject;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;

import org.example.domain.Book;
import org.example.domain.BooksResultDto;
import org.example.repository.BookRepository;

@ApTaskEntry(name = "Sample Task One")
public class SampleTaskOne extends ApTask {

    // key to output identification
    public static final String BOOK_KEY = "Book";

    @Output(BOOK_KEY)
    private BooksResultDto booksResult;

    @Inject
    private BookRepository bookRepository;

    @Override
    public void execute() {

        booksResult = new BooksResultDto(getUuid());

        for (Book book : getBooks()) {
            // save to DataStore
            bookRepository.save(book);

            // add ID to output
            booksResult.add(book.getIndex());
        }
    }

    // Getting books. Data can be obtained from the client program, database,
    // scraped from the website, etc.
    private List<Book> getBooks() {
        Book thinkingInJava = new Book(UUID.randomUUID().toString(), "Thinking in Java", "Bruce Eckel");
        Book cipollino = new Book(UUID.randomUUID().toString(), "Le avventure di Cipollino", "Giovanni
Francesco Rodari");
        Book wadAndPeace = new Book(UUID.randomUUID().toString(), "War and Peace", "Lev Tolstoy");

        return Arrays.asList(thinkingInJava, cipollino, wadAndPeace);
    }
```

SampleTaskTwo получает входные данные из SampleTaskOne используя @Input аннотацию.
Затем, используя идентификаторы из входного DTO объекта, данные извлекаются из хранилища данных. После выполнения манипуляций с полученными данными, они обновляются в хранилище данных, идентификаторы новых объектов добавляются к выходному объекту.

Аннотация @Output определяет, что после обработки данные будут отправлены в следующий шаг.

SampleTaskTwo.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;
import org.example.domain.Book;
import org.example.domain.BooksResultDto;
import org.example.repository.BookRepository;

import javax.inject.Inject;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.UUID;

import static org.example.tasks.SampleTaskOne.BOOK_KEY;

@ApTaskEntry(name = "Sample Task Two")
@Slf4j
public class SampleTaskTwo extends ApTask {

    @Input(BOOK_KEY)
    @Output
    private BooksResultDto booksResult;

    @Inject
    private BookRepository bookRepository;

    @Override
    public void execute() {

        //get entities
        List<Book> books = new ArrayList<>();
        for (String id : booksResult.getBookIds()) {
            books.add(bookRepository.findById(id));
        }

        //update entities
        for (Book book : books) {
            book.setCount(String.valueOf(getBooksCount(book.getName())));
            bookRepository.save(book);
        }

        //add new entities to repository
        Book newBook = getNewBook();
        bookRepository.save(newBook);

        //add new entities to result
        booksResult.add(newBook.getIndex());
    }

    private Book getNewBook() {
        return new Book(UUID.randomUUID().toString(), "The Godfather", "Mario Puzo", "50");
    }

    //Getting additional information. Data can be obtained from the client program, database, scraped from the
website, etc.
    private int getBooksCount(String name) {
        return new Random().nextInt(100);
    }
}
```

Автоматизированный процесс для запуска задач:

Module.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import org.example.tasks.SampleTaskOne;
import org.example.tasks.SampleTaskTwo;


@ApModuleEntry(name = "InputOutputDTO", description = "This process automates a lot of work...")
public class Module extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), SampleTaskOne.class).thenCompose(execute(SampleTaskTwo.class)).get();
    }
}
```

Локальное средство запуска:

ModuleLokalRun.java

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(Module.class, new DevelopmentConfigurationModule(args));
        //ApModuleRunner.localLaunch(Module.class);
    }
}
```

Аннотации

Компонент автоматизированных процессов

Следующие аннотации обязательны для успешного запуска автоматизированного процесса. В противном приложение может не запустится.

@ApModuleEntry

Аннотация **@ApModuleEntry** означает, что целевой класс представляет автоматизированный процесс.

Когда узел выполняет автоматизированный процесс из jar пакета, класс с аннотацией ApModuleEntry будет запущен.

В случае, когда в классе более одной аннотации, класс с `defaultRunModule=true` будет запущен.

Класс с аннотацией обычно расширяется от `ApModule` класса, поэтому его выполнение начинается с метода `run()`.

```
@ApModuleEntry(defaultRunModule=true)
public class RdpAp extends ApModule {
    @Override
    public TaskOutput run() {
        // workflow resulting in TaskOutput
    }
}
```

@ApTaskEntry

ApTaskEntry можно рассматривать как один шаг внутри автоматизированного процесса

Класс с аннотацией обычно расширяется от **ApTask** с методом `execute()` как точка входа, который инкапсулирует бизнес-логику этого шага:

```
@ApTaskEntry(name = "Calculator task", description = "Demonstrates image-based automation")
public class PerformCalcMultiUser extends ApTask {
    @Override
    public void execute() {
                // sequence of actions
    }
}
```

Аннотации уровня задачи

Аннотации, представленные ниже, могут быть применены для полей и методов внутри записи задачи автоматизированного процесса.

@Output

Поле с аннотацией **@Output** будет экспортировано как часть выходных данных задачи после завершения ее выполнения.

Позже ее можно импортировать в следующую задачу используя аннотацию **@Input**:

```
@ApTaskEntry(name = "Extract Debtors")
public class ExtractDebtors extends ApTask {
    @Output(key = "Debtor")
    private DebtorsResultTo debtorsResult;

    @Override
    public void execute() {
        debtorsResult = new DebtorsResultTo("abc");
    }
}
```

@Input

@Input аннотация позволяет импортировать значение, которое было экспортировано в предыдущей задаче.

```
@ApTaskEntry(name = "Check Bank Guarantees")
public class CheckBankGuarantees extends ApTask {

    @Input(key = "Debtor")
    private DebtorsResultTo debtorsResult;

    @Override
    public void execute() {
                // actions performed on debtorsResult
    }
}
```

`@Input` и `@Output` имеют свойства `key` и `required`:

`key` - ключ, который ассоциируется с сохраненным и извлеченным значением.

`required` - флаг логического значения, который принудительно проверяет связанное значение на null.

@Configuration

@Configuration извлекает значение по имени из сервиса конфигурации или хранилища секретов.

```
@Configuration(value = "invoiceplane.client.url", defaultValue = "http://10.224.0.35:8085/index.php/sessions
/login")
protected String invoicePlaneUrl;

@Configuration(value = "invoiceplane.secrets", defaultValue = "{\"user\": \"admin@ibagroup.eu\",\"password\": \"
o66Lc1Jn6Z\"}")
protected SecretCredentials invoicePlaneCredentials;

@Configuration(value = "products_count", defaultValue = "20")
private int maxProductsCountToRead;
```

@AfterInit

@AfterInit будет выбывать метод после инициализации базовой задачи и после вызова метода `execute()`.

Хорошая практика инициализировать настройки конфигурации и драйвера следующим способом:

```
@ApTaskEntry(name = "Extract Debtors", description = "Extract debtors from 1C Accounting")
public class ExtractDebtors extends ApTask {
    private BrowserDriver driver;
    private String url;

    @AfterInit
    public void init() {
        url = getConfigurationService().get("acc.client.url");
        driver = getDriver(BrowserDriver.class, new HashMap<Enum, Object>() {{
            put(DriverParams.SELENIUM_NODE_CAPABILITIES, new ChromeOptions());
        }});
    }

    @Override
    public void execute() {
        OnecApplication application = new OnecApplication(driver);
        LoginPage loginPage = application.open(url);
                // ...
    }
}
```

@OnError

Метод **execute()** может распространять исключения без входа в задачу

Для захвата таких исключений и создания логики для обработки исключений, рекомендуется создать специальный метод с аннотацией **@OnError:**

```
@ApTaskEntry(name = "Arithmetic Calculation")
public class ArithmeticCalculation extends ApTask {
    @Override
    public void execute() throws Throwable {
        int a = 1/0;
    }

    @OnError
    public void onError(Throwable throwable) {
        System.err.println("Arithmetic error occured!");
    }
}
```

Элементы страницы

Эта группа аннотирует элементы объекта страницы, такие как: поля ввода, кнопки, панели и пр.

@FIndBy

Указывает механизм поиска элемента страницы. Можно получить id поля, имя, xpath и другие параметры в зависимости от драйвера.

```
@FindBy(name = "One")
private RemoteWebElement one;

@FindBy(xpath = "//*[@id='userName']")
private RemoteWebElement username;
```

@Wait

Указывает время ожидания в секундах и функцию ожидания как Selenium FluentWait условие. Может быть использована вместе с аннотацией **@FindBy.**

```
@FindBy(xpath = "//input[@id=\"Name\"]")
@Wait(waitFunc = Wait.WaitFunc.VISIBLE)
private WebElement nameField;

@FindBy(xpath = "//a[@id=\"searchButton\"]")
@Wait(value = 30, waitFunc = Wait.WaitFunc.CLICKABLE)
private WebElement searchBtn;
```

@Image

Отмечает элемент, который должен быть обнаружен драйвером экрана, и получает URL-адрес изображения относительно каталога ресурсов/изображений.

```
@Image(url = "navigation-btn-1.png")
private IMElement navigation;
```

@WithTimeout

Указывает время ожидания для элементов драйвера экрана, получает время в секундах в качестве параметра. Может быть использована вместе с аннотацией **@Image**.

```
@Image(url = "windows-start-1.png")
@WithTimeout(time = 3)
private IMElement start;
```

Аннотации Lombok

Этот набор аннотаций предоставляется Lombok, библиотекой, которая генерирует код. Это помогает избежать повторения написания кода и, следовательно, уменьшает типичные ошибки в задачах. Использование Lombok описано в следующей статье: Lombok.

Обработка исключений

Механизм обработки исключений Канцлер RPA

У разработчика есть три варианта работы с исключением внутри автоматизировнного процесса:

- перехватить его внутри автоматизированного процесса и, следовательно, предотвратить завершение автоматизированного процесса со статусом **Не выполнен**.
- перевыбросить иключение или оставить его как есть для того, чтобы оно прошло по стеку вызовов и было перехвачено в последствии.
- декларировать `@OnError` обработчик для передачи способа работы с исключениями, не обработанными разработчиком.

Лучший способ перехвата - определить особые исключения для автоматизированного процесса.

Определение исключений для автоматизированного процесса

Разработчик может определить ошибки, относящиеся к автоматизированному процессу путем реализации `com.iba.kanclerrpa.engine.exception.ErrorDetails` интерфейса в классе Enam.

Рекомендуется хранить сообщения об ошибках в recourse bundle.

InvoicePlaneError.java

```
public enum InvoicePlaneError implements ErrorDetails {

    LOGIN_FAILED,
    ADD_PRODUCT_FAILED;

    private static ResourceBundle rb = ResourceBundle.getBundle("invoiceplane_error");

    public String getMessageTemplate() {
        return rb.getString(this.name());
    }
}
```

Сообщения об ошибках определены в соответствующем recourse bundle, где каждый ключ соответствует имени из enum.

Сообщение может содержать произвольное количество параметров, заключенных в `{}`.

invoiceplane_error.properties

```
LOGIN_FAILED=Login failed with user {0}
ADD_PRODUCT_FAILED=Add failed for product {0}
```

Как использовать вновь созданное исключение, просто выбросьте `com.iba.kanclerrpa.engine.exception.RpaException` и передайте аргументы, если они требуются сообщению.

```
if (loginFailed) {
        throw new RpaException(InvoicePlaneError.LOGIN_FAILED, credentials.getUser());
}
```

Встроенные исключения

Ядро Канцлер RPA поставляется с рядом предопределенных исключений. Их можно найти в трех классах: **CoreError**, **CsError**, **NodeError**. Все эти классы являются перечислениями, реализуемыми в интерфейсе com.iba.kanclerrpa.engine.exception.ErrorDetails.

CoreError

Данные исключения относятся ко внутренней работе ядра и таким проблемам как: драйвера, данные, запуск автоматизированного процесса и т.д.

Код исключения	Описание
E1000	Unexpected fatal error
E1001	Unexpected error occurs
E1002	Booting error occurs
E1003	Booting error: module instantiation error
E1004	Unable to load a configuration file or no configuration file provided
E2000	Configuration key is not defined
E2001	Wrong configuration value
E2002	The specified configuration has wrong structure
E2003	Cannot find data
E2004	S3 is not configured
E2005	General configuration error
E3000	Execution error occurs in module
E3001	Execution error occurs in task
E3002	Instance creation error
E3003	Can not inject fields into instance
E3004	Can not execute annotated method for instance
E3005	Execution stopped at task
E5000	REST service call failed
E5001	Network timeout
E5002	REST call failed: standalone mode
E5003	An error occurs during uploading file to S3
E5004	An error occurs during generation of S3 url
E5005	An error occurs during downloading file from S3
E6000	Database consistency error
E6001	Database foreign constraint error
E6002	Database unique constraint error
E6003	Database duplicate record error
E6004	Datasource persistence error
E6005	Datasource SQL error
E6006	Datasource error: cannot determine table
E7000	Secret vault error

E7001	No trust store password found
E7002	No MongoDb password found
E7003	General security exception
E7004	Secret vault unsealing error
E7005	An error occurs during processing of file with secret configuration
E7006	Secret alias contains illegal characters
E8000	An error occurs during closing the driver
E8001	An error occurs during page creation
E8002	An error occurs during driver initialization
E8003	There is not current window for driver
E8004	Can't switch to window. Window with the following selector not found
E8005	An error occurs during opening application
E8006	An error occurs during taking screenshot
E8007	Timeout during waiting for UI element
E8008	Text validation error for UI element
E8009	List of opened windows is empty
E8010	Cannot find UI element on the screen
E8011	Can't launch application*/
E8012	Can't get windows
E8013	Unexpected driver error

CsError

Этот класс включает исключения связанные с HTTP ошибками и ошибками, которые связаны с проблемами на сервере, такие как: контроль доступа, целостность базы данных и другие.

Код исключения	HTTP код	Описание
S1000	500	Unexpected error occurs
S4000	401	User is disabled
S4001	401	Invalid credentials for user
S4002	403	Access denied
S4003	404	Entity not found
S4004	400	Wrong input format
S4005	400	CSV error
S4006	400	Wrong input
S4007	400	Method arguments are not valid
S4008	400	Invalid cron expression
S4009	400	Wrong input format: type mismatch
S4010	400	Wrong input format: missing request parameter
S4011	401	User does not exist

S4012	401	Wrong token
S4013	400	Wrong input format: DataStore record
S4014	400	Wrong input: missing DataStore record in request
S4015	400	Wrong input: either schema or csv file is accepted but not both
S4016	400	Please provide valid JSON
S4017	400	Cannot extract zip package
S4018	400	Wrong structure of ap package
S4019	400	CSV header error
S3000	500	Asynchronous job exception
S6000	500	Database consistency error
S6001	500	Database foreign constraint error
S6002	500	Database unique constraint error
S6003	500	Database duplicate record error
S6004	500	Datasource persistence error
S6005	500	Datasource error
S6007	500	Automation process with the same name already exists
S6008	500	Data Store with the same name already exists
S6009	500	Schedule with the same name already exists
S6010	500	Node with the same name already exists
S6011	500	Role with the same name already exists
S6012	500	NodeConfiguration parameter with the same key already exists
S6013	500	Secret entry with the same alias already exists
S6014	500	User with the same username already exists
S6015	500	Capability with the same name already exists
S7000	500	Secret vault error

NodeError

Эти исключения возникают только для агента узла.

Exception Code	Description
N1000	Unexpected error
N2000	No configuration found
N3000	Multiply nodes
N4000	Communicate with another JVM failed

Lombok

Введение

Канцлер RPA поставляется со встроенной библиотекой Lombok. Это генератор кода времени компиляции, который помогает избавиться от повторяющихся структур кода, а также унифицировать практику разработки в команде.

Любая популярная сегодня интегрированная среда разработки поддерживает Lombok через плагины, дополнительные сведения см. в Установка плагина Lombok.

Lombok API включает в себя стабильные и экспериментальные функции. Рекомендуется придерживаться стабильной версии, поскольку она хорошо протестирована, давно поддерживается и с меньшей вероятностью будет изменена.

Мы используем только самые популярные приложения Lombok. Больше можно найти в официальной документации.

Стабильные функции

@Slf4j

Создаёт поле регистратора slf4j в целевом классе:

```
@Slf4j
public class CalculatorMainPage extends ScreenPage {

    public void displayText() {
        log.info("Displaying text from calculator page");
    }
}
```

@Getter и @Setter

Генерирует методы getter и setter для целевого поля:

```
class Person {
    @Getter
        @Setter
    private String name;

        Person (String name) {
                this.name = name;
        }
}

class AppMain {
        public static void main(String[] args) {
                Person person = new Person("Ellie");
                System.out.println(person.getName());

                person.setName("Abby");
                System.out.println(person.getName());
        }
}
```

Аннотации также могут иметь модификации. Одной из них является `@Getter(lazy=true)`, которая разрешает отложенную загрузку.

Это означает, что базовое поле будет инициализировано только при первом вызове getter, а результат будет кэширован для последующего повторного использования.

@AllArgsConstructor

Генерирует конструктор, который получает все поля класса в качестве аргументов.

```
@AllArgsConstructor
class Cat {
    private String name;
    private int age;
}

public static void main(String[] args) {
    Cat cat = new Cat("Felix", 8);
}
```

Lombok предоставляет другие аннотации для создания конструкторов:

`@NoArgsConstructor` создает конструктор по умолчанию, т.е. без параметров.

`@RequiredArgsConstructor` создает конструктор только с полями final и @NonNull.

@ToString

Как следует из названия, @ToString генерирует одноименный метод.

```
class LocalRunner {
    @ToString
    static class Product {
        String name = "Butter";
        int price = 2;
    }

    public static void main(String[] args) {
        Product product = new Product();
        System.out.println(product.toString());
    }
}
```

Если мы запустим приведенный выше код, вывод будет выглядеть следующим образом: LocalRunner.Product(name=Butter, price=2) вместо com.iba.kanclerrpa.lombok.LocalRunner$Product@121714c.

@Builder

Шаблон builder упрощает создание класса с большим количеством параметров, особенно когда большинство из них являются необязательными.

```
@Builder
public class NutritionFacts {
    private int servingSize = -1;
    private int servings = -1;
    private int calories = 0;
    private int fat = 0;
    private int sodium = 0;
    private int carbohydrate = 0;

    public static void main(String[] args) {
        NutritionFacts facts = NutritionFacts.builder()
                .calories(200)
                .fat(40)
                .carbohydrate(120)
                .build();
    }
}
```

Как вы видите , при применении `@Builder` не нужно объявлять все возможные конструкторы `NutritionFacts`, охватывающие различные наборы параметров. Вместо этого мы создаем экземпляр класса builder и передаем только необходимые параметры.

Более подробно о функциях Lombok вы можете прочитать в официальной документации.

Экспериментальные функции

@UtilityClass

Утилитарный класс, к которому нельзя создать экземпляры, и который содержит только статические методы.

```
@UtilityClass
public class MathUtils {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Данный класс после декомпиляции выглядит следующим образом. Добавлен приватный конструктор, метод multiply стал статическим, а сам класс - final.

```
public final class MathUtils {
    private MathUtils() {
        throw new UnsupportedOperationException("This is a utility class and cannot be instantiated");
    }

    public static int multiply(int a, int b) {
        return a * b;
    }
}
```

Мы не будем рассматривать более подробно экспериментальные функции, так как они могут изменяться. Информацию о них вы можете получить в официальной документации.

Delombok

Аннотации Lombok можно delombok, т. е. развернуть в фактический код.

Это можно сделать применив команду плагина IDE или выполнив командную строку.

Чтобы применить delombok в Intellij IDEA, вызовите аннотации Refactor/Delombok/All lombok для целевого класса. Имейте в виду, что должен быть установлен плагин Lombok .

Шаблон проектирования Page Object

Page Object — это шаблон проектирования, который стал популярным в RPA для улучшения поддержки скриптов автоматизации и уменьшения дублирования кода. Page Object — это объектно-ориентированный класс, который служит интерфейсом к странице вашего приложения. Сценарии автоматизации используют методы класса Page Object всякий раз, когда необходимо взаимодействовие с пользовательским интерфейсом страницы. Преимущество заключается в том, что при изменении пользовательского интерфейса страницы отсутствует необходимость изменять сами сценарии автоматизации, необходимо изменить только код внутри Page Object. Все изменения для поддержки нового пользовательского интерфейса находятся в одном месте.

Шаблон проектирования Page Object предоставляет следующие преимущества:

- Существует четкое разделение между кодом сценариев автоматизации и кодом, привязанным к странице, таким как локаторы и макет.
- Существует единый репозиторий для сервисов или операций, предлагаемых страницей, они не разбросаны по сценариям автоматизации.

Реализация

Подход Page Object может быть реализован в 3 этапа:

- Расширение класса Application
- Возврат Page object из метода open
- Расширение соответствующего класса Page

Расширение класса Application

При проектировании Page Object Канцлер RPA мы создаем класс в качестве точки входа для любого приложения. Все классы должны расширять класс **Application**.

В классе **Application** есть конструктор, который принимает объект **Driver** , который отвечает за инициализацию страниц через PageFactory (используя метод **createPage**). Также в качестве точки входа **application** он обладает методом **open**, который принимает любые аргументы приложения и отвечает за открытие приложения и возврат стартовой страницы (обычно это страница входа).

Например, класс автоматизации веб-приложения Invoice Plane:

InvoicePlaneApplication.java

```
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.element.UiElement;

public class InvoicePlaneApplication extends Application<BrowserDriver, BrowserElement> {

    public InvoicePlaneApplication(BrowserDriver driver) {
        super(driver);
    }

    @Override
    public LoginPage open(String... args) {
                ...
    }
}
```

Возврат Page object из метода open

Метод **open** должен возвращать page object, поэтому весь класс **InvoicePlaneApplication.java** будет выглядеть следующим образом:

InvoicePlaneApplication.java

```
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.element.UiElement;
import com.iba.kanclerrpa.system.invoiceplane.page.LoginPage;

public class InvoicePlaneApplication extends Application<BrowserDriver, BrowserElement> {

    public InvoicePlaneApplication(BrowserDriver driver) {
        super(driver);
    }

    @Override
    public LoginPage open(String... args) {
        String invoicePlaneUrl = args[0];
        getDriver().get(invoicePlaneUrl);
        getDriver().manage().window().maximize();
        LoginPage loginPage = createPage(LoginPage.class);
        return loginPage;
    }
}
```

Метод **createPage** отвечает за вызов соответствующей PageFactory для создания страницы, необходимой для инициализации селекторов-аннотаций (таких как @FindBy, @Wait). Также он содержит ссылку на объект Application, который содкржит ссылку на объект Driver.

Расширение соответствующего класса Page

Любой класс page obhect должен расширять соответствующую базовую страницу:

- **WebPage** - если страница работает с **BrowserDriver**.
- **DesktopPage** - если страница работает с **DesktopDriver**.
- **ScreenPage** - если страница работает с **ScreenDriver**.
- **SapPage** -если страница работает с **SapDriver**.

В нашем примере **LoginPage** должен расширять **WebPage,** так как он описывает страницу веб-приложения Invoice Plane:

LoginPage

```
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import com.iba.kanclerrpa.engine.rpa.page.WebPage;

public class LoginPage extends WebPage {

    public Dashboard login(SecretCredentials invoicePlaneCredentials) {
        ...
        return createPage(Dashboard.class);
    }

}
```

Обратите внимание, что все новые страницы, создаваемые внутри других страниц, должны быть инициализированы с помощью метода **createPage** , который необходим для инициализации селекторных аннотаций и содержит ссылку на объект Application, который содержит ссылку на объект Driver.

Селекторные аннотации

Аннотации FindBy и Image

Все драйверы имеют разные ограничения на использование атрибутов селекторных аннотаций. Используйте приведенную ниже матрицу совместимости, чтобы проверить, какие атрибуты можно использовать для следующих аннотаций:

- `org.openqa.selenium.support.FindBy`
- `com.iba.kanclerrpa.engine.rpa.po.annotation.Image`

	`@FindBy (id = "any")`	`@FindBy (name = "any")`	`@FindBy (className = "any")`	`@FindBy (css= "any")`	`@FindBy (tagName= "any")`	`@FindBy (linkText= "any")`	`@FindBy (partialLinkText = "any")`	`@FindBy (xpath = "any")`	`@Image (url= "any")`
BrowserDriver									
DesktopDriver									
JavaDriver									
SapDriver									
ScreenDriver									

Аннотации Wait

Обычно мы используем селекторные аннотации (**FindBy** и **Image**) вместе с аннотациями **Wait**. Существуют 2 аннотации для сообщения драйверу, как мы должны ожидать элемент:

- `com.iba.kanclerrpa.engine.rpa.po.annotation.Wait`
- `com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout`

	`@Wait`	`@WithTimeout`
BrowserDriver		
DesktopDriver		
JavaDriver		

SapDriver		
ScreenDriver		

@Wait аннотация

Аннотация **@Wait** содержит два атрибута: **wait timeout** (*по умолчанию 20 секунд*) и **wait function.**

Таким образом, примеры использования могут выглядеть следующим образом:

LoginPage

```
import com.iba.kanclerrpa.engine.rpa.page.WebPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Wait;
import com.iba.kanclerrpa.engine.rpa.element.BrowserElement;
import org.openqa.selenium.support.FindBy;

public class MyPage extends WebPage {

    @FindBy(id = "email")
    @Wait(30)
    private BrowserElement emailField;

        ...
}
```

ИЛИ с функцией Wait:

LoginPage

```
import com.iba.kanclerrpa.engine.rpa.page.WebPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Wait;
import com.iba.kanclerrpa.engine.rpa.element.Browserlement;
import org.openqa.selenium.support.FindBy;

public class MyPage extends WebPage {

    @FindBy(id = "email")
    @Wait(value = 30, waitFunc = Wait.WaitFunc.CLICKABLE)
    private BrowserElement emailField;

        ...
}
```

@WithTimeout аннотация

Аннотация **@WithTimeout** содержит 2 атрибута: **time** (*обязателен*) и **poll** (*по умолчанию 1 сек*).

Ниже приведен пример использования:

LoginPage

```
import com.iba.kanclerrpa.engine.rpa.page.DesktopPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import org.openqa.selenium.support.FindBy;

public class MainPage extends DesktopPage {

    @FindBy(name = "password")
    @WithTimeout(time = 20, poll = 2)
    private DesktopElement passwordField;

        ...
}
```

Полный пример Page object

Используя правильные структуры классов и селекторных аннотаций, полный класс **LoginPage.java** для веб-приложения может выглядеть следующим образом:

LoginPage

```
package com.iba.kanclerrpa.system.invoiceplane.page;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.exception.RpaException;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import com.iba.kanclerrpa.engine.rpa.element.BrowserElement;
import com.iba.kanclerrpa.engine.rpa.locator.BrowserSearch;
import com.iba.kanclerrpa.engine.rpa.page.WebPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Wait;
import com.iba.kanclerrpa.system.invoiceplane.exception.InvoicePlaneError;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.TimeoutException;
import org.openqa.selenium.support.FindBy;

import static com.iba.kanclerrpa.engine.rpa.locator.BrowserSearch.ExpectedConditions.or;
import static com.iba.kanclerrpa.engine.rpa.locator.BrowserSearch.ExpectedConditions.presenceOfElementLocated;
import static com.iba.kanclerrpa.engine.rpa.locator.BrowserSearch.ExpectedConditions.visibilityOfElementLocated;

@Slf4j
public class LoginPage extends WebPage {

    private static final String PAGE_CSS_SELECTOR = ".container #login";

    private static final int PAGE_LOAD_TIMEOUT = 20;

    private static final int LOGIN_WAIT_TIMEOUT = 20;

    private static final String successLoginCssSelector = "div#main-area";

    private static final String failedLoginCssSelector = "#login .alert";

    @FindBy(id = "email")
    @Wait(waitFunc = Wait.WaitFunc.CLICKABLE)
    private BrowserElement email;

    @FindBy(id = "password")
    @Wait(waitFunc = Wait.WaitFunc.CLICKABLE)
    private BrowserElement password;

    @FindBy(xpath = "//button[@type='submit']")
```

```
    @Wait(waitFunc = Wait.WaitFunc.CLICKABLE)
    private BrowserElement submit;

    @AfterInit
    public void init() {
        getDriver().waitForElement(visibilityOfElementLocated(BrowserSearch.cssSelector(PAGE_CSS_SELECTOR)),
PAGE_LOAD_TIMEOUT);
    }

    public Dashboard login(SecretCredentials invoicePlaneCredentials) {
        email.click();
        email.clear();
        email.sendKeys(invoicePlaneCredentials.getUser());

        password.click();
        password.clear();
        password.sendKeys(invoicePlaneCredentials.getPassword());

        submit.click();

        try {
            getDriver().waitFor(or(presenceOfElementLocated(BrowserSearch.cssSelector(successLoginCssSelector)),
                    presenceOfElementLocated(BrowserSearch.cssSelector(failedLoginCssSelector))),
LOGIN_WAIT_TIMEOUT);

            boolean loginFailedAppeared = getDriver().findElements(BrowserSearch.cssSelector
(failedLoginCssSelector)).size() > 0;
            if (loginFailedAppeared) {
                throw new RpaException(InvoicePlaneError.LOGIN_FAILED, invoicePlaneCredentials.getUser());
            }
        } catch (TimeoutException e) {
            log.debug("Unknown error during InvoiceTO Place authorisation process.");
            throw new RuntimeException();
        }

        return createPage(Dashboard.class);
    }

}
```

Таким образом, **LoginPage** имеет только метод **login**, который принимает секретные учетные данные. После нажатия кнопки могут появиться 2 элемента: для успешного входа в систему и для неудачного входа. Таким образом, в приведенном выше примере мы создаем условие wait как комбинацию двух возможных условий для разных селекторов. Если вход выполнен успешно, **LoginPage** возвращает **DashboardPage.**

Использование page-objects внутри задач (на уровне бизнес-логики)

После того как структура page-objects создана, можно использовать ее в задаче следующим образом:

```
InvoicePlaneApplication invoicePlaneApplication = new InvoicePlaneApplication(browserDriver);
LoginPage loginPage = invoicePlaneApplication.open(invoicePlaneUrl);
Dashboard dashboard = loginPage.login(invoicePlaneCredentials);
test(dashboard);
invoicePlaneApplication.close();
```

Дополнительные правила

Проектирование Page Object является гибким в применении, но нужно придерживаться нескольких основных правил, чтобы поддержка кода автоматизации была удобной:

- Сами Page objects никогда не должны содержать какой-либо бизнес-логики. Это часть ваших скриптов автоматизации, и они всегда должны находиться в коде скриптов, а не в Page object. Page object содержит представление страницы и

сервисы, предоставляемые страницей с помощью методов, но код, связанный с тем, что автоматизируется, не должен находиться в Page object.

- Page object не обязательно должен представлять все части самой страницы. Те же принципы, которые применяются для Page objects, можно использовать для создания Page *Component* Objects, которые представляют отдельные фрагменты страницы и могут быть включены в Page objects. Эти объекты компонентов могут предоставлять ссылки на элементы внутри отдельных фрагментов и методы для использования предоставляемой ими функциональности. Вы даже можете вкладывать объекты компонентов внутрь других объектов компонентов для более сложных страниц. Если страница в приложении, которое вы автоматизируете, имеет несколько компонентов или общие компоненты, используемые на всем сайте (например, панель навигации), это может повысить удобство поддержки и уменьшить дублирование кода.

Драйвера

Драйвер интерфейса

Драйвер интерфейса предоставляет методы API для взаимодействия с драйверами таким же образом. Этот интерфейс реализован следующими классами:

- BrowserDriver - для работы с веб-приложениями.
- DesktopDriver - для работы с дектопными приложениями (кроме приложений на Java).
- JavaDriver - для работы в десктопными приложениями на Java.
- SapDriver - для работы с Sap приложениями.
- ScreenDriver - для работы с любыми приложениями на основе распознавания областей экрана по шаблонам.

Некоторые методы из драйвер интерфейса поддерживаются во всех вышеперечисленых драйверах, но часть из них не поддерживаться в некоторых из них.

В дополнение к методам поиска элементов этот драйвер содержит следующие полезные методы:

- initPage
- sleep
- getScreenshot
- getScreenshotAsBytes
- getScreenshotAsFile
- getScreenshotAsBase64
- clipboard
- waitForElement
- waitForElementNot
- waitFor
- getInputDevices

Открытие новых приложений

get(String)

⚠ **BrowserDriver** - нативная поддержка Selenium

DesktopDriver - поддержка (пожалуйста, обратитесь странице драйвера чтобы увидеть дополнительную информацию)

JavaDriver - supports поддержка (пожалуйста, обратитесь к странице драйвера чтобы увидеть дополнительную информацию)

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

Первое, что вы захотите сделать после запуска бота, это открыть приложение. Этот метод используется для открытия приложения\ве-сайта и принимает путь к исполняемому файлу\ссылке на веб-сайт (зависит от типа драйвера).

```
public void get(String url)
```

Навигация браузера

getTitle

⚠ **BrowserDriver** - нативная поддержка Selenium

DesktopDriver - поддержка

JavaDriver - поддержка

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

Вы можете прочитать текущий заголовок приложения (или заголовок браузера)

```
public String getTitle()
```

getCurrentUrl

⚠ **BrowserDriver** - нативная поддержка Selenium

DesktopDriver - метод не поддерживается

JavaDriver - перейдите на страницу драйвера, чтобы увидеть дополнительную информацию

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

С помощью этого метода вы можете прочитать текущий URL-адрес из адресной строки браузера.

```
public String getCurrentUrl()
```

navigate

⚠ **BrowserDriver** - нативная поддержка Selenium

DesktopDriver - метод не поддерживается

JavaDriver - метод не поддерживается

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

Этот метод возвращает объект навигации, которая содержит полезные методы для выполнения навигации в браузере.

```
public Navigation navigate()
```

```
driver.navigate().to("https://selenium.dev"); //longer way of driver.get("https://selenium.dev")
driver.navigate().back();
driver.navigate().forward();
driver.navigate().refresh();
```

Работа с окнами

switchToWindow

⚠ Из-за характера драйверов у каждого драйвера есть свой способ переключения между окнами:

BrowserDriver - перейдите на страницу драйвера, чтобы увидеть дополнительную информацию

DesktopDriver - перейдите на страницу драйвера, чтобы увидеть дополнительную информацию

JavaDriver - перейдите на страницу драйвера, чтобы увидеть дополнительную информацию

SapDriver - перейдите на страницу драйвера, чтобы увидеть дополнительную информацию

ScreenDriver - метод не поддерживается

getWindowHandle

⚠ **BrowserDriver** - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

Каждое окно имеет уникальный дескриптор, который сохраняется в течение одной сессии. С помощью этого метода вы можете получить дескриптор текущего окна.

```
public String getWindowHandle()
```

getWindowHandles

⚠ **BrowserDriver** - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - метод не поддерживается

ScreenDriver - метод не поддерживается

Возвращает набор дескрипторов окон, которые можно использовать для перебора всех открытых окон, доступных для драйвера, чтобы вы могли использовать дескриптор для переключения на окно, передав их методу switchToWindow.

```
public Set<String> getWindowHandles()
```

close

⚠ **BrowserDriver** - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - поддерживает (пожалуйста, перейдите на страницу драйвера чтобы увидеть дополнительню информацию)

ScreenDriver - метод не поддерживается

Закройте текущее окно (закройте браузер, если это последнее окно, открытое в данный момент для BrowserDriver).

```
public void close()
```

window management

⚠ **BrowserDriver** - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - поддерживает

ScreenDriver - метод не поддерживается

Вы можете управлять объектом Window(изменить и получить размер, изменить положение). Возвращает интерфейс для управления текущим окном.

```
public Window window()
```

Каждый драйвер имеет собственную реализацию оконного интерфейса. Пожалуйста, обратитесь к странице конкретного драйвера для более подробной информации:

- Методы драйвера браузера для работы с окнами
- Методы драйвера рабочего стола для работы с окнами
- Методы драйвера Java для работы с окнами
- Методы SapDriver для работы с окнами

Скриншоты

BrowserDriver - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - поддерживает

ScreenDriver - поддерживает

Все драйвера поддерживают получение скриншота всего экрана. Захваченный скриншот можно вернуть в другом формате.

getScreenshotAsBytes

```
public byte[] getScreenshotAsBytes()
```

getScreenshotAsFile

```
public File getScreenshotAsFile()
```

getScreenshotAsBase64

```
public String getScreenshotAsBase64()
```

Также одни и те же методы могут использоваться для каждого конкретного элемента для захвата снимка элемента на экране. Пожалуйста, обратитесь к странице конкретного драйвера для более подробной информации:

- Методы драйвера браузера для работы с элементами интерфейса
- Методы драйвера рабочего стола для работы с элементами интерфейса
- Методы драйвера Java для работы с элементами интерфейса
- Методы SapDriver для работы с элементами интерфейса
- Методы драйвера экрана для работы с элементами интерфейса

Работа с UI элементами

Каждый драйвер имеет свой определенный тип элементов и свой способ построения поискового запроса (По объекту). Поэтому необходимо просмотреть дополнительные сведения на страницах конкретных драйверов.

Ниже приведены сведения о том, какие локаторы поддерживаются каким драйвером:

```
com.iba.kanclerrpa.engine.rpa.locator.BrowserSearch;
com.iba.kanclerrpa.engine.rpa.locator.DesktopSearch;
com.iba.kanclerrpa.engine.rpa.locator.JavaSearch;
com.iba.kanclerrpa.engine.rpa.locator.SapSearch;
com.iba.kanclerrpa.engine.rpa.locator.ScreenSearch;
```

Selector type	BrowserDriver	DesktopDriver	JavaDriver	SapDriver	ScreenDriver
By.id					
By.name					
By.xpath					
By.cssSelector					
By.className					
By.tagName					
By.linkText					
By.partialLinkText					
By.desktopSearch					
By.image					
By.anchor					
By.sapId					
By.sapName					

BrowserDriver - нативная поддержка Selenium (пожалуйста, перейдите на страницу Драйвер браузера чтобы увидеть дополнительную информацию)

DesktopDriver - поддерживает (пожалуйста, перейдите на страницу DesktopDriver чтобы увидеть дополнительную информацию)

JavaDriver - поддерживает (пожалуйста, перейдите на страницу Java драйвер чтобы увидеть дополнительную информацию)

SapDriver - поддерживает (пожалуйста, перейдите на страницу SAP-драйвер чтобы увидеть дополнительную информацию)

ScreenDriver - поддерживает (пожалуйста, перейдите на страницу Драйвер экрана чтобы увидеть дополнительную информацию)

findElement

Используется для поиска элемента и возвращает первый совпадающий WebElement, который можно использовать для будущих манипуляций с элементом.

```
public UiElement findElement(By by)
```

findElements

Аналогичен «findElement», но возвращает список соответствующих элементов UI. Чтобы использовать определенный элемент UI из списка, вам нужно пройтись по списку элементов, чтобы выполнить действие над выбранным элементом.

Каждый драйвер имеет свой определенный тип элементов и свой способ построения поискового запроса. Поэтому необходимо просмотреть дополнительные сведения на страницах конкретных драйверов.

```
public List<UiElement> findElement(By by)
```

waitForElement

Аналогичен «findElement», но не генерирует исключение сразу, если элемент не найден. Он будет ждать указанный тайм-аут, пока не появится элемент. Также поддерживает необязательный параметр «подавить исключение», который подавляет исключение в случае «true» значения, а метод вместо этого возвращает «null».

Также вместо поискового запроса метод может принимать объект функции. Используется для условных запросов (*ExpectedConditions.visibilityOfElementLocated(By.xpath("xpath_string")*)

```
public UiElement waitForElement(By by, int secondsToPoll)
public UiElement waitForElement(By by, int secondsToPoll, boolean suppressTimeoutException)

public UiElement waitForElement(Function<WebDriver, UiElement> function, int secondsToPoll)
public UiElement waitForElement(Function<WebDriver, UiElement> function, int secondsToPoll, boolean
suppressTimeoutException)
```

Keyboard, Mouse, Clipboard

⚠ **BrowserDriver** - поддерживает

DesktopDriver - поддерживает

JavaDriver - поддерживает

SapDriver - поддерживает

ScreenDriver - поддерживает

Все драйверы поддерживают устройства ввода, такие как клавиатура и мышь. Также есть доступ к системному буферу обмена со всех драйверов.

getInputDevices

Метод, который поддерживается всеми драйверами, возвращает абстракцию, которая позволяет вам получить доступ к устройствам мышь и клавиатура (имеет соответствующие геттеры).

```
public InputDevices getInputDevices()
```

Например:

```
import com.iba.kanclerrpa.engine.rpa.interactions.InputDevices;
import com.iba.kanclerrpa.engine.rpa.interactions.Mouse;
import com.iba.kanclerrpa.engine.rpa.interactions.Keyboard;

InputDevices inputDevices = getDriver().getInputDevices();
Mouse mouse = inputDevices.getMouse();
Keyboard keyboard = inputDevices.getKeyboard();
```

getKeyboard

Этот метод возвращает интерфейс (объект клавиатуры), которая позволяет выполнять действия с клавиатурой, такие как отправка, нажатие и отпускание клавиш.

```
public Keyboard getKeyboard()
```

Короткий пример того, как нажать комбинацию «Ctrl + S», которая обычно отвечает за сохранение:

```
import com.iba.kanclerrpa.engine.rpa.interactions.Keyboard;
import org.sikuli.hotkey.Keys;

Keyboard keyboard = driver.getInputDevices().getKeyboard();
keyboard.pressKey(Keys.CTRL);
keyboard.sendKeys("s");
keyboard.releaseKey(Keys.CTRL);
```

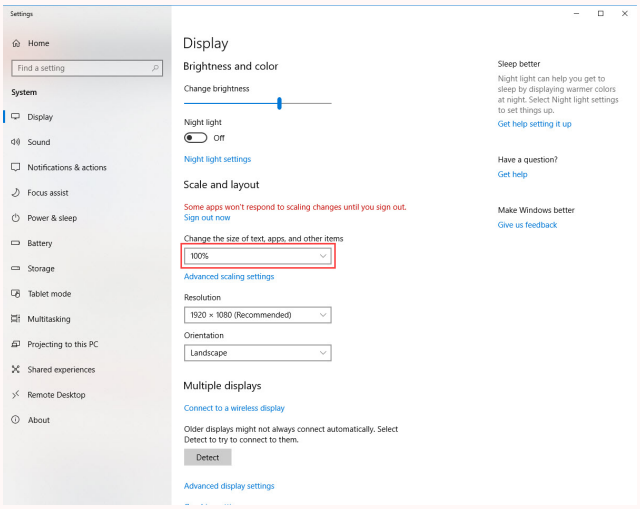
Класс клавиатуры использует библиотеку Selenium для отправки ключей, поэтому важно использовать константы ключей из **org.openqa.selenium.Keys**.

Также важно, чтобы в меню панели задач Windows был выбран **тот же язык клавиатуры**, что и **язык символов, который вы отправляете в методы клавиатуры**. В противном случае вы получите исключение.

getMouse

Прежде чем начать использовать действия мыши, убедитесь, что **масштабирование экрана Windows** установлено на **100%** , иначе робот не сможет щелкнуть нужные элементы.

1. Перейдите **Start** > **Settings** > **System** > **Display**.
2. В **Scale and Layout**, **выберите 100% для изменения размера текста, приложений и других элементов.** Изменения применяются автоматически.



Этот метод возвращает интерфейс (объект мыши), которая позволяет выполнять действия с помощью мыши, такие как нажатие правой и левой кнопок, перемещение мыши, двойной щелчок.

```
public Mouse getMouse()
```

Объект Mouse имеет различные типы методов для выполнения щелчка, перемещения мыши и т. д., которые принимают такие аргументы, как целочисленные координаты «x» и «y», объект Point. Также перемещение мыши можно выполнять со смещением:

```
public void click(Point where)
public void click(int x, int y)

public void doubleClick(Point where)
public void doubleClick(int x, int y)

public void contextClick(Point where)
public void contextClick(int x, int y)

public void mouseDown(Point where)
public void mouseDown(int x, int y)

public void mouseUp(Point where)
public void mouseUp(int x, int y)

public void mouseMove(Point where)
public void mouseMove(int x, int y)
public void mouseMove(Point where, long xOffset, long yOffset)
public void mouseMove(int x, int y, long xOffset, long yOffset)
```

Ниже приведен пример того, как выполнить щелчок мышью и перемещение со смещением:

```
driver.getInputDevices().getMouse().click(10, 10);
driver.getInputDevices().getMouse().mouseMove(10, 10, 40, 40);
```

clipboard

Этот метод возвращает интерфейс (объект буфера обмена), которая позволяет получать, устанавливать и очищать текст буфера обмена.

```
public Clipboard clipboard()
```

Объект буфера обмена содержит следующие методы:

```
public String getText()
public void setText(String text)
public void clean()
```

Короткий пример того, как получить значение из буфера обмена:

```
String valueFromClipboard = getDriver().clipboard().getText();
```

Java драйвер

На этой странице описываются только специальные функции и методы драйвера Java, которые могут отличаться в других драйверах.

Единая для всех драйверов реализация описана на Странице драйвера. Пожалуйста, ознакомьтесь с ней заранее.

Драйвер Java автоматизирует приложения с помощью пользовательского интерфейса Java, созданного из пакета java.awt.*. Это, например, включает Swing и Oracle Forms.

Драйвер Java состоит из реализации драйвера **Канцлер RPA** и **агента Java.**

Агент Java представляет собой пакет jar, реализующий JsonWire протокол. Он подключается к целевому Java-приложению и запускает http-сервер.

Реализация драйвера Java является частью движка. Он использует протокол JsonWire, предоставляя разработчику знакомый интерфейс программирования, подобный селениуму.

Подготовка узла для автоматизации Java

Следующие шаги должны быть выполнены вручную для подготовки узла к автоматизации Java.

1. Если вы запускаете процесс автоматизации вручную, укажите путь к **java-agent.jar** в свойствах системы. Агент узла делает это автоматически.

```
// is equivalent to VM option -Dmarathon.agent.file="D:/path/to/java-agent.jar"
System.setProperty("marathon.agent.file", "D:/path/to/java-agent.jar");
```

java.policy должна быть обнавлена в соответствии с политикой доступа java-agent.jar. Путь к jar должен быть такой же, как указан в п.1. Для запуска агентов узла необходимо обратиться к jar, который хранится в пакете агента узла.

```
grant codeBase "file:/C:/<Path to node directory>/java-agent.jar" {
   permission java.security.AllPermission;
};
```

3. JDK для запуска процесса автоматизации и приложение Java, которое вы автоматизируете, должны быть одинаковыми. По умолчанию агент узла использует ту же JDK для запуска автоматизированного процесса, что использует сам. Чтобы изменить JDK для процесса автоматизации, необходимо установить переменную JAVA_HOME в конфигурации узла. См. Кон фигурационные параметры.

Архитектура процессора должна совпадать: если целевое приложение работает под java x64, то автоматизированный процесс также должен быть запущен под java x64.

Инициализация драйвера

Вы можете инициализировать драйвер в процессе автоматизации следующим образом:

```
@Driver(DriverParams.Type.JAVA)
private JavaDriver jDriver;
```

Открытие новых приложений

См. Как запустить или подключиться к Java-приложению.

Навигация в браузере

Метод getCurrentUrl()

Метод возвращает свойства окна (т.е. верхнего контейнера пользовательского интерфейса Java) в виде строки JSON.

```
// returns {"title":"InputVerificationDemo","tagName":"window","component.class.name":"javax.swing.JFrame"}
String url = driver.getCurrentUrl();
```

Метод getTitle()

Метод возвращает заголовок java.awt.Frame или java.awt.Dialog.

```
String title = driver.getTitle();
```

Метод getWindowHandle()

Метод возвращает хэш-код идентификатора java.awt.Window (верхний контейнер пользовательского интерфейса Java), закодированный как шестнадцатеричная строка.

```
// returns "7abde323"
String handle = driver.getWindowHandle();
```

Метод getWindowHandles()

Метод возвращает все идентификаторы окон, зарегистрированные в `sun.awt.AppContext`.

```
Set<String> handles = driver.getWindowHandle();
```

Метод close()

Метод вызывает метод dispose() для текущего `java.awt.Window`.

```
drive.close();
```

Метод switchTo()

Драйвер Java может переключаться на окна только внутри одного приложения Java. Чтобы взять на себя управление другим приложением, необходимо создать еще один драйвер.

Драйвер Java поддерживает переключение на окно по заголовку или дескриптору:

- переключается на окно по названию

```
driver.switchTo().window("Login Success");
```

- последовательно переключается на каждое окно по его идентификатору

```
Set<String> windowHandles = getDriver().getWindowHandles();
for (String handle : windowHandles ) {
        driver.switchTo().window(handle);
}
```

Работа с элементами интерфейса

Инспектор

Для проверки приложений Java мы предлагаем использовать инструмент Java inspector, описанный в Как использовать Java инспектора.

Поиск элементов интерфейса

Ниже приведена таблица с локаторами, которые поддерживает драйвер Java:

Selector type
JavaSearch.id
JavaSearch.name
JavaSearch.cssSelector
JavaSearch.className
JavaSearch.tagName

Селекторы `JavaSearch.id` и `JavaSearch.name` позволяют искать элемент по значение, возвращаемому методом `java.awt.Component.getName()`.

```
// these are the same
driver.findElement(JavaSearch.id("username"));
driver.findElement(JavaSearch.name("username"));
```

Селектор `JavaSearch.className` позволяет искать элемент по полному имени класса, включая все подклассы.

```
// returns the elements of class javax.swing.JPasswordField
driver.findElements(JavaSearch.className("javax.swing.JPasswordField"));

// returns the elements of class javax.swing.JTextField including JPasswordField
// because JPasswordField extends JTextField
driver.findElements(JavaSearch.className("javax.swing.JTextField"));
```

Канцлер RPA реализует средство поиска CSS для приложений Java. Вы можете использовать метод findElement(s), чтобы быстро найти компонент в приложении.

Селектор	Описание	Пример

название тэга	Выбирает элемент с заданным тэгом. Тэг вычисляется путем нахождения суперкласса Swing/AWT компонента и преобразования CamelCase в Camel-Case.	`driver.findElement(JavaSearch.tagName("text-field"));`
		`// JTextField`
		`driver.findElement(JavaSearch.tagName("spinner")); // JSpinner`

*	Получает все элементы, перенастраивает список элементов	`driver.findElements(JavaSearch.cssSelector("*"));`

.	Возвращает тот же элемент	`// tree and tree_1 are same`
		`tree = driver.findElements(JavaSearch.cssSelector("tree"));`
		`tree_1 = tree.findElement(JavaSearch.cssSelector("."));`

#name	Возвращает элемент с заданным именем	`// the element has name "loanAmount" set through java.awt.Component#setName` `driver.findElement(JavaSearch.cssSelector("#loanAmount"));`

[attribute=value]	Возвращает элемент с заданным значением атрибута. Возможно задавать следующие условия поиска: *= (содержит), /= (соответствует регулярному выражению), = (равно), ^= (начинается с), $= (заканчивается).	`// finds all the buttons whose text is equal to "Click Me"`
		`driver.findElement(JavaSearch.cssSelector("button[text='Click Me']"));`
		
		`// finds all the buttons whose text starts with "Click"`
		`driver.findElement(JavaSearch.cssSelector("button[text^='Click']"));`

:	Возвращает элементы, для котрых возвращается значение true. Доступные псевдоклассы: `selected`, `enabled`, `displayed`, `hidden` and `instance-of("")`	`// finds all the text-fields which are enabled` `driver.findElement(JavaSearch.cssSelector("text-field:enabled"));`
::	найти псевдоэлементы	

- **JComboBox**
  - Доступные псевдоэлементы: **nth-option(index)** и **all-options**.
- **JList**
  - Доступные псевдоэлементы: **nth-item(index)** и **all-items**.
- **JEditorPane**
  - Доступные псевдоэлементы: **tag(name)**, где имя может быть **a**, **ol**, **ul** и тд.
- **JTabbedPane**
  - Доступные псевдоэлементы: **nth-tab(index)**, **all-tabs** и **selected-tab**.
- **JTableHeader**
  - Доступные псевдоэлементы: **nth-item(index)** и **all-items**.
- **JTable**
  - Доступные псевдоэлементы: **header**, **mnth-cell(row, column)**, **all-cells** и **mnth-cell-editor(row, column)**.
- **JTree**
  - Доступные псевдоэлементы: **nth-node(index)**, **all-nodes, editor** и **root**.

JComboBox

```
// Returns 2nd option from the
combo-box
driver.findElement(JavaSearch.
cssSelector("combo-box::nth-option
(2)"));
// OR
combobox = driver.findElement
(JavaSearch.tagName("combo-box"));
combobox.findElement(JavaSearch.
cssSelector(".::nth-option(2)"));

// Returns list of all options
from JComboBox driver.findElement
(JavaSearch.cssSelector("combo-
box::all-options"));
```

JList

```
// Returns 4th item from the list
driver.findElement(JavaSearch.
cssSelector("list::nth-item(4)"));
// OR
list = driver.findElement
(JavaSearch.tagName("list"));
list.findElement(JavaSearch.
cssSelector(".::nth-item(4)"));

// Returns list of all the options
from the list element
driver.findElement(JavaSearch.
cssSelector("list::all-items"));
```

JEditorPane

```
editor = driver.findElement
(JavaSearch.cssSelector("editor-
pane"));
editor.findElement(JavaSearch.
cssSelector(".::tag('a')
[text='Title Page']"));
```

JTabbedPane

```
// Returns selected tab
tabbedPane = driver.findElement
(JavaSearch.cssSelector("tabbed-
pane"));
tabbedPane.findElement(JavaSearch.
cssSelector(".::selected-tab"));
```

JTableHeader

```
tableHeader = driver.findElement
(JavaSearch.tagName("table-
header"));
tableHeader.findElement(JavaSearch.
tagName(".::nth-item(3)"));
```

JTable

```
table = driver.findElement
(JavaSearch.cssSelector("table"));
// To get particular cell
table.findElement(JavaSearch.
cssSelector(".::mnth-cell(2, 3)"));
// To get all the cells
table.findElement(JavaSearch.
cssSelector(".::all-cells"));
// To get the cell editor
table.findElement(JavaSearch.
cssSelector(".::mnth-cell-editor
(2, 3)"));
```

JTree

```
tree = driver.findElement
(JavaSearch.cssSelector("tree"));
// To get particular node
tree.findElement(JavaSearch.
cssSelector(".::nth-node(2)"));
// To get all the nodes
tree.findElement(JavaSearch.
cssSelector(".::all-nodes"));
// To get the root
tree.findElement(JavaSearch.
cssSelector(".::root"));
// To get the editor
tree.findElement(JavaSearch.
cssSelector(".::editor"));
```

Для получения более подробной информации об элементах и селекторах, а также примеры см. https://marathontesting.com/marathonite-user-guide/java-swing-components/

Как запустить или подключиться к Java-приложению

Введение

В настоящее время драйвер Java поддерживает несколько режимов запуска приложения.

Каждый из них имеет свою специфику, поэтому давайте рассмотрим подробнее.

Запуск нового Java-приложения

Запуск из JAR

При запуске приложения из jar-файла в качестве первого аргумента необходимо передать LaunchMode.EXECUTABLE_JAR, а вторым - путь к jar-файлу приложения.

```
jDriver.get(JavaDriver.LaunchMode.EXECUTABLE_JAR, "D:\\path\\to\\swing-application.jar");
```

Запуск из JNLP

JNLP устарел, начиная с Java 9, и удален из Java 11. Мы рекомендуем запускать JNLP под Java 8.

Первоначальная настройка должна быть выполнена для автоматизации приложений JNLP.

- чтобы избежать предупреждений "Ваша версия Java устарела":

Откройте файл `%userprofile%\AppData\LocalLow\Sun\Java\Deployment\deployment.properties` и добавить строку в начале:

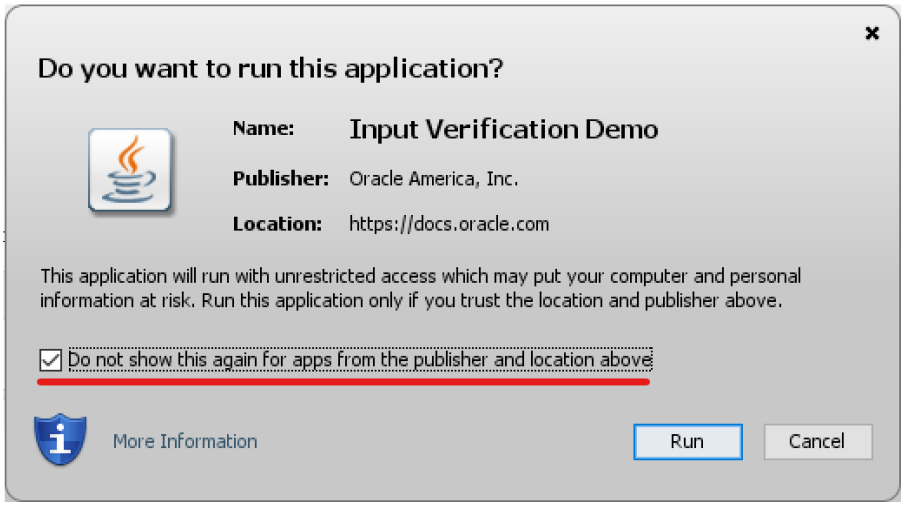
deployment.properties

```
deployment.expiration.check.enabled=false
```

- если запуск приложения заблокирован настройками безопасности:

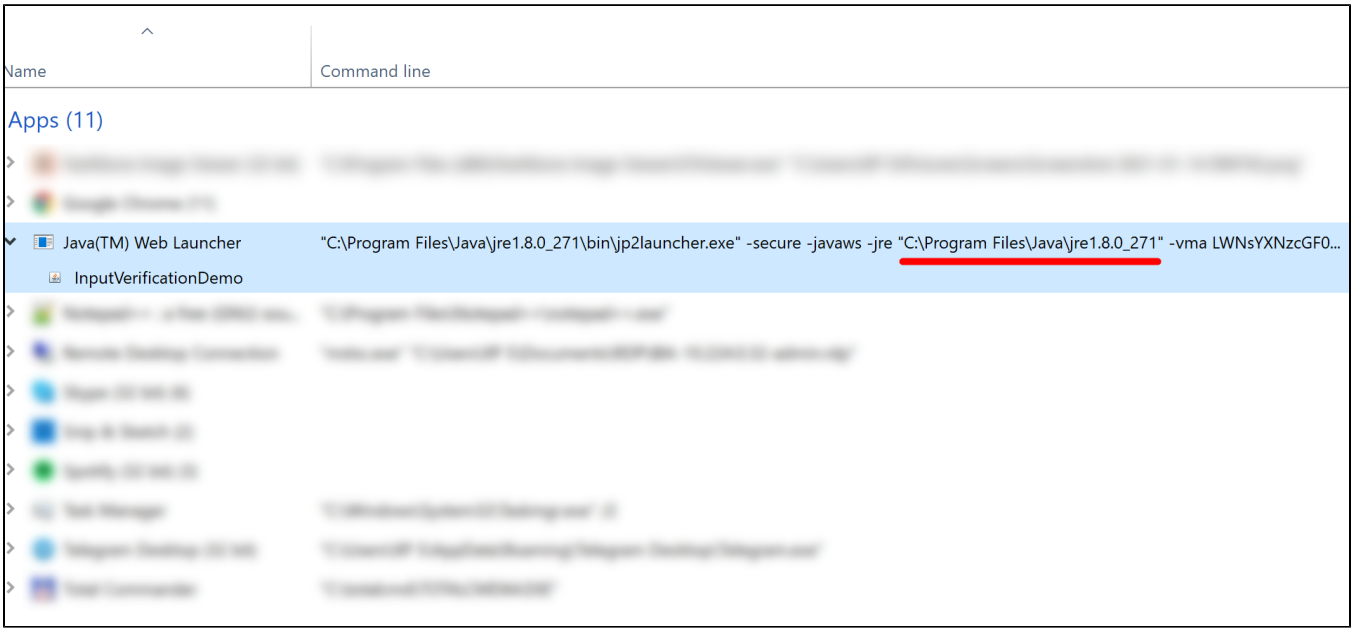
В Windows откройте панель управления->Java->Security->Edit Site List... и добавте путь к jnlp file, т.е. `file:///D:/InputVerificationDemo.jnlp`

При первом запуске появится запрос безопасности, установите флажок "Больше не показывать...", а затем запустите



- если агент Java подключается к приложению, но не может запустить сервер JsonWire:

Вам следует проверить `javaws.policy` файл. Чтобы найти его, сначала определите путь к JRE, на котором выполняются ваши приложения jnlp.
Запустите файл jnlp вручную, затем проверьте диспетчер задач Windows. Найдите свое приложение jnlp в процессе Java Web Launcher, `-jre` аргумент из командной строки укажет путь, который вам нужен.



Теперь перейдите в папку JRE - согласно приведенному выше экрану, это `C:\Program Files\Java\jre1.8.0_271` `javaws.policy` в `lib\security\` папке. Итак, окончательный путь `c:\Program Files\Java\jre1.8.0_271\lib\security\javaws.policy`

Откройте `javaws.policy` в текстовом редакторе и добавьте фрагмент в конец файла:

java.policy

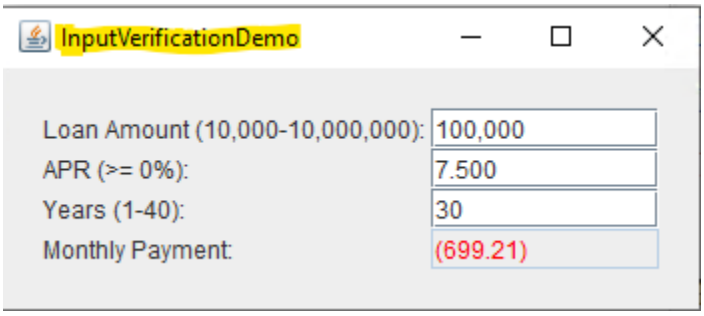
```
grant codeBase "file:/path/to/java-agent.jar" {
    permission java.security.AllPermission;
  };
```

Когда все настроено и сделано, приложение jnlp можно запустить, передав аргументы методу driver.get:

- `LaunchMode.JAVA_WEBSTAR`
- путь к файлу jnlp

- заголовок начального окна

Заголовок начального окна - это, по сути, заголовок в верхней части окна, который обычно появляется первым после исчезновения экрана загрузки Java.



Окончательный код может выглядеть следующим образом:

```
jDriver.get(JavaDriver.LaunchMode.JAVA_WEBSTART, "D:\\InputVerificationDemo.jnlp", "InputVerificationDemo");
```

Запуск из командной строки

Драйвер Java также может запускать приложение Java из командной строки. Для этого просто пройдите `LaunchMode.COMMAND_LINE` и путь к скрипту командной строки `driver.get` метод

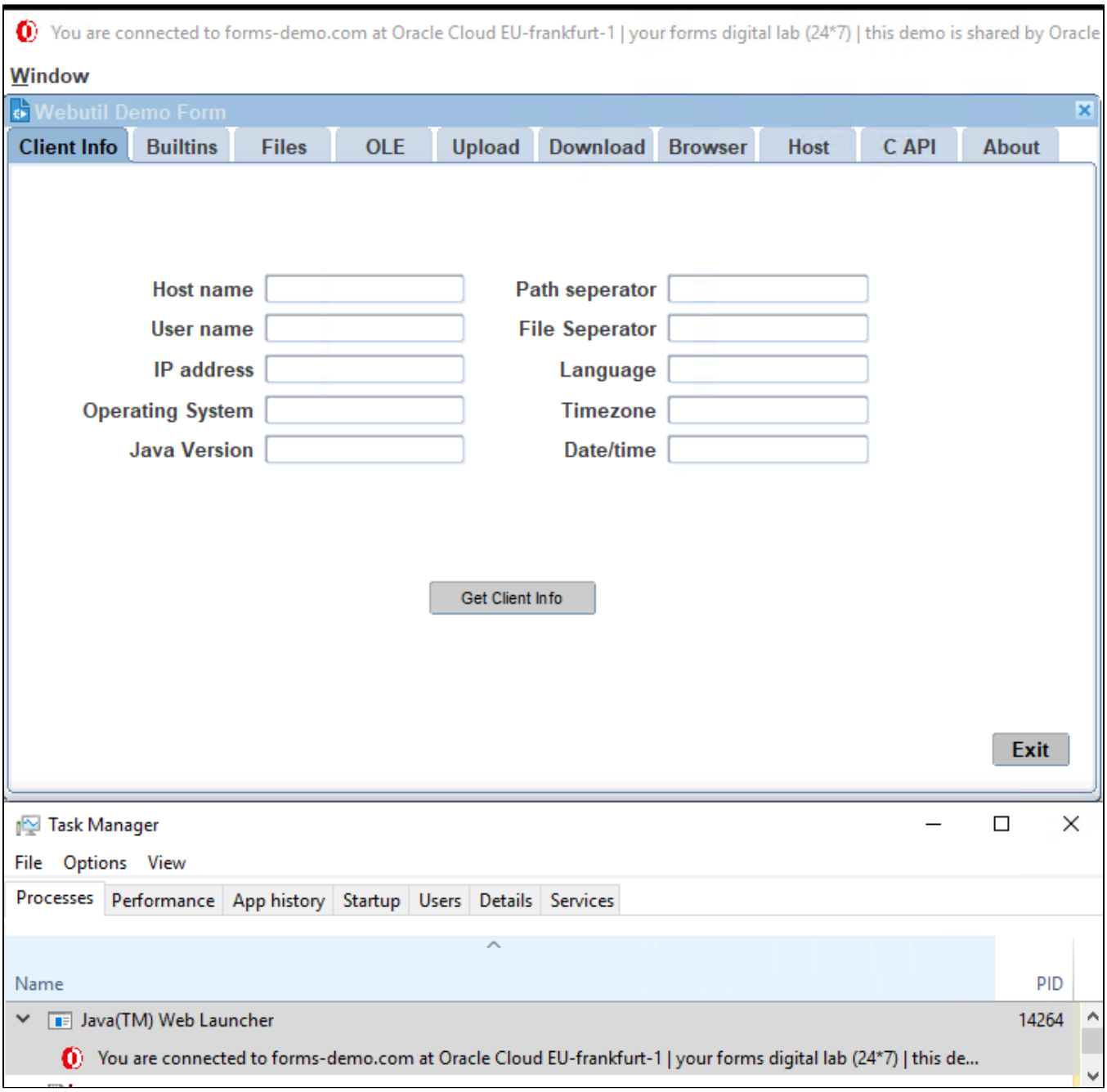
```
jDriver.get(JavaDriver.LaunchMode.COMMAND_LINE, "D:\\run-application.bat");
```

Подключиться к запущенному Java-приложению

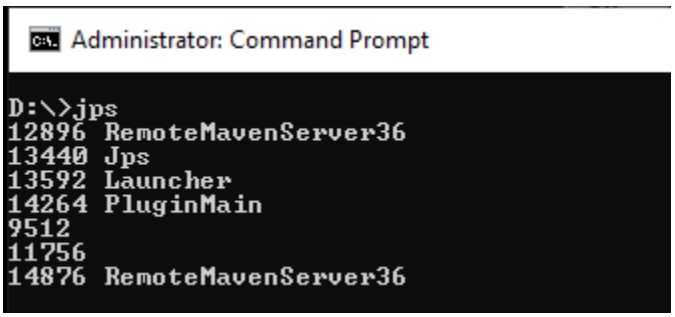
Драйвер Java предлагает возможность подключения к уже работающему Java-приложению, используя его имя, когда запуск нового экземпляра приложения одновременно с драйвером не нужно.

Чтобы выяснить имя приложения, запустите его вручную и найдите идентификатор процесса, используя выбранный вами инструмент.

Например, здесь мы открыли webutl demo от Oracle и обнаружил, что его PID равен 14264 в диспетчере задач Windows.



Далее нам нужно запустить утилиту jps из jdk в командной строке, чтобы найти имя приложения, соответствующее 14264.



Итак, имя приложения `PluginMain`. Передадим его в метод `driver.get` вместе с `LaunchMode.JAVA_ATTACH`

```
jDriver.get(JavaDriver.LaunchMode.JAVA_ATTACH, "PluginMain");
```

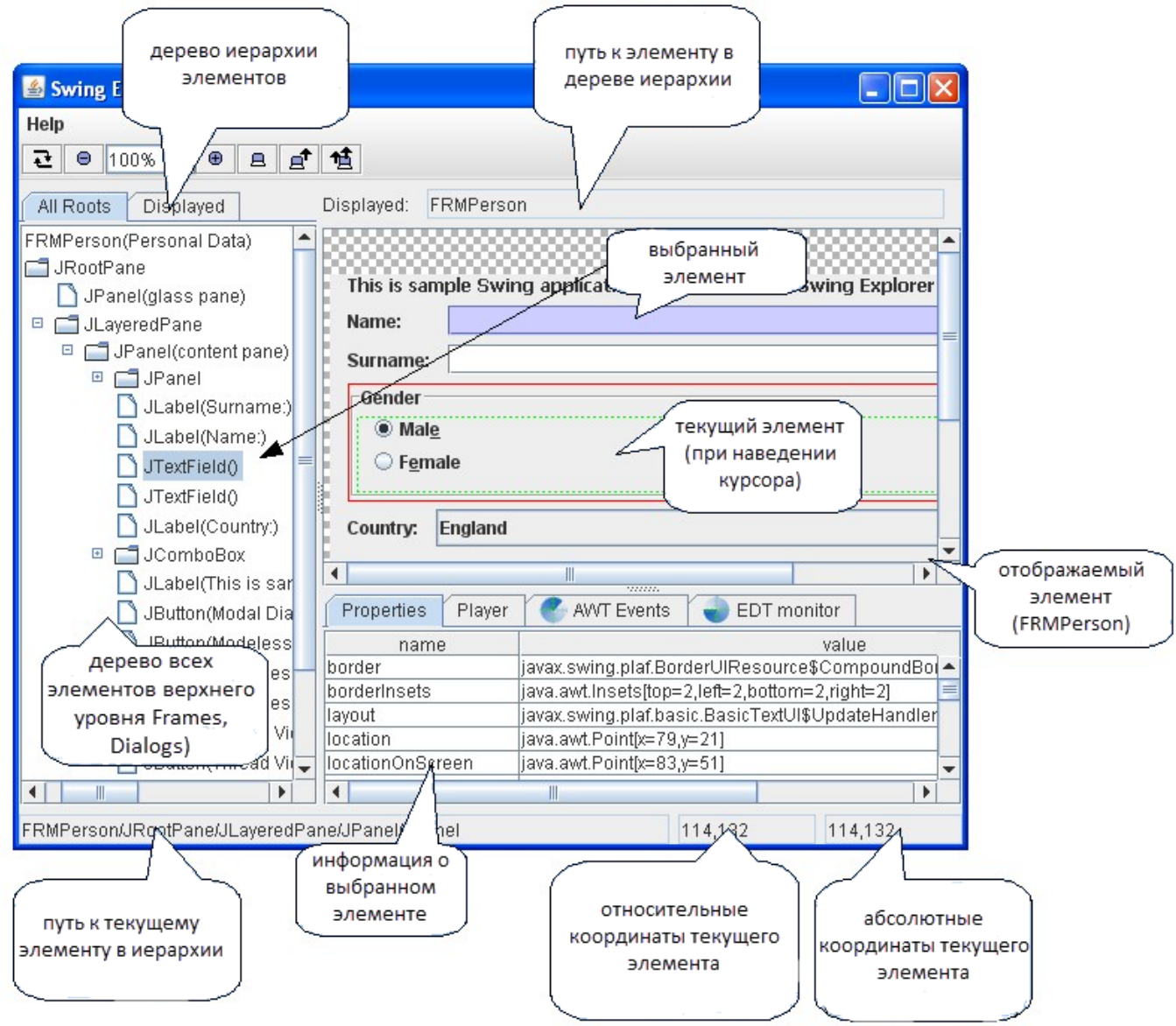
Как использовать Java-инспектор

Введение

Java инспектор для проверки графических интефейсов Java Swing GUIs. Он работает аналогично подключаемым модулям разработчика для браузеров HTML, но для инструментария Java Swing.

С помощью Swing Explorer вы можете визуально просматривать иерархию компонентов приложения.

Ниже показано, как базовое приложение Swing проверяется Java инспектор.



Скачать Java Инспектор

Пожалуйста, найдите инспектора здесь: java-inspector.zip

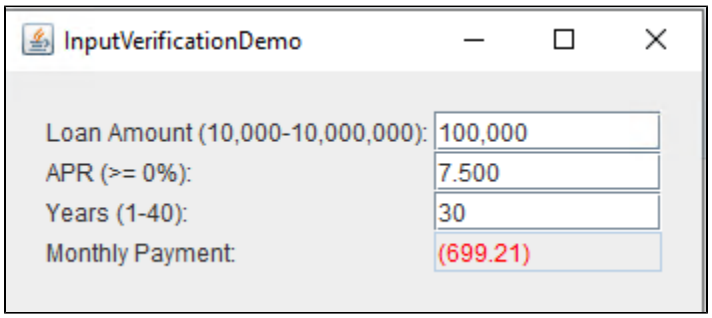
Запуск Java-инспектора

Для запуска инспектора используйте `startup` команду, расположенную в корневом каталоге.

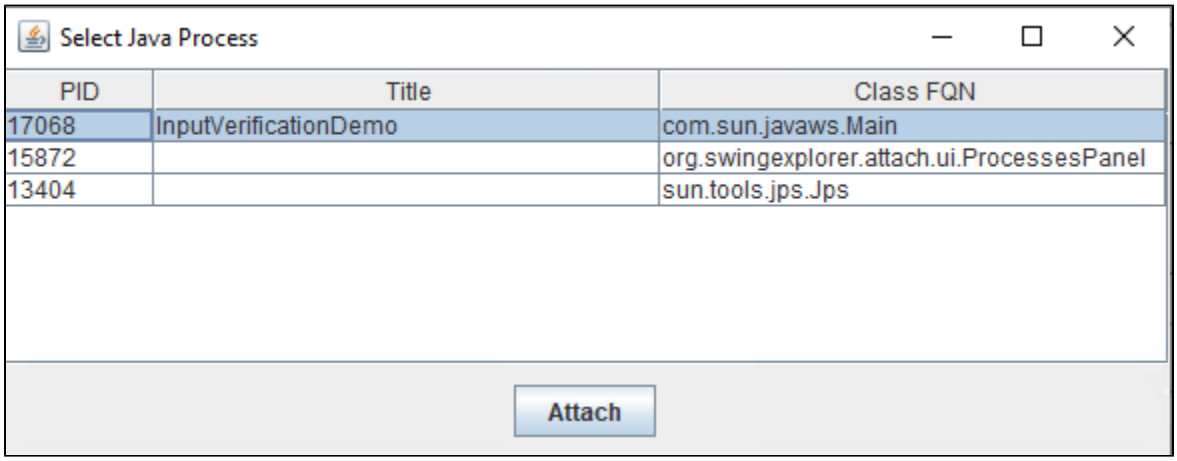
Инспектор запускается с версией java по умолчанию, которая зависит от вашей операционной системы - это может быть x64, x86 или другая. Если вам нужно проверить приложение, работающее под определенной архитектурой ОС, измените путь java в `startup.bat` файле.

При запуске на первом экране Java Инспектора отображается список процессов Java. Не все из них можно проверить, поэтому ответственность за правильное определение требуемого приложения лежит на пользователе.

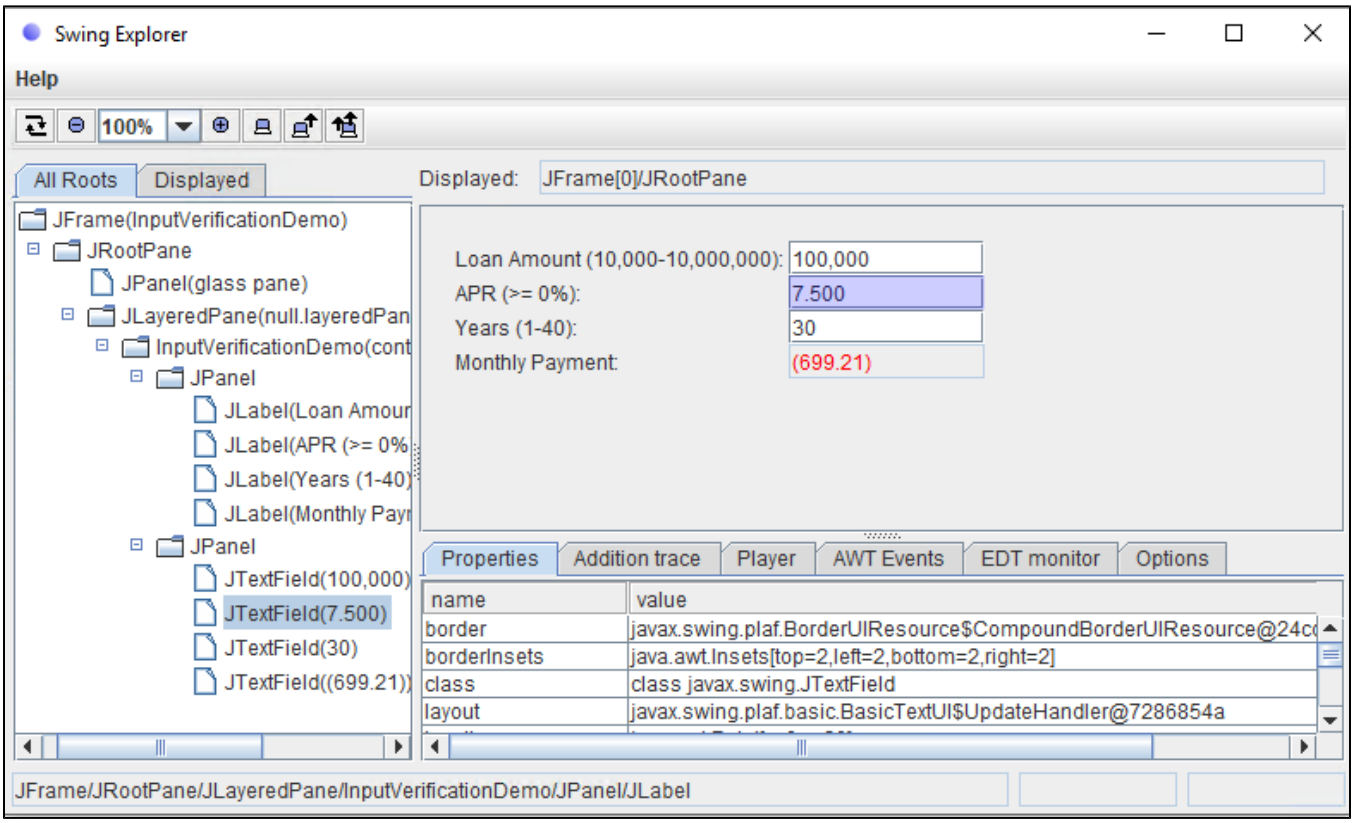
Рассмотрим пример: запустим InputVerification из Oracle документация



Теперь запустите инспектор Java, используя `startup`



Несмотря на то, что InputVerification — это единственное настольное Java-приложение, работающее на компьютере, мы по-прежнему видим другие Java-процессы. Очевидно, что мы находимся в первом списке с заголовком «InputVerificationDemo». Пойдем и прикрепим к нему



После прикрепления инспектора появляется окно проводника с деревом элементов, отображаемым на левой панели.

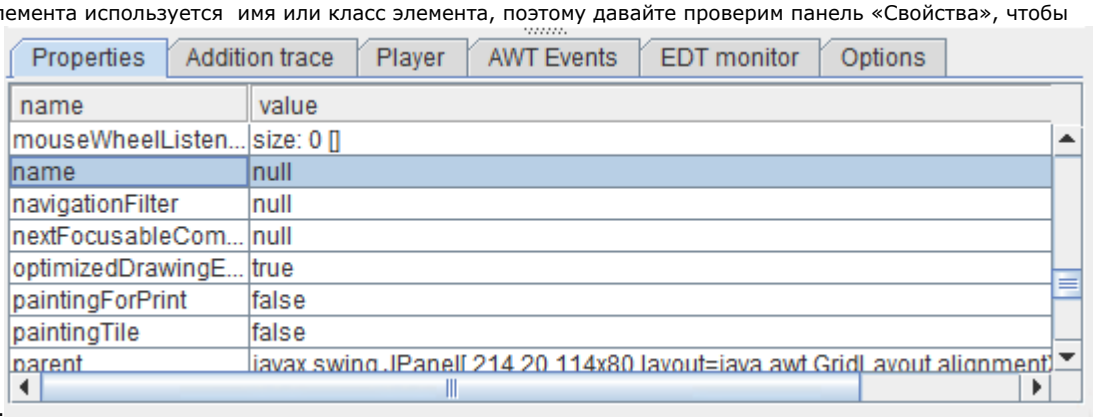
Двойной щелчок по элементу дерева отобразит элемент внутри инспектора, а один щелчок выделит его.

Вы можете наоборот выделить его прямо в отображении и элемент автоматически подсветится в дереве.

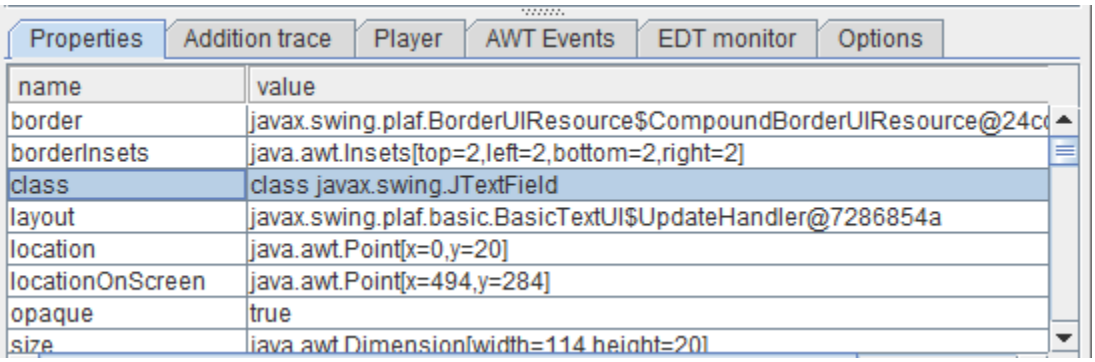
Получение селектора элементов

Скажем, мы хотим, чтобы наш робот изменил значение APR, это поле, которое мы выбрали на последнем экране.

Обычно для поиска элемента используется имя или класс элемента, поэтому давайте проверим панель «Свойства», чтобы



увидеть его значения.



Поскольку у нас нет уникального имени для этого элемента и на текущем экране есть несколько элементов с одинаковым именем класса, мы найдем все элементы и получим тот, который нам нужен, по индексу

```
        List<WebElement> elements = driver.findElements(By.className("javax.swing.JTextField"));

        WebElement loanAmountInput = elements.get(0);
        WebElement aprInput = elements.get(1); // this our APR field
        WebElement yearsInput = elements.get(2);
        WebElement paymentInput = elements.get(3);

                // now let's change its value
        aprInput.clear();
        aprInput.sendKeys("12.5");
```

Методы драйвера Java для работы с окнами

Методы управления окном:

Метод getSize()

`getSize()` для `javax.swing.JFrame`

```
java.awt.Dimension size = driver.manage().window().getSize();
```

Метод setSize()

Вызывает `setSize` для `javax.swing.JFrame`

```
driver.manage().window().setSize(new Dimension(600, 300));
```

Метод getPosition()

Вызывает `getLocation()` для `javax.swing.JFrame`

```
org.openqa.selenium.Point = driver.manage().window().getPosition();
```

Метод setPosition()

Вызывает `setLocation` на `javax.swing.JFrame`

```
driver.manage().window().setPosition(new Point(10, 10));
```

Метод maximize()

Вызывает `setExtendedState(Frame.MAXIMIZED_BOTH)` для `javax.swing.JFrame` если тот поддерживается окном.

```
driver.manage().window().maximize();
```

Методы драйвера Java для работы с элементами интерфейса

Что такое элемент драйвера Java?

Взаимодействие с элементом драйвера Java подобно на то, как взаимодействует Selenium с веб-элементом, но осуществляется посредством Java драйвера, вместо Selenium WebDriver.

Большинство методов драйвера Java действуют аналогично методам Selenium WebDriver, поэтому мы рассмотрим только различия.

Метод Submit()

вызывает `stopEditing()` для `javax.swing.JTree`, который сохраняет любые изменения, которые в настоящее время выполняются в ячейке.

Метод getTagName()

возвращает название класса элемента через дефис

```
JavaElement element = driver.findElement(JavaSearch.className("javax.swing.JTextField"))
String tagName = element.getTagName(); //returns "text-field"
```

Not implemented

Эти методы не реализованы и будут вызывать `org.openqa.selenium.UnsupportedCommandException`:

```
getRect()
getCssValue()
isSelected()
getScreenshotAs()
```

SAP-драйвер

На этой странице описываются только специальные функции и методы драйвера SAP, которые могут отличаться в других драйверах.

Единая для всех драйверов реализация описана на странице драйвера. Пожалуйста, прочитайте это заранее.

Подготовьте узел для SAP автоматизации

Драйвер SAP автоматизирует действия, выполняемые с настольным клиентом SAP. Драйвер SAP использует компонент ActiveX, который требует подготовки узела к автоматизации SAP:

- Настройте SAP GUI для Windows 7.50 в соответствии с https://www.sap.com/documents/2017/05/bc8d025a-b97c-0010-82c7-eda71af511fa.html
- Примените исправление реестра **sap_driver_fix.reg** из пакета агента узла.
- Установите Microsoft Visual C++ Redistributable 2015-2019 x64 и/или x86

Вам также необходимо настроить подключения клиента SAP, которые может выполнять пользователь SAP.

Enable SAP Scripting on SAP Server

Рекомендуется скачать трекер с https://tracker.stschnell.de/

После выполнения этого шага необходимо иметь возможность запускать трекер для SAP Frontend и видеть его дерево объектов.

Включите скриптинг на стороне клиента, отключите уведомления:

Включить скрипты на сервере:

Войдите в созданное соединение, используя свои учетные данные

Запустите код транзакции rz11

Введите `sapgui/user_scripting` имя параметра и нажмите Enter

Убедитесь, что текущее значение установлено на TRUE, в противном случае измените его с помощью верхнего меню.

Для быстрой справки, пожалуйста, обратитесь к примеру SAP Material Management Demo.

Инициализация драйвера

Вы можете инициализировать драйвер в процессе автоматизации следующим образом:

```
@Driver(Type.SAP)
private SapDriver sapDriver;
```

Открытие и закрытие приложения SAP

Чтобы использовать драйвер SAP, вы должны сначала запустить сеанс SAP. Для этого вы можете либо использовать настольный драйвер, либо использовать утилиту sapshcut, входящую в состав дистрибутива SAP Logon.

Откройте SAP с помощью утилиты sapshcut

1. Запустите команду sapshcut со всеми необходимыми аргументами, такими как пользователь и пароль, SID и клиент и другие.

   ```
   String sapPath = "C:/Program Files (x86)/SAP/FrontEnd/SAPgui/sapshcut";
   String user = "userid";
   String password = "password";
   String system = "S4H";
   String client = "100";

   String command = String.format("\"%s\" -user=%s -pw=%s -language=EN -system=%s -client=%s -command=%s -
   trace=0",
                   sapPath, user, password, system, client);

   Runtime.getRuntime().exec(command);
   ```

   Запустите `sapshcut -help` в командной строке, чтобы узнать больше доступных аргументов

2. Через определенное время, когда появится окно SAP Frontend, сеанс SapDriver можно безопасно инициализировать.

   ```
   sapDriver.sleep(6000); // sleep until SAP Frontend window shows up
   sapDriver.initSession();
   ```

Откройте SAP с помощью DesktopDriver

1. Запустите SAP Logon с помощью DesktopDriver.

   ```
   @Driver(Type.DESKTOP)
   private DesktopDriver desktopDriver;
   ...

   desktopDriver.get("C:/Program Files (x86)/SAP/FrontEnd/SAPgui/saplogon.exe");
   ```

2. Выберите необходимое соединение из списка и запустите его.

   ```
   DesktopSearch.UIQuery connectionQuery = DesktopSearch.UIQuery.builder().tagName("Text").name
   (sapConnectionName).build();
   DesktopElement connection = getDriver().findElement(DesktopSearch.query(connectionQuery));
   connection.sendKeys(Keys.ENTER);
   ```

3. Откроется окно SAP Frontend, которое подлежит автоматизации.

```
@Driver(Type.SAP)
private SapDriver sapDriver;

sapDriver.initSession();
```

4. Войдите в SAP Frontend с необходимыми учетными данными и выполните другие автоматизированные процессы, с помощью драйвера SAP.

```
sapDriver.findElementById("wnd[0]/usr/txtRSYST-BNAME").setText(user);
sapDriver.findElementById("wnd[0]/usr/pwdRSYST-BCODE").setText(password);
sapDriver.findElementById("wnd[0]").sendKeys(SapKeyCode.ENTER);

// other actions after user has been authenticated
```

5. Выход из SAP Frontend с помощью драйвера SAP

```
// logs out without prompt using command input
sapDriver.findElementById("wnd[0]/tbar[0]/okcd").setText("/nex");
```

Закройте SAP

Перед закрытием окна SAP важно выйти, т. е. уничтожить SapDriver, чтобы предотвратить утечку памяти или другие проблемы, связанные с базовыми системными компонентами.

```
sapDriver.quit();
```

Оставшееся окно SAP можно закрыть, убив процесс saplogon.

```
Runtime.getRuntime().exec("TASKKILL /F /IM saplogon.exe /T");
```

Если запущен с помощью DesktopDriver, SAP также можно закрыть, отправив клавиши Alt+F4.

```
desktopDriver.getKeyboard().sendKeys(Keys.chord(Key.ALT, Key.F4));
```

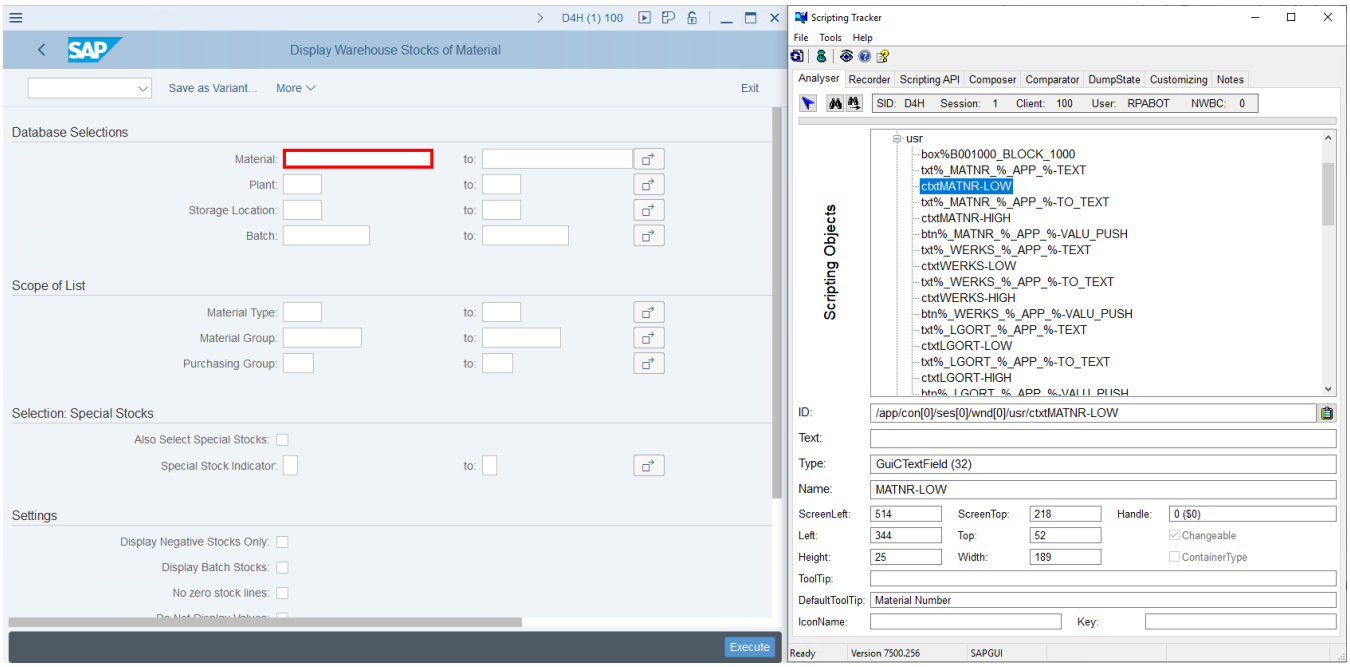
Работа с элементами UI

Инспектор

Чтобы определить идентификатор элемента SAP, запустите Scripting Tracker и обновите дерево объектов. Далее выберите нужный элемент с помощью кнопок со стрелками и скопируйте значение из поля ID field.

Ниже приведен пример получения идентификатора для поля ввода материала.

```
@FindBy(xpath = "/app/con[0]/ses[0]/wnd[0]/usr/ctxtMATNR-LOW")
private SapElement materialField;
```



Найти элементы

Ниже приведена таблица с локаторами, которые поддерживает SapDriver:

Selector type
SapSearch.sapId
SapSearch.sapName

Инициализированный драйвер SAP можно использовать для поиска элементов страницы.

Для этого идентификатор поля должен быть передан в `sapDriver.findElement(By sapId)`.

Теперь мы можем получить доступ к этому элементу следующим образом:

```
SapElement descriptionInput = sapDriver.findElement(SapSearch.sapId("wnd[0]/usr/txtRSYST-BNAME"));
descriptionInput.setText("KanclerRPA material");
```

Методы SapDriver для работы с окнами

Текущий объект окна можно получить с помощью метода:

```
Window window = driver.manage().window();
```

Методы управления окном:

Метод maximize()

Развернуть текущее окно.

```
driver.manage().window().maximize();
```

Метод focus()

Вызов этого метода фокусируется на текущем окне.

```
((SapWindow) driver.manage().window()).focus();
```

Метод getSize()

Возвращает объект измерения со свойствами «ширина» и «высота».

```
Dimension windowDimension = driver.manage().window().getSize();

System.out.println("Current window size is: " + windowDimension.getWidth() + "x" + windowDimension.getHeight());
```

Метод getPosition()

Возвращает текущую позицию окна на экране. Значение возвращается как объект «Точка» со свойствами «x» и «y».

```
Point windowPosition = driver.manage().window().getPosition();

System.out.println("Current window position is: " + windowPosition.getX() + "x" + windowPosition.getY());
```

Методы SapDriver для работы с элементами интерфейса

SapDriver на основе ActiveXComponent из библиотеки Jacob (JAVA-COM Bridge), который соответствует элементу структуры приложения SAP.

```
SapPElement e1 = getDriver().findElementById("foo"); SapPElement e2 = getDriver().findElementByName("bar");
```

Методы работы с элементами интерфейса:

Метод click()

Эмулирует ручное нажатие кнопки

Метод getDisplayedText()

Возвращает значение свойства Displayed Text, которое содержит текст в том виде, в котором он отображается на экране, включая предшествующие или завершающие пробелы.

Метод sendKeys()

Принимает в качестве аргумента код клавиши или сочетания клавиш. Выполняет базовый вызов `setText`.

```
element.sendKeys(SapKeyCode.CTRL_V);
element.sendKeys(SapKeyCode.ENTER);
```

Метод setText()

Используется для изменения отображаемого текста элемента, такого как метка, текстовое поле или кнопка, во время выполнения.

Метод selectDropDownByKey()

Устанавливает значение свойства «key», которое является ключом текущего выбранного элемента.

Метод getTableRowByIndex()

Метод принимает в качестве параметра индекс строки таблицы и возвращает соответствующий этой строке SapPElement. Индексация, поддерживаемая этой функцией, не сбрасывает индекс после прокрутки, а подсчитывает строки, начиная с первой строки, относительно первой позиции прокрутки. Если выбранная строка в данный момент не видна, возникает исключение.

```
SapPElement row = table.getTableRowByIndex(3);
```

Методы checkBoxCheck and checkBoxUncheck

Изменяет свойство Selected, которое устанавливает и снимает флажок с элемента.

Метод sendEnter()

Эмулирует нажатие клавиши Enter.

```
element.sendEnter();

//the same
element.sendKeys(SapKeyCode.ENTER);
```

Метод sendExit()

Эмулирует нажатие сочетания клавиш Shift+F3.

```
element.sendExit();

//the same
element.sendKeys(SapKeyCode.SHIFT_F3);
```

Сочетания клавиш и команд SAP GUI

SAP GUI команды

Нижеуказанные команды вводятся в поле **Код ОК**, что позволяет выполнять некоторые действия быстрее.

Команда	Описание
`/n`	используется вместе с именем транзакции, переходит к транзакции из существующего окна транзакции
`/o`	отображает список открытых сеансов SAP GUI; все ваши сеансы будут отображаться в диалоговом окне с параметрами для создания и завершения сеансов
`/i`	используется вместе с именем транзакции, закрывает указанную транзакцию

window /nend|выходит из системы с предложением подтвердить, что вы хотите закрыть все сеансы /nex|закрывает все сеансы и выходит из настольного клиента SAP GUI без запроса

Startup параментры

Ярлык	Описание
`-version`	Показать информацию о версии
`-edit`	Редактировать ярлык через диалог
`-register`	Зарегистрируйте класс ярлыков для интеграции в Windows
`-maxgui`	Отобразить окно SAP GUI в развернутом виде

Параметры входа – идентификация пользователя

Ярлыки	Описание
`-user=userid`	Идентификация пользователя системы SAP
`-pw=password`	Пароль для пользователя системы SAP
`-language=EN`	Язык или вход в систему SAP

Параметры входа в систему – идентификация системы

Ярлыки	Описание
`-system=DEV`	SID системы SAP
`-client=032`	Клиент системы SAP для входа в систему
`-sysname="DEV [PUBLIC]"`	Подключение через сервер сообщений (балансировка нагрузки)
`-quiparam="sapserver 10"`	Подключение через единый сервер приложений

Параметры входа в систему – идентификация функции

Ярлыки	Описание
`-command="se38"`	Транзакция или функция, которая должна быть выполнена
`-type=Transaction`	Тип используемой команды (транзакция/отчет/системная команда)
`-title="ABAP/4 Editor"`	Название отображается в диалоговом окне быстрого входа

Драйвер браузера

На этой странице описываются только специальные функции и методы драйвера браузера.

Единая для всех драйверов реализация описана на странице драйвера. Пожалуйста, прочтите это заранее.

Прежде чем углубляться в автоматизацию браузера, рассмотрите задачу, которую вы автоматизируете. Предоставляет ли веб-приложение, с которым вы работаете, доступ к API? API-интерфейсы меняются не так часто, как графические пользовательские интерфейсы. Автоматизация с использованием API более надежна.

Иногда API не предоставляется либо в API отсутствуют некоторые функции для завершения процесса. В этих случаях вам необходимо взаимодействовать с пользовательским интерфейсом приложения; заполнение и отправка форм, нажатие кнопок, щелчок по элементам, очистка контента и многие другие взаимодействия. В этом случае драйвер браузера — это то, что вам нужно.

Драйвер браузера используется для автоматизации действий веб-браузера.

Этот драйвер основан на Selenium WebDriver, который управляет браузером как пользователь. В дополнение к Selenium WebDriver, Канцлер RPA драйвер браузера предоставляет дополнительные полезные методы.

Инициализация драйвера

Вы можете инициализировать драйвер в задаче процесса автоматизации следующим образом:

```
@Driver(value = Type.BROWSER, param = {
    @DriverParameter(key = DriverParams.Browser.SELENIUM_NODE_CAPABILITIES, initializerName = DriverParams.
BrowserCapabilities.CHROME) })
private BrowserDriver browserDriver;
```

Пожалуйста, обратите внимание, что мы передаём имя `DriverParams.BrowserCapabilities.CHROME` в качестве ссылки на инициализатор класса возможностей в приведенном выше примере, поэтому мы ожидаем, что Selenium Grid предоставит нам chrome node. Если мы хотим использовать браузер, отличный от Chrome, нам нужно вместо этого передать соответствующий инициализатор возможностей.

Все поддерживаемые браузеры перечислены в таблице ниже.

Target Browser	Capability Initializer
Chrome	DriverParams.BrowserCapabilities.CHROME
Edge	DriverParams.BrowserCapabilities.EDGE
Firefox	DriverParams.BrowserCapabilities.FIREFOX
Internet Explorer	DriverParams.BrowserCapabilities.IE
Opera	DriverParams.BrowserCapabilities.OPERA
Safari	DriverParams.BrowserCapabilities.SAFARI

Какой браузер следует использовать для автоматизации?

Вы можете выбрать браузер, который вам больше нравится. Мы рекомендуем использовать Google Chrome, если только приложение, которое вы пытаетесь автоматизировать, не работает только с определенным браузером. Разные браузеры могут вести себя по-разному. Робот может работать с Chrome.

Параметры драйвера

Параметр	Тип значения	Значение по умолчанию	Описание
DriverParams.Browser.**SELENIUM_HUB_URL**	URL	`http://localhost:4444/wd/hub`	Selenium Hub URL.
DriverParams.Browser.**SELENIUM_NODE_CAPABILITIES**	`org.openqa.selenium.Capabilities`	null	Используется для установки свойств браузеров для выполнения браузерной автоматизации веб-приложений. Он хранит возможности в виде пар ключ-значение, и эти возможности используются для установки свойств браузера, таких как имя браузера, версия браузера, путь к драйверу браузера в системе и т. д., чтобы определить поведение браузера во время выполнения. Вы можете прочитать о возможностях браузера Chrome по следующей ссылке.
DriverParams.Browser.**PAGE_LOAD_TIMEOUT_SECONDS**	seconds	1800	Ограничивает время, отводимое сценарием для загрузки веб-страницы. Если страница не загружается в течение тайм-аута, скрипт будет остановлен.
DriverParams.Browser.**IMPLICITLY_WAIT_TIMEOUT_SECONDS**	seconds	0	Этот тайм-аут используется для указания количества времени, в течение которого драйвер должен ждать при поиске элемента, если он отсутствует в данный момент.

Ниже приведен пример того, как настроить пользовательские параметры Chrome и изменить параметры драйвера по умолчанию:

```
@Driver(value = DriverParams.Type.BROWSER, param = {
                @DriverParameter(key = DriverParams.Browser.SELENIUM_NODE_CAPABILITIES, initializer =
ChromeInitializer.class),
                @DriverParameter(key = DriverParams.Browser.PAGE_LOAD_TIMEOUT_SECONDS, direct = "20"),
                @DriverParameter(key = DriverParams.Browser.IMPLICITLY_WAIT_TIMEOUT_SECONDS, direct = "5")
})
private BrowserDriver browserDriver;

private static final String filePath = System.getProperty("user.home") + "\\Downloads";

public static final class ChromeInitializer implements Supplier<Capabilities> {

        @Override
        public Capabilities get() {
                ChromeOptions chromeOptions = new ChromeOptions();

                HashMap<String, Object> chromePrefs = new HashMap<>();
                chromePrefs.put("profile.default_content_settings.popups", 0);
                chromePrefs.put("download.default_directory", filePath);

                chromeOptions.setExperimentalOption("prefs", chromePrefs);
                return chromeOptions;
        }
}
```

Поскольку в приведенном выше коде используется Selenium класс ChromeOptions, следует добавить следующую зависимость в свой проект:

```
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-chrome-driver</artifactId>
            <version>${selenium.version}</version>
            <scope>provided</scope>
        </dependency>
```

Открытие новых приложений

Первое, что вы захотите сделать после запуска браузера, это открыть свой сайт. Для этого следует реализовать следующий код:

```
browserDriver.get("https://selenium.dev");
```

Управление окнами

Получение дескриптора окна

WebDriver не делает различий между окнами и вкладками. Если ваш сайт открывает новую вкладку или окно, Selenium позволит вам работать с ним, используя уникальный для каждого окна или вкладки идентификатор, который сохраняется в течение сеанса. Вы можете получить идентификатор, реализуя следующий код:

```
driver.getWindowHandle();
```

Переключение между окнами

Драйвер браузера поддерживает переключение на окно по **имени** или **идентификатору окна**. Для этого используется следующий метод:

```
public void switchToWindow(String nameOrHandle);
```

Также поддерживаются методы, позволяющие дождаться появления окна и только после этого переключиться на него. Методы возвращают «true», если драйвер браузера успешно переключился:

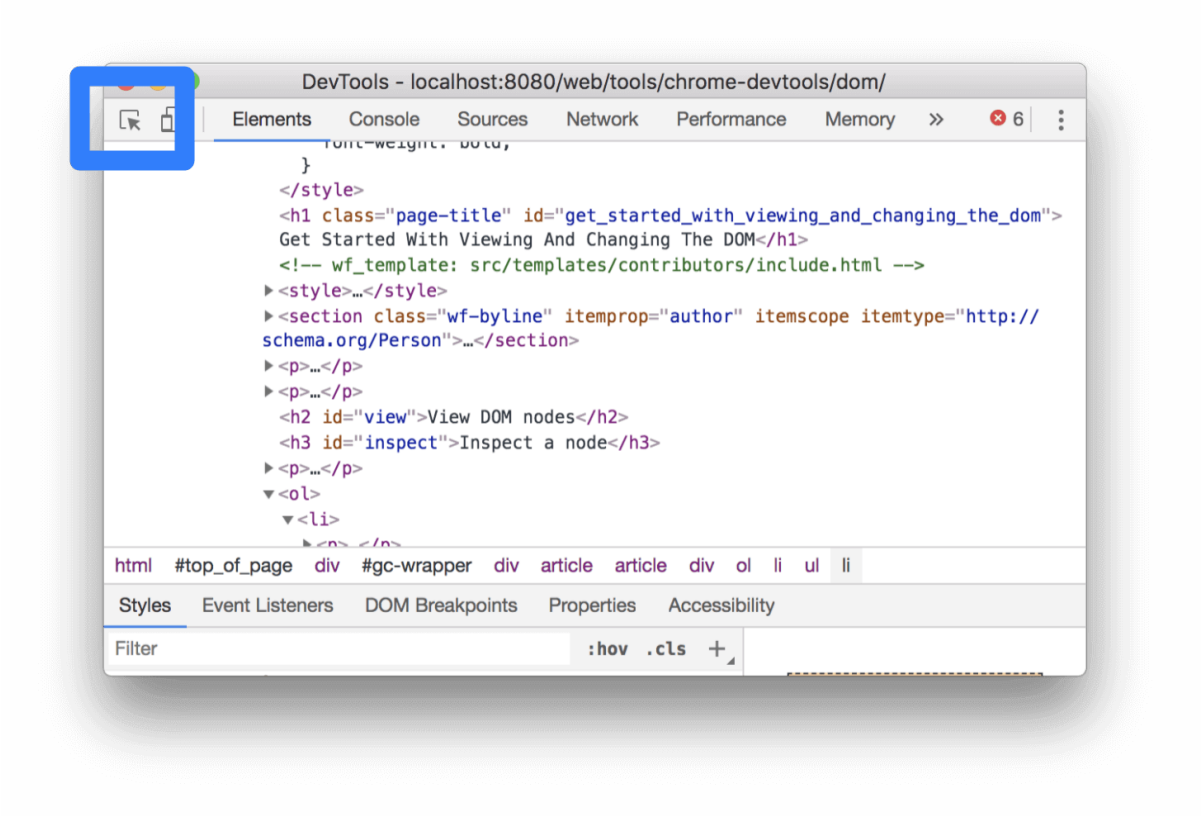
```
public boolean waitAndSwitchToWindow(String windowSearch, int secondsToPoll);
public boolean waitAndSwitchToWindow(String windowSearch, By by, int secondsToPoll);
public boolean waitAndSwitchToWindow(String windowSearch, By by, int secondsToPoll, boolean
suppressTimeoutException)
```

Работа с элементами интерфейса

Инспектор

Элементы интерфейса представляют элементы модели DOM. API драйвера браузера предоставляет встроенные методы для поиска элементов, основанных на различных свойствах, таких как идентификатор, имя, класс, XPath, селекторы CSS, текст ссылки и т. д. Чтобы создать селектор, вы можете использовать встроенный инспектор DOM браузера.

Например, вы можете прочитать об инспекторе браузера Chrome по следующей ссылке.



Поиск элементов

Ниже приведена таблица с локаторами, которые поддерживает BrowserDriver:

Тип селектора
BrowserSearch.id
BrowserSearch.name
BrowserSearch.xpath
BrowserSearch.cssSelector
BrowserSearch.className
BrowserSearch.tagName
BrowserSearch.linkText
BrowserSearch.partialLinkText

Ниже приведен пример того, как найти элемент по имени:

```
BrowserElement webElement = browserDriver.findElement(BrowserSearch.name("q"));
```

Вы можете прочитать больше информации о том, как работать с селекторами Selenium:

XPath Tutorial

CSS selectors

Locators Tutorial

Selenium documentation

Методы драйвера браузера для работы с окнами

Разрешение экрана может повлиять на отображение вашего веб-приложения, поэтому WebDriver предоставляет механизмы для перемещения и изменения размера окна браузера.

Текущий объект окна можно получить из драйвера с помощью метода:

```
Window window = driver.manage().window();
```

Получить размер окна

Получает размер окна браузера в пикселях.

```
//Access each dimension individually
int width = driver.manage().window().getSize().getWidth();
int height = driver.manage().window().getSize().getHeight();

//Or store the dimensions and query them later
Dimension size = driver.manage().window().getSize();
int width1 = size.getWidth();
int height1 = size.getHeight();
```

Установить размер окна

Восстанавливает окно и устанавливает размер окна.

```
driver.manage().window().setSize(new Dimension(1024, 768));
```

Получить положение окна

Получает координаты верхней левой координаты окна браузера.

```
// Access each dimension individually
int x = driver.manage().window().getPosition().getX();
int y = driver.manage().window().getPosition().getY();

// Or store the dimensions and query them later
Point position = driver.manage().window().getPosition();
int x1 = position.getX();
int y1 = position.getY();
```

Установить положение окна

Перемещает окно в выбранное положение.

```
// Move the window to the top left of the primary monitor
driver.manage().window().setPosition(new Point(0, 0));
```

Развернуть окно

Увеличивает окно. Для большинства операционных систем окно будет заполнять экран, не блокируя собственные меню и панели инструментов операционной системы.

```
driver.manage().window().maximize();
```

Свернуть окно

Сворачивает окно текущего контекста просмотра. Точное поведение этой команды специфично для определенных оконных менеджеров.

Обычно окно скрывается на панели задач.

```
driver.manage().window().minimize();
```

Полноэкранное окно

Заполняет весь экран, аналогично нажатию F11 в большинстве браузеров.

```
driver.manage().window().fullscreen();
```

Методы драйвера браузера для работы с элементами интерфейса

BrowserElement представляет элемент DOM. BrowserElements можно найти путем поиска в корне документа с помощью экземпляра BrowserDriver или путем поиска в другом BrowserElement. BrowserElement аналогичен Selenium WebElement, поэтому вы можете прочитать больше документации о нем на Selenium project documentation.

Получить активный элемент

Он используется для отслеживания (или) поиска элемента DOM, который имеет фокус в текущем контексте просмотра.

```
// Get attribute of current active element
String attr = driver.switchTo().activeElement().getAttribute("title");
```

Клик

Кликните по этому элементу

```
driver.findElement(BrowserSearch.name("btnK")).click();
```

Активность элемента

Этот метод используется для проверки того, включен или отключен упомянутый элемент на веб-странице. Возвращает логическое значение true, если указанный элемент включен в текущем контексте просмотра, в противном случае возвращает false.

```
//returns true if element is enabled else returns false
boolean value = driver.findElement(BrowserSearch.name("btnK")).isEnabled();
```

Выбранный элемент

Этот метод определяет, выбран ли упомянутый элемент или нет. Этот метод широко используется для флажков, переключателей, элементов ввода и опций. Возвращает логическое значение true, если указанный элемент выбран в текущем контексте просмотра, в противном случае возвращает false.

```
//returns true if element is checked else returns false
boolean value = driver.findElement(BrowserSearch.cssSelector("input[type='checkbox']:first-of-type")).
isSelected();
```

Получить имя тега элемента

Используется для получения имени тега выбранного элемента.

```
//returns TagName of the element
String value = driver.findElement(BrowserSearch.name("title")).getTagName();
```

Получить область занимаемую элементом

Используется для получения размеров и координат выбранного элемента.

Извлеченные данные содержат следующие детали:

- Положение по оси X от верхнего левого угла элемента
- Положение по оси Y от верхнего левого угла элемента
- Высота элемента
- Ширина элемента

```
// Returns height, width, x and y coordinates referenced element
Rectangle res = driver.findElement(BrowserSearch.cssSelector("h1")).getRect();

// Rectangle class provides getX,getY, getWidth, getHeight methods
System.out.println(res.getX());
```

Получить значение CSS элемента

Извлекает значение стиля элемента в текущем контексте просмотра.

```
// Retrieves the computed style property 'color' of linktext
String cssValue = driver.findElement(BrowserSearch.linkText("More information...")).getCssValue("color");
```

Получить текст элемента

Извлекает отображаемый текст указанного элемента.

```
// Retrieves the text of the element
String text = driver.findElement(BrowserSearch.cssSelector("h1")).getText();
```

Драйвер рабочего стола

На этой странице описываются только специальные функции и методы Desktop Driver, которые могут отличаться в других драйверах.

Единая для всех драйверов реализация описана на Странице драйверов. Пожалуйста, прочитайте это заранее

Несмотря на постоянно растущую популярность веб-приложений на рабочем месте, во многих бизнес-процессах по-прежнему задействованы настольные приложения по соображениям совместимости, безопасности или в соответствии с техническими требованиями. Возможность программного управления этими типами приложений открывает целый мир возможностей автоматизации.

DesktopDriver используется для автоматизации настольных приложений и позволяет выполнять задачи, действуя как человек-оператор, напрямую управляя интерфейсом рабочего стола. Сюда входят такие операции, как открытие и закрытие приложений, имитация движений и щелчков мыши, срабатывание клавиш клавиатуры и сочетаний клавиш.

По сравнению с автоматизацией браузера, автоматизация настольных приложений — более разнообразная и сложная область. Однако основная идея остается неизменной для всех операционных систем и методов доступа: вам нужен способ указать на определенные элементы настольного приложения, и взаимодействовать с ними, управляя нажатием на кнопку, вводом текста, перемещением и так далее.

Этот драйвер основан на проекте Mmarquee UIAutomation project который представляет собой оболочку Microsoft UIAutomation Library на основе JAVA. Эта платформа предназначена для автоматизации многофункциональных клиентских приложений на основе Win32 (включая Delphi), WPF и других приложений Windows (включая Java SWT). Он использует библиотеку JNA для вызова библиотеки автоматизации WIndows на основе COM.

Инициализация драйвера

Вы можете инициализировать драйвер в процессе автоматизации следующим образом:

```
@Driver(DriverParams.Type.DESKTOP)
private DesktopDriver desktopDriver;
```

Параметры драйвера

Параметр драйвера	Единица измерения	Значение по умолчанию	Описание
DriverParams.Desktop.**LAUNCH_APPLICATION_TIMEOUT**	ms	5000	Параметр отвечает за таймаут, который должен ждать драйвер после вызова команды на запуск приложения и до момента запуска приложения.

```
@Driver(value = DriverParams.Type.DESKTOP, param = {
        @DriverParameter(key = DriverParams.Desktop.LAUNCH_APPLICATION_TIMEOUT, direct = "50000")
})
private DesktopDriver desktopDriver;
```

Открытие новых приложений

Desktop Driver поддерживает 2 метода запуска нового приложения: указание пути или пути с аргументами.

```
public void get(String path)
public void get(String... command)
```

Ниже приведен пример того, как запустить приложение, используя путь:

```
driver.get("C:\\Windows\\system32\\calc.exe");

//or

driver.get("notepad.exe");
```

Управление окнами

Desktop Driver поддерживает переключение на окно, используя **имя класса окна, заголовок или регулярное выражение заголовка, идентификатор окна.**

Для ипользуются следующие методы:

```
public void switchToWindow(String titleOrHandle)
public void switchToWindow(Pattern titleRegexp)
public void switchToWindow(String className, String titleOrHandle)
public void switchToWindow(String className, Pattern titlePattern)
```

Также поддерживаются методы, позволяющие дождаться появления окна и только после этого переключиться на него. Методы возвращают «true», если драйвер успешно переключился:

```
/**
 * Switches to the desktop 'window' associated with the class name and title regexp (or window handle).
 *
 * @param className Class name to search for.
 * @param titleRegexp Regexp for title to search for or window handle.
 * @param titleOrHandle title to search for or window handle.
 * @param secondsToPoll Seconds to wait for windows to be appeared.
 * @param suppressTimeoutException Set true to don't let method throw an exception, so it returns "false" value
instead.
 *
 * @return True if driver switched successfully
 *
*/
public boolean waitAndSwitchToWindow(String titleOrHandle, int secondsToPoll)
public boolean waitAndSwitchToWindow(Pattern titleRegexp, int secondsToPoll)
public boolean waitAndSwitchToWindow(String titleOrHandle, int secondsToPoll, boolean suppressTimeoutException)
public boolean waitAndSwitchToWindow(Pattern titleRegexp, int secondsToPoll, boolean suppressTimeoutException)
public boolean waitAndSwitchToWindow(String className, String titleOrHandle, int secondsToPoll)
public boolean waitAndSwitchToWindow(String className, Pattern titleRegexp, int secondsToPoll)
public boolean waitAndSwitchToWindow(String className, String titleOrHandle, int secondsToPoll, boolean
suppressTimeoutException)
public boolean waitAndSwitchToWindow(String className, Pattern titlePatternOrHandle, int secondsToPoll, boolean
suppressTimeoutException)
```

Пример:

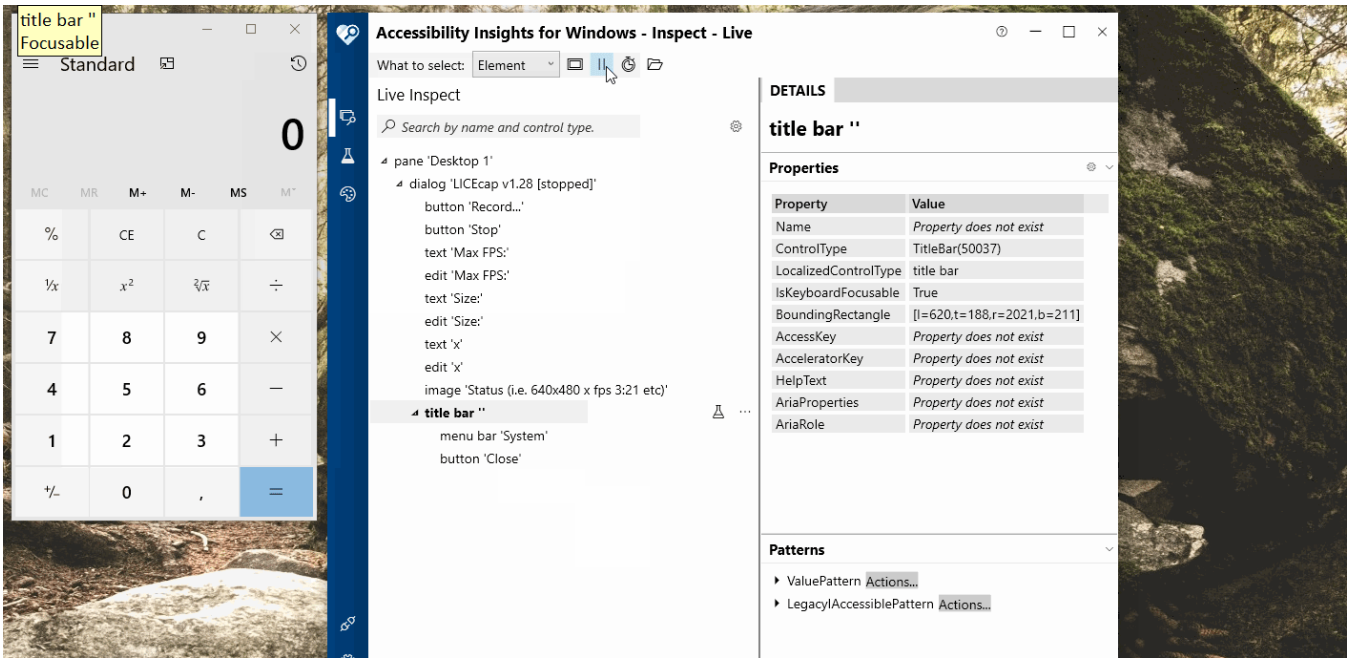
```
driver.switchToWindow("Untitled - Notepad");
```

Работа с элементами интерфейса

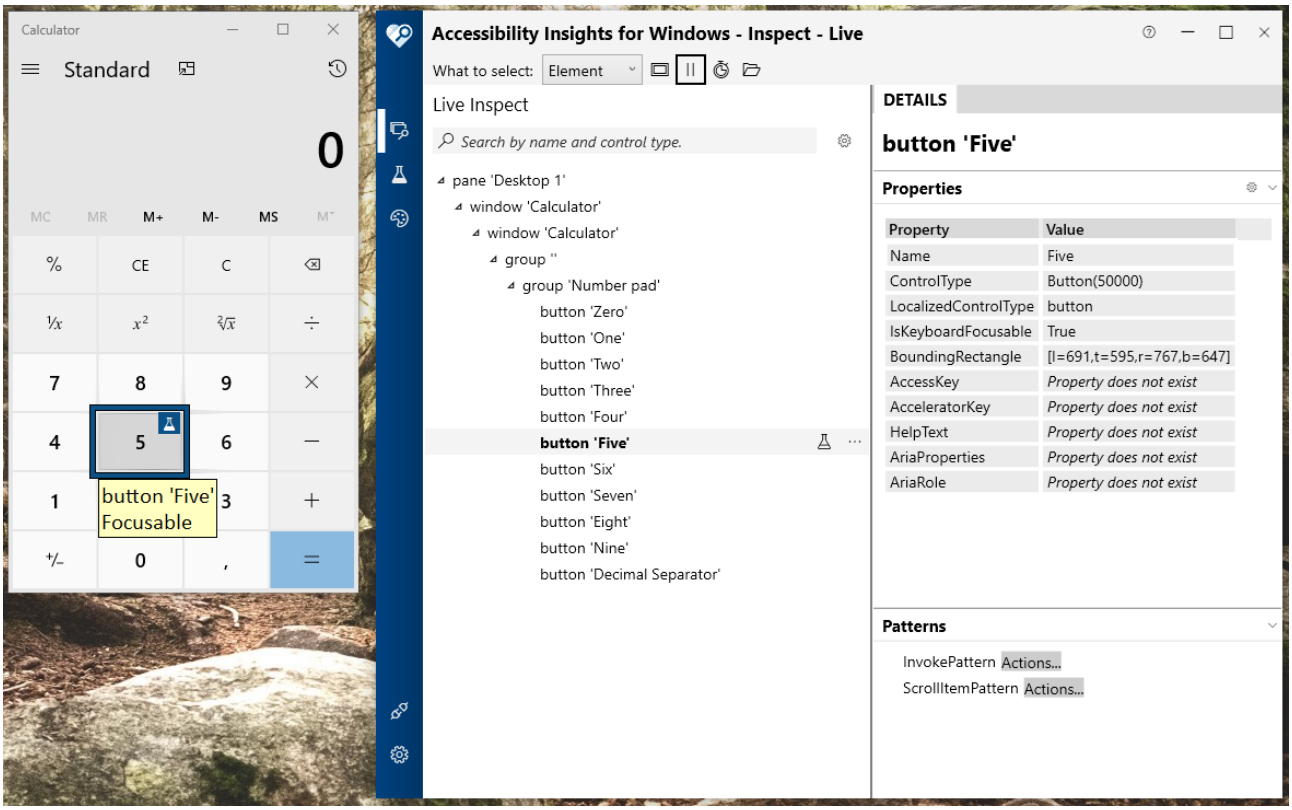
Инспектор

Один из способов автоматизации настольных приложений – поиск элементов интерфейса по их локаторам. Microsoft рекомендует Accessibility Insights для просмотра свойств элементов интерфейса. Также можно использовать устаревшие инструменты, такие как Inspect.exe. Этот инструмент позволяет получить информацию об элементах интерфейса, таких как Имя, ID, Тип и т.д.

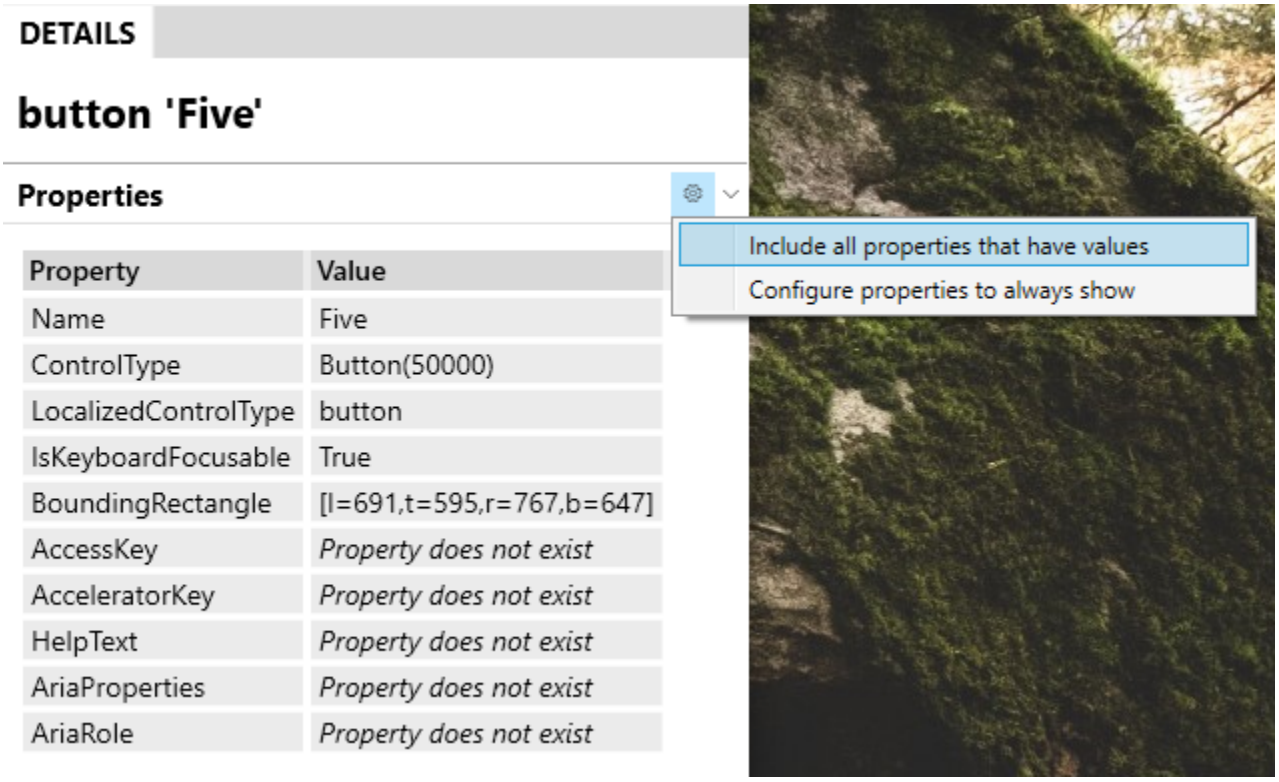
Ниже демонстрируется работа с Accessibility Insights for Windows на примере Windows приложения Калькулятор. При наведении курсора на кнопку калькулятора, свойства этой кнопки отображаются в Accessibility Insights for Windows.



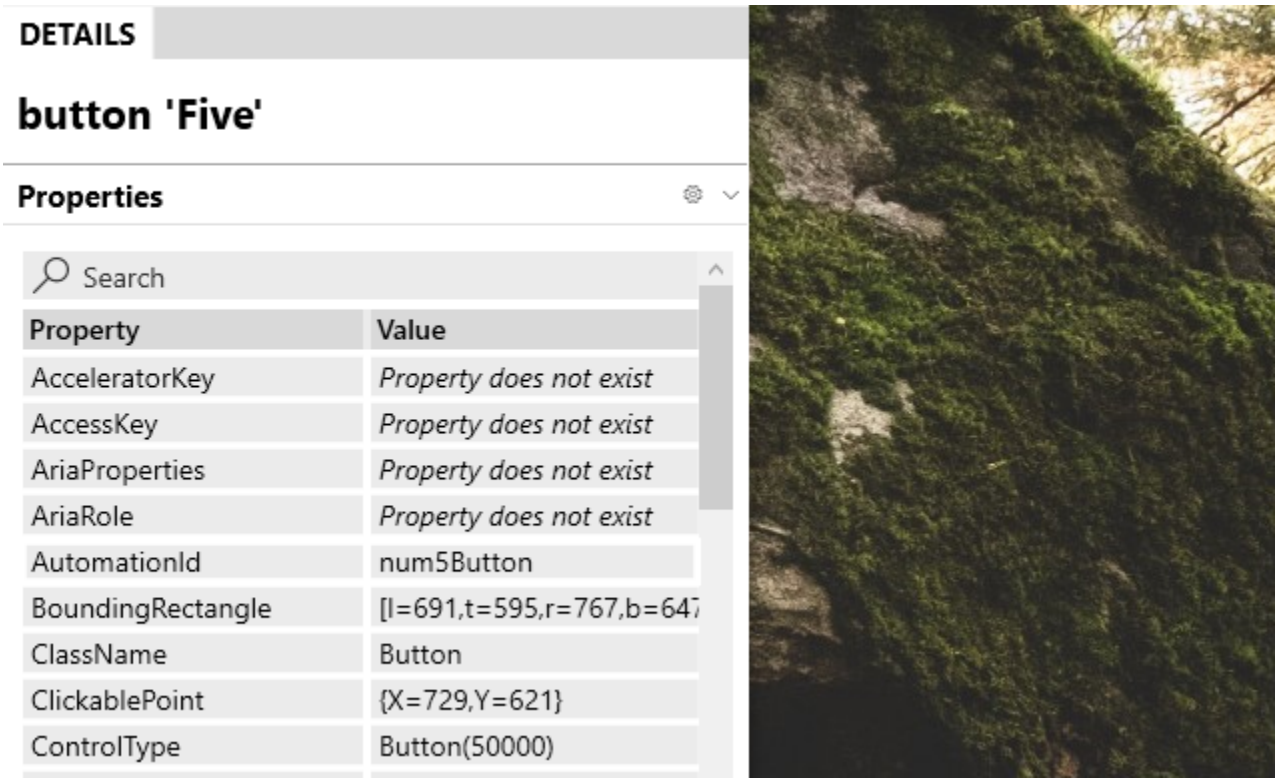
Следует учитывать, что использование для автоматизации некоторых свойств, например Name, — не самый надежный вариант, поскольку значение этого свойства может меняться в зависимости от языковых настроек Windows.



Чтобы увидеть больше свойств, нажмите на значок настроек и выберите `Include all properties that have values`:



Это позволит получить значение свойства `AutomationId`. В данном случае значение этого свойства равно `num5Button`:



Поиск элементов интерфейса

> До переключения на окно конкретного приложения с помощью драйвера, окно рабочего стола (корневое) считается активным и для всех других окон на экране, поиск будет выполняться медленнее.
>
> Поэтому перед тем, как вызвать метод "findElement(s)", **настоятельно рекомендуется переключиться на целевое окно,** чт обы корневое окно не было активным.

Ниже приведена таблица с локаторами, которые поддерживает DesktopDriver:

Тип локатора
DesktopSearch.query

Desktop Driver поддерживает только локатор элементов "**DesktopSearch.desktopSearch**". Этот локатор принимает специальный объект **UIQuery**, который вы можете создать, используя атрибуты элемента.

Ниже приведен пример того, как найти элемент по атрибутам Name, className и controlType (вся информация может быть извлечена с помощью Inspector.exe):

```
DesktopElement desktopElement = driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder()
        .name("Text Editor")
        .className("Edit")
        .controlType(ControlType.Edit.getValue())
        .build()
));
```

Методы драйвера рабочего стола для работы с окнами

Для получить доступа к управлению окном используется метод window():

```
Window window = driver.manage().window();
```

Методы управления окном:

Метод maximize()

Разворачивает текущее окно до максимального размера.

```
driver.manage().window().maximize();
```

Метод focus()

Вызов этого метода фокусируется на текущем окне.

```
((DesktopWindow) driver.manage().window()).focus();
```

Метод getSize()

Возвращает объект Dimension со свойствами «ширина» и «высота».

```
Dimension windowDimension = driver.manage().window().getSize();

System.out.println("Current window size is: " + windowDimension.getWidth() + "x" + windowDimension.getHeight());
```

Метод getPosition()

Возвращает текущую позицию окна на экране. Значение возвращается как объект «Point» со свойствами «x» и «y».

```
Point windowPosition = driver.manage().window().getPosition();

System.out.println("Current window position is: " + windowPosition.getX() + "x" + windowPosition.getY());
```

Методы драйвера рабочего стола для работы с элементами интерфейса

DesktopElement - это элемент интерфейса рабочего стола. Для взаимодействия с DesktopElement используются методы Desktop Driver. **Получение DesktopElement путем поиска в другом DesktopElement пока не поддерживается.**

DesktopDriver API предоставляет встроенные методы для поиска DesktopElement, основанные на различных свойствах, таких как идентификатор, имя, текст и т. д.

Методы работы с элементами интерфейса:

Метод click()

Метод имитирует щелчек мышкой по элементу.

```
driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().className("Button")).click();
```

Метод isEnable()

This method is used to check if the referenced Element is enabled or disabled on a screen. Returns a boolean value, **True** if the referenced element is **enabled** in the current browsing context else returns **false**.

Этот метод используется для проверки того, включен или отключен элемент на экране. Возвращает логическое значение, **True**, если указанный элемент **включен**, в противном случае возвращает **false**.

```
//returns true if element is enabled else returns false
boolean value = driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().className("Edit")).
isEnabled();
```

Метод isSelected()

Этот метод определяет, выбран ли упомянутый элемент или нет. Этот метод часто используется для переключателей, флажков или опций в меню.

Возвращает логическое значение, **True**, если указанный элемент **выбран**, в противном случае возвращает **false**.

```
//returns true if element is checked else returns false
boolean value = driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().className("Checkbox")).
isSelected();
```

Метод getTagName()

Метод используется для получения значения тега веб-элемента. **В контексте Desktop Driver означает значение атрибута ControlType.**

```
//returns ControlType of the element
String value = driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().name("Text Editor")).
getTagName();
```

Метод getRect()

Он используется для получения размеров и координат элемента, на который указывает ссылка.

Метод возвращает следующие данные:

1.Положение по оси X от верхнего левого угла элемента
2.положение по оси Y от верхнего левого угла элемента
3.Высота элемента
4.Ширина элемента

```
// Returns height, width, x and y coordinates referenced element
Rectangle res =  driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().name("Text Editor")).
getRect();

// Rectangle class provides getX,getY, getWidth, getHeight methods
System.out.println(res.getX());
```

Метод getText()

Возвращает видимый текст элемента.

```
// Retrieves the text of the element
String text = driver.findElement(DesktopSearch.query(DesktopSearch.UIQuery.builder().className("Edit")).
getText();
```

Драйвер экрана

На этой странице описываются только специальные функции и методы драйвера экрана, которые могут отличаться в других драйверах.

Единая для всех драйверов реализация описана на странице драйверов. Пожалуйста, прочитайте это заранее.

ScreenDriver использует автоматизацию рабочего стола на основе шаблонов изображений. В общем, локаторы на основе изображений довольно хрупкие, и их следует использовать в крайнем случае.

В большинстве десктопных приложений многое можно сделать с помощью сочетаний клавиш, а поскольку мы можем управлять клавиатурой, мы можем использовать их напрямую. Проверьте документацию по приложению, которое вы автоматизируете, чтобы узнать, что доступно.

При автоматизации рабочего стола на основе шаблонов изображений вы предоставляете роботу скриншоты частей интерфейса, с которыми ему необходимо взаимодействовать, например кнопки или поля ввода. Изображения сохраняются вместе с вашим кодом автоматизации. Робот сравнит изображение с тем, что в данный момент отображается на экране, и найдет свою цель.

Используя тот же метод, вы также можете найти определенную часть интерфейса на экране, а затем добавить смещение в пикселях, указав роботу, например, «щелкнуть 200 пикселей справа» от изображения, которое вы предоставляете.

Этот метод позволяет автоматизировать такие среды, как Citrix и другие удаленные терминалы, где у вас нет доступа к самой целевой машине, а только к «видеопотоку» рабочего стола.

ScreenDriver используется для автоматизации чего-либо на экране настольного компьютера под управлением Windows, Mac или некоторых Linux/Unix.. Этот драйвер основан на проекте **sikuliX** .

Класс ScreenDriver основан на классе Screen — центральном объекте библиотеки sikuli. Согласно документации, класс Screen предназначен для представления физического монитора, на котором реализован процесс захвата (снятие прямоугольника со снимка экрана, который будет использоваться для дальнейшей обработки с операциями поиска ,разработки, но если вам нужны какие-то специфические методы из класса Screen, вы можете обратиться непосредственно к нему. Ниже приведены некоторые примеры инициализации и использования драйвера.

Распространенные проблемы автоматизации рабочего стола на основе локатора изображений автоматизация операций на рабочем столе

При использовании этого подхода вы должны знать о некоторых проблемах:

- **Системные настройки могут повлиять на распознавание изображений:** Внешний вид элементов интерфейса на экране зависит от системных настроек, таких как цветовые схемы, прозрачность и системные шрифты. Изображения, сделанные в системе, могут в конечном итоге выглядеть иначе, чем целевая система, и робот может не распознать их, остановив процесс.
- **Разрешение экрана как фактор:** Другое разрешение экрана может привести к тому, что элементы на экране будут перемещаться или изменяться в размере.
- **Разные версии одной и той же операционной системы могут отличаться визуально:** Операционные системы предоставляют общие рекомендации по отображению элементов интерфейса на экране. Если операционная система обновлена, шаблоны изображений могут перестать распознаваться.

Чтобы смягчить этот тип проблем и сделать вашу автоматизацию более надежной, мы рекомендуем:

- придерживаться настроек по умолчанию для шрифтов и цветов
- использование специальных возможностей для уменьшения визуальных эффектов, таких как тени и прозрачность
- если возможно, используя целевую машину для получения изображений локатора, чтобы убедиться, что все настройки одинаковы.

Инициализация драйвера

Вы можете инициализировать драйвер в процессе автоматизации следующим образом:

```
@Driver(DriverParams.Type.SCREEN)
private ScreenDriver screenDriver;
```

Параметры драйвера

Имя параметра	Тип	Значение по умолчанию	Описание
DriverParams.Screen.**MIN_SIMILARITY**	float	0.7	Минимальное сходство операций поиска по умолчанию. Если при использовании операции «findElement» предоставляется только файл изображения, драйвер экрана ищет область, используя минимальное сходство по умолчанию, равное 0,7.

```
@Driver(value = DriverParams.Type.SCREEN, param = {
        @DriverParameter(key = DriverParams.Screen.MIN_SIMILARITY, direct = "0.8")
})
private SapDriver sapDriver;
```

Открытие новых приложений

Используйте DesktopDriver, если вам нужно запустить приложение с помощью драйвера. Ссылка как открыть новое приложение с помощью десктопнго Driver

Управление окнами

В ScreenDriver нет такой сущности, как «Окно», поскольку этот драйвер работает со всем экраном как с одним окном, и драйвер не может различать все разные окна на экране. Если вам нужно выполнить какую-либо операцию, связанную с окном (закрытие, развертывание и т. д.), вы можете использовать горячие клавиши.

Работа с элементами инткрфейса

Подготовка скриншотов

Скриншоты должны быть сделаны на экране, где процесс должен работать. Различия в разрешении экрана или цветовой схеме могут нарушить процесс.

1. Сделайте скриншот нужного элемента и сохраните его в файл.

   Мы рекомендуем использовать утилиту, которая позволяет легко делать скриншоты. lightshot, например.

2. Поместите этот файл в ресурсы проекта.
3. Вы можете использовать имя файла изображения (или путь к файлу, если его нет в ресурсах) в качестве селектора.

Поиск элементов

Ниже приведена таблица с локаторами, которые поддерживает ScreenDriver:

Тип селектора
ScreenSearch.image
ScreenSearch.anchor

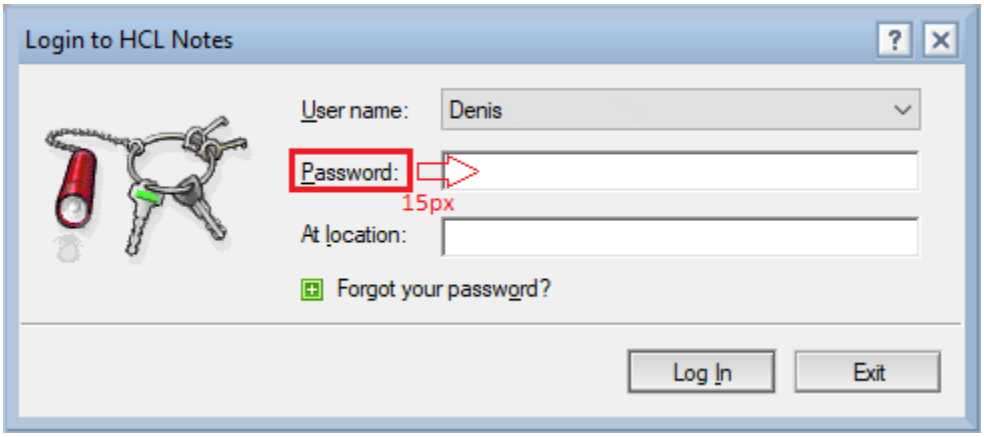
ScreenDriver поддерживает два типа селекторов: "**ScreenSearch.image**" и "**ScreenSearch.anchor**".

Ниже приведен пример того, как найти элемент, используя "**ScreenSearch.image**":

```
ScreenElement imageElement = driver.findElement(ScreenSearch.image("navigation-btn-1.png"));
```

ScreenDriver также поддерживает селектор **«ScreenSearch.anchor»**. Это полезно, когда вы не можете указать уникальную область UI вашего целевого элемента (например, потому что он содержит динамический текст или это пустое поле ввода), но вы можете указать некоторую область пользовательского интерфейса рядом с вашей целевой областью.

Рассмотрим следующий пример, когда вам нужно найти расположение поля ввода «Пароль»:



Вы не можете использовать область поля ввода в качестве селектора, потому что есть 2 одинаковых пустых ввода: «Password» и «At location».

Таким образом, вы можете использовать область с меткой «Password:» (она выделена красным) в качестве **привязки** для поля ввода пароля. Поскольку вы знаете, что ввод следует за меткой после 15px, ваш селектор привязки будет выглядеть так:

```
ScreenElement passwordLabelAnchor = driver.findElement(ScreenSearch.image("password-label.png"));
int passwordFieldWidth = 250;
int passwordFieldHeight = 20;

ScreenElement passwordInputField = driver.findElement(ScreenSearch.anchor(passwordLabelAnchor, 15, 0,
passwordFieldHeight, passwordFieldWidth));
```

Методы драйвера экрана для работы с элементами интерфейса

Элемент экрана (ScreenElement) это некая прямоугольная область экрана. Искать и получить этот элемент можно только по изображению (см. Найдите элементы секции на странице драйвера):

Методы работы с элементами интерфейса:

Метод contains()

Принимает другой объект «ScreenElement» и возвращает «true» текущего элемента, содержащего внутри другой элемент изображения.

```
ScreenElement mainWindow = driver.findElement(ScreenSearch.image("main-window-title.png"));
ScreenElement okButton = driver.findElement(ScreenSearch.image("ok-button.png"));
boolean mainWindowHasOkButton =  mainWindow.contains(okButton);
```

Метод copyToClipboard()

Выбирает внутри целевого элемента изображения, нажимает комбинацию клавиш «Ctrl+A» (выбрать все), а затем комбинацию клавиш «Ctrl+C» (копировать), чтобы сохранить выделенный текст в буфер.

```
driver.findElement(ScreenSearch.image("first-name-input-area.png")).copyToClipboard();
```

Метод doubleClick()

Выполняет двойной щелчок по центру целевого элемента изображения.

```
driver.findElement(ScreenSearch.image("edit-area.png")).doubleClick();
```

Метод getOcrText()

Модуль OCR распознает и возвращает текст из целевого элемента изображения.

```
String text = driver.findElement(ScreenSearch.image("user-input-field.png")).getOcrText();
```

Метод getRegion()

Возвращает `org.sikuli.script.Region` объект, который содержит множество полезных методов.
Например. методы региона `getX()`, `getY()`, `getW()` и `getH()` определяют положение на экране и размер элемента.

```
Region region = driver.findElement(ScreenSearch.image("user-input-field.png")).getRegion();

System.out.println("x: " +  region.getX());
System.out.println("y: " + region.getY());
System.out.println("width: " + region.getW());
System.out.println("height: " + region.getH());
```

Метод getUrl()

Возвращает относительный путь к искомому изображению.

```
String userInputFieldPngUrl = driver.findElement(ScreenSearch.image("user-input-field.png")).getUrl();
```

Метод highlight()

Выделяет область за некоторый период времени. Принимает время в секундах как double.

```
driver.findElement(ScreenSearch.image("user-input-field.png")).highlight(10);
```

Метод rightClick()

Выполняет правый щелчок в центре целевого элемента изображения.

```
driver.findElement(ScreenSearch.image("user-input-field.png")).rightClick();
```

Метод keyDown()

Нажимает и держит ключ. Принимает ключевой код от `org.sikuli.script.Key`

```
ScreenElement mainWindowElement = driver.findElement(ScreenSearch.image("main-window.png"));

mainWindowElement.keyDown(Key.ALT);
mainWindowElement.keyDown(Key.F4);
mainWindowElement.keyUp();
```

Метод keyUp()

Отпускает все нажатые клавиши. Также может принимать константу кода клавиши из `org.sikuli.script.Key`.

```
ScreenElement mainWindowElement = driver.findElement(ScreenSearch.image("main-window.png"));

mainWindowElement.keyDown(Key.ALT);
mainWindowElement.keyDown(Key.F4);
mainWindowElement.keyUp();
```

Метод pasteFromClipboard()

Щелкает внутри целевого элемента изображения и нажимает комбинацию клавиш «Ctrl+V» (вставить).

```
driver.findElement(ScreenSearch.image("first-name-input-area.png")).pasteFromClipboard();
```

Метод sendEnter()

Отправляет клавишу ввода внутри целевого элемента изображения.

```
driver.findElement(ScreenSearch.image("first-name-input-area.png")).sendEnter();
```

Службы передачи данных

В этом разделе описываются службы конфигурации и хранения данных, предоставляемые платформой.

- Служба настроек
- Служба хранилища данных
- Служба хранилища файлов
- Служба уведомлений
- Служба хранилища секретов

Служба настроек

Обзор

ConfigurationService предоставляет доступ на чтение к свойствам ключ-значение (например, URL-адрес ресурса, псевдоним хранилища, имя хранилища данных и т.д.), которые могут использоваться при выполнении автоматизированных процессов и могут быть изменены в любое время без повторного развертывания автоматизированного процесса.

Определение свойств в standalone и developer настройках

Для этих настроек свойства могут храниться в файле ***resources/apm_run.properties*** в виде пар ключ-значение

Пример ***apm_run.properties***

apm_run.properties

```
hello=Hello property
boolean.value=true
url.value=https://172.20.194.57:8444
number.value=123
long.value=9223372036854775807
```

Определение свойств для запуска сервера управления

Вы можете определить настройки в следующих местах:

- **Настройки сервера управления** - доступны для всех процессов автоматизации загруженных на сервер управления. См. На стройки сервера.
- **Конфигурационные параметры автоматизированного процесса** - видны только для запусков автоматизированного процесса и переопределяют свойства, определенные на уровне сервера управления. См. Конфигурационные параметры автоматизированного процесса.
- **Конфигурационные параметры узла** - применяются для запусков, выполняемых на узле, и переопределяют свойства, определенные на уровнях автоматизированного процесса и сервера управления. См. Конфигурационные параметры системного узла.

Свойства, связанные с функциями на узле, будут использованы только в том случае, если на уровне выше нет определенных свойств. Эти свойства связаны с конечными точками драйвера, такими как SELENIUM_HUB_URL, WINDOWS_APPLICATION_DRIVER_URL. См. Функции узла.

Когда вы развертываете процесс автоматизации на узле, вы должны определить все необходимые свойства в настройках сервера управления, resources/apm_run.properties недоступен во время выполнения на узле.

Свойства конфигурации узла

Есть несколько параметров, указывающих, для какого узла вы можете использовать JVM для автоматизированного процесса:

Параметр	Описание	Пример

JAVA_HOME	Измените JVM, который будет использоваться для запуска автоматизированных процессов.	`c:\Program Files\Java\jdk-11.0.8`

JAVA_OPTS	Параметры JVM, которые будут переданы для запуска процесса автоматизации.
	
	Используя его, вы можете запустить процесс автоматизации в режиме отладки.
		`-DmyProperty1=123 -DmyProperty2=123`
		
		`-agentlib:jdwp=transport=dt_socket,server=y, suspend=n,address=5005`

Configuration Service API (API службы конфигурации)

Во-первых, нужно ввести **ConfigurationService** в ваш класс задачи:

```
@ApTaskEntry(name = "ConfigurationService Example Task")
@Slf4j
public class ConfigurationExample extends ApTask {
    @Inject
    private ConfigurationService cfg;
}
```

Извлечь строку с помощью **ConfigurationService**:

```
String testProperty = this.cfg.get("hello");
```

Можно установить значение по умолчанию, если ключ не существует:

```
String property2 = this.cfg.get("no.value", "Default value");
```

Также можно использовать предопределенные параметры форматирования для форматирования значений:

```
Boolean booleanFromCfg = this.cfg.get("boolean", ConfigurationService.Formatter.BOOLEAN);
Integer number = this.cfg.get("number", ConfigurationService.Formatter.INT);
Long longNumber = this.cfg.get("long.number", ConfigurationService.Formatter.LONG);
URL url = this.cfg.get("url", ConfigurationService.Formatter.URL);
```

Полный пример:

ConfigurationExample.java

```
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.service.ConfigurationService;
import lombok.extern.slf4j.Slf4j;

import javax.inject.Inject;
import java.net.URL;

@ApTaskEntry(name = "Configuration Service Example")
@Slf4j
public class ConfigurationExample extends ApTask {

    @Inject
    private ConfigurationService cfg;

    @Override
    public void execute() {
        String testProperty = this.cfg.get("hello");
        log.info("hello = {}", testProperty);

        String property2 = this.cfg.get("no.value", "Default value");
        log.info("no.value = {}", property2);

        Boolean booleanFromCfg = this.cfg.get("boolean.value", ConfigurationService.Formatter.BOOLEAN);
        Integer number = this.cfg.get("number.value", ConfigurationService.Formatter.INT);
        Long longNumber = this.cfg.get("long.value", ConfigurationService.Formatter.LONG);
        URL url = this.cfg.get("url.value", ConfigurationService.Formatter.URL);
        log.info("boolean.value = {}", booleanFromCfg);
        log.info("number.value = {}", number);
        log.info("long.value = {}", longNumber);
        log.info("url.value = {}", url);
    }
}
```

ConfigurationExampleAp.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import inarchetype.tasks.ConfigurationExample;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@ApModuleEntry(name = "Configuration Service Example")
public class ConfigurationExampleAp extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), ConfigurationExample.class).get();
    }
}
```

ModuleLocalRun

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;
import com.iba.kanclerrpa.engine.boot.configuration.StandaloneConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(ConfigurationExampleAp.class, new DevelopmentConfigurationModule());
        //ApModuleRunner.localLaunch(ConfigurationExampleAp.class);
    }
}
```

Служба хранилища данных

Что такое Хранилище данных?

Хранилище данных - это хранилище, которое позволяет управлять данными. Хранилище данных предоставляет полезный интерфейс для работы с вашими автоматизированными процессами.

Большинство процессов сосредоточены вокруг определенной сущности. На разных этапах эта сущность дополняется данными, проверяется, вводится в разные системы и т.д.
Например, при обработке платежей вам может потребоваться извлечь данные о плательщике из какой-либо системы, проверить эти данные, ввести платежные данные в базу данных клиентов и отправить отчет. Для такого случая удобно создать объект платежа в коде и сохранять/обновлять его поля и статусы в хранилище данных на каждом этапе обработки. Таким образом, архитектура процесса становится простой и понятной, риск потери данных сводится к минимуму, а процесс становится легко масштабируемым и изменяемым.

В этой статье мы рассмотрим взаимодействие хранилища данных из кода автоматизированного процесса, а именно сохранение и получение java-объектов.

API хранилища данных Канцлер RPA

Благодаря тому, что API хранилища данных Канцлер RPA соответствует Java persistence API, разработчики Java, знакомые с Java persistence API, найдут его простым в использовании.

Entity

Термин **Entity** используется для описания постоянных объектов. Аннотация @Entity указывает, что класс и его объекты должны сохраняться. Она описывает сущность в хранилищах данных. Эта аннотация должна быть указана на уровне класса. Имя сущности по умолчанию должно быть таким же, как и имя класса. Его можно изменить с помощью параметра **value**.

Entity Manager

Entity Manager управляет жизненным циклом Entity. Он предоставляет основные методы для работы с объектами хранилищ данных, которые могут быть введены в ваши классы задач с помощью аннотаций **@Inject**.

Entity Manager выполняет следующие роли:

- Используя Entity Manager, разработчики могут реализовать API и инкапсулировать его с помощью единого интерфейса.
- Entity Manager позволяет вам читать, изменять и удалять сущности.
- Entity Manager управляет объектами, на которые ссылаются сущности.

Entity Manager предоставляет методы для получения объектов с помощью аннотации **@Query**. Запрос, который должен быть выполнен, определяется внутри аннотации **@Query** вместе с ожидаемым типом в ответе. Язык запросов использует строку запроса, подобную JPQL, для выбора объектов. В результате выполнения запроса возвращаются сущности. Возвращенные сущности могут быть обновлены и сохранены с помощью метода **persist()**. Кроме того, эти сущности могут быть удалены с помощью метода **delete()**.

Репозитории

В дополнение к непосредственному использованию менеджера сущности для операций с базами данных репозитории могут использоваться для упрощения и организации операций, связанных с БД. Существует два типа репозиториев: **Repository** и **CrudRepository**. **Repository** предоставляет работать с базовыми операциями, такими как сохранение или удаление объекта, а также создание пользовательских запросов с помощью аннотации **@Query**. **CrudRepository** предоставляет ту же функциональность, что и Repository , но предполагает, что данный объект имеет первичный ключ, помеченный столбцом **@Id**, и обеспечивает операции на основе PK.

Repository

Repository может использоваться для более легкого и удобного доступа к хранилищу данных. В дополнение к использованию основных методов из Repository, в него можно добавлять пользовательские запросы с помощью аннотации **@Query**. Пример использования аннотации **@Query** можно найти ниже.

```
import com.iba.kanclerrpa.persistence.Repository;
import com.iba.kanclerrpa.persistence.annotation.Param;
import com.iba.kanclerrpa.persistence.annotation.Query;


import java.awt.print.Book;
import java.util.List;

public interface BookRepo extends Repository<Book> {

    @Query("select * from Book")
    public List<Book> findAllCustom();

    @Query("select b from Book b order by b.value1")
    public List<Book> findAllOrderByVal();

    @Query("select b from Book b order by b.date1")
    public List<Book> findAllOrderByDate();

    @Query("select b from Book b where b.value1 = :value")
    public List<Book> findAllWhere(@Param("value") String value);

    @Query("select b from Book b where b.value1 = :value and b.value3 = :other")
    public List<Book> findAllWhereAnd(@Param("value") String value, @Param("other") int value3);

    @Query("select b.value1,b.date1 from Book b")
    public List<Book> findAllBasic();

}
```

CrudRepository

CrudRepository можно использовать только для хранилищ данных, содержащих первичные ключи. Таким образом, использование столбца аннотации **@Id** внутри объекта является необходимым условием для использования репозитория, поскольку все основные методы, предоставляемые репозиторием, основаны на использовании Id.

```
public interface UserRepository extends CrudRepository<User, Integer> {

@Query("select u from crud_example_users u where u.name = :name")
List<User> getBookByName(@Param("name") String name);

}
```

Аннотации сущностей (package com.iba.kanclerrpa.persistence.annotation)

Аннотации на уровне класса

@Entity - В этой аннотации указано, что это класс сущностей, который должен быть сохранен в хранилище данных. Параметр "value" задает логическое имя таблицы. Параметр "Type" определяет тип объекта, которым он может быть

DATASTORE	Хранилище данных, управляемое с обеих сторон: сервер управления и процесс автоматизации. Пользователь сервера управления или автоматизированного процесса может создать его или изменить его схему. Если автоматизированный процесс использует это хранилище данных в своем коде, и он не существует в сервере управления, он будет создан автоматически.
AP_RESULT_DATASTORE	Хранилище данных о результатах запуска автоматизированного процесса. Когда автоматизированный процесс использует это хранилище данных в своем коде, он будет воссоздан при каждом запуске.
DOCUMENT_SET	Набор документов. Управление возможно только со стороны сервера управления. Автоматизированный процесс не может создать/воссоздать набор документов.

Примечания к полям

@Id - указывает, что поле используется в качестве уникального идентификатора. Требуется для хранилищ данных, имеющих первичный ключ, поэтому, если первичный ключ отсутствует, аннотация @Id не требуется.

@Column("column_name") - сопоставляет поле из объекта с соответствующим столбцом таблицы.

Аннотации репозитория

@Query определяет JPQL запрос. Обратите внимание, что в запросах на объекты ссылаются по логическому имени.

@Param определяет именованный параметр запроса в аргументах метода.

Пример

Класс **Book** - это объект, который мы сохраним в хранилище данных.

Book.java

```
package org.example.domain;

import com.iba.kanclerrpa.persistence.annotation.Column;
import com.iba.kanclerrpa.persistence.annotation.Entity;
import com.iba.kanclerrpa.persistence.annotation.EntityType;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity(value = "crud_example_books", type = EntityType.DATASTORE)
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Book {
    @Column("name")
    private String name;

    @Column("author")
    private String author;
}
```

Класс **User** - это простая пользовательская сущность с первичным ключом.

User

```
package org.example.domain;

import com.iba.kanclerrpa.persistence.annotation.Column;
import com.iba.kanclerrpa.persistence.annotation.Entity;
import com.iba.kanclerrpa.persistence.annotation.EntityType;
import com.iba.kanclerrpa.persistence.annotation.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity(value = "crud_example_users", type = EntityType.DATASTORE)
@NoArgsConstructor
@AllArgsConstructor
@Data
public class User {
    @Column("id")
    @Id
    private Integer id;

    @Column("name")
    private String name;
}
```

Класс **BookRepository** позволяет взаимодействовать с хранилищем данных.

BookRepository.java

```
package org.example.repository;

import java.util.List;

import com.iba.kanclerrpa.persistence.Repository;
import com.iba.kanclerrpa.persistence.annotation.Param;
import com.iba.kanclerrpa.persistence.annotation.Query;
import org.example.domain.Book;


public interface BookRepository extends Repository<Book> {

    @Query("select b from crud_example_books b  where b.name = :name")
    List<Book> getBookByName(@Param("name") String name);
}
```

Пользовательский репозиторий предоставляет доступ к операциям на основе PK.

UserRepository

```
package org.example.repository;

import com.iba.kanclerrpa.persistence.CrudRepository;
import org.example.domain.User;


public interface UserRepository extends CrudRepository<User, Integer> {

}
```

SampleTask демонстрирует, как сохранять и получать объекты из репозитория.

SampleTask.java

```
package org.example.tasks;

import java.util.Arrays;
import java.util.List;

import javax.inject.Inject;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;

import lombok.extern.slf4j.Slf4j;
import org.example.domain.Book;
import org.example.repository.BookRepository;

@ApTaskEntry(name = "Sample Task")
@Slf4j
public class SampleTask extends ApTask {

    @Inject
    private BookRepository bookRepository;

    @Override
    public void execute() throws Exception {
        // Create list of books
        Book thinkingInJava = new Book("Thinking in Java", "Bruce Eckel");
        Book cipollino = new Book("Le avventure di Cipollino", "Giovanni Francesco Rodari");
        Book wadAndPeace = new Book("War and Peace", "Lev Tolstoy");

        List<Book> books = Arrays.asList(thinkingInJava, cipollino, wadAndPeace);

        // Save books to DataStore
        books.forEach(b -> bookRepository.save(b));

        // Get books from DataStore and display it
        for (Book book : bookRepository.findAll()) {
            log.info(book.toString());
        }

        // Find book by name
        for (Book book : bookRepository.getBookByName("Thinking in Java")) {
            log.info(book.toString());
        }

        // Clean repository
        for (Book book : bookRepository.findAll()) {
            bookRepository.delete(book);
        }
    }
}
```

UsersTask

```
package org.example.tasks;

import javax.inject.Inject;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;

import lombok.extern.slf4j.Slf4j;
import org.example.domain.User;
import org.example.repository.UserRepository;

@ApTaskEntry(name = "Users Task")
@Slf4j
public class UsersTask extends ApTask {

    @Inject
    private UserRepository userRepository;

    @Override
    public void execute() throws Exception {
        User root = userRepository.findById(0);
        if (root == null) {
            root = new User(0, "Vasya");
            userRepository.save(root);
            log.info("Created new root user");
        } else {
            log.info("Root user name is {}", root.getName());
        }

    }
}
```

Служба хранилища файлов

В качестве хранилища файлов S3 API используется MinIO.

Для работы с этим файловым хранилищем через S3 API у нас есть специальная утилита - **StorageManager**.

Начало работы с StorageManager

StorageManager может быть встроен в класс задачи:

```
import com.iba.kanclerrpa.utils.storage.StorageManager;

@Inject
private StorageManager storageManager;
```

Свойства конфигурации StorageManager

По умолчанию S3Manager использует следующие ключи свойств из конфигурации:

- **s3.url** - настроен по умолчанию на сервере управления. Требуется настроить в вашем локальном файле "properties".
- **s3.access_key** - настроен по умолчанию на сервере управления. Требуется настроить в вашем локальном файле "properties".
- **s3.secret_key** - настроен по умолчанию на сервере управления. Требуется настроить в вашем локальном файле "properties".
- **s3.bucket** - корзина, которая используется по умолчанию. Если свойство не настроено - по умолчанию будет использоваться корзина "resources".

Подключение к другим приложениям, совместимыми с S3

Вы также можете использовать S3Manager для работы с любыми другими приложениями, совместимыми с S3. Для этого вы можете создать свой собственный экземпляр S3Manager, используя следующий конструктор:

```
public S3Manager(String s3EndpointUrl, String s3AccessKey, String s3SecretKey, String bucket,
CannedAccessControlList acl)
```

Где **CannedAccessControlList acl** - это список управления доступом для объекта, который вы добавляете в файловое хранилище. Этот параметр конструктора является необязательным. По умолчанию используется значение **CannedAccessControlList.PublicRead**.

Загрузка файл в S3

Для загрузки файла вы можете использовать метод со следующей сигнатурой:

```
public void uploadFile(String bucket, String path, InputStream input, String contentType)
```

Где:

- **bucket** - целевая корзина.

- **path** - это путь к целевому файлу, который включает структуру папок, имя файла и расширение файла, но без указания имени корзины. Например: "*folder1/subfolder/my_file.txt*".
- **input** - объект входного потока.
- **contentType** - тип содержимого целевого файла. Например: "*text/plain*". "*image/png*", "*application/pdf*", "*text/html*". По умолчанию он пуст.

В следующем примере показано, как загрузить файл в S3 и получить его URL-адрес:

```
String fileName = UUID.randomUUID().toString() + ".txt";
String s3FilePath = "upload_file_test/" + fileName;
byte[] fileContent = "Text file content example".getBytes();

storageManager.uploadFile(StorageManager.DEFAULT_S3_BUCKET, s3FilePath, new ByteArrayInputStream(fileContent));
boolean fileUploadedSuccessfully = storageManager.isExist(StorageManager.DEFAULT_S3_BUCKET, s3FilePath);
if (fileUploadedSuccessfully) {
            String finalS3Url = storageManager.getFileLink(StorageManager.DEFAULT_S3_BUCKET, s3FilePath);
            log.info("Test file successfully uploaded. Final URL: {}", finalS3Url);
```

Скачивание файла из S3

Если файл в S3 имеет общедоступные права доступа на чтение, то его можно легко скачать по прямой ссылке, используя общепринятый подход для выполнения HTTP Get запроса.

Но если файл не имеет общедоступных прав доступа на чтение, то для скачивания требуются учетные данные S3, файл должен быть скачан с помощью S3 API. Для этого вы можете использовать следующий метод из S3Manager:

```
public InputStream getFile(String bucket, String path)
```

Где:

- **bucket** - целевая корзина.
- **path** - путь к целевому файлу, включая структуру папок, имя файла и расширение файла, но без указания имени корзины. Например: "*folder1/subfolder/my_file.txt*".

В следующем примере показано, как загрузить текстовый файл из S3 и прочитать его содержимое:

```
InputStream fileContentInputStream = storageManager.getFile(StorageManager.DEFAULT_S3_BUCKET, s3FilePath);
String fileContent = toString(fileContentInputStream);
 log.info("Test file successfully downloaded. File content: {}", fileContent);
```

Удаление файла из S3

Для удаления файла из S3 следует использовать метод со следующей сигнатурой:

```
public void deleteFile(String bucket, String path)
```

Где:

- **bucket** - целевая корзина.
- **path** - путь к целевому файлу, включая структуру папок, имя файла и расширение файла, но без указания имени корзины. Например: "*folder1/subfolder/my_file.txt*".

Получение списка файлов

Для получения списка файлов можно использовать метод со следующей сигнатурой:

```
public List<String> listFiles(String bucket, String path, String filter)
```

Где:

- **bucket** - целевая корзина.
- **path** - Этот параметр используется как префикс к пути файла, который ищут. Например: "*folder1/subfolder/*", "*folder2/*".
- **filter** - Regexp выражения для фильтрации файлов. Например: "*.*\\.txt*"" чтобы найти все файлы с расширением .txt.

В следующем примере показано, как получить список всех файлов txt:

```
String baseFolder = "upload_file_test/";
List<String> filesPathList = storageManager.listFiles(StorageManager.DEFAULT_S3_BUCKET, baseFolder, ".*\\.txt");
for (String filePath : filesPathList) {
     log.info("The following file exist on S3: {}", filePath);
```

Полный пример

UploadAndReadFilesFromS3.java

```
package org.example.tasks;


import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.utils.storage.StorageManager;
import lombok.extern.slf4j.Slf4j;


import javax.inject.Inject;


import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.List;

@ApTaskEntry(name = "Upload to S3")
@Slf4j
public class UploadAndReadFilesFromS3 extends ApTask {

    @Inject
    private StorageManager storageManager;

    private String baseFolder;

    private String s3FilePath;

    private byte[] fileContent;


    @AfterInit
    public void init() {
        String fileName = "test.txt";
        baseFolder = "upload_file_test/";
        s3FilePath = baseFolder + fileName;
        fileContent = "Text file content example".getBytes();
    }

    @Override
    public void execute() throws IOException {
        uploadFile();
        printListFiles();
        readFile();
    }

    private void uploadFile() {
        storageManager.uploadFile(StorageManager.DEFAULT_S3_BUCKET, s3FilePath, new ByteArrayInputStream
(fileContent));
```

```
        boolean fileUploadedSuccessfully = storageManager.isExist(StorageManager.DEFAULT_S3_BUCKET, s3FilePath);
        if (fileUploadedSuccessfully) {
            String finalS3Url = storageManager.getFileLink(StorageManager.DEFAULT_S3_BUCKET, s3FilePath);
            log.info("Test file successfully uploaded. Final URL: {}", finalS3Url);
        }
    }

    private void printListFiles() {
        List<String> filesPathList = storageManager.listFiles(StorageManager.DEFAULT_S3_BUCKET, baseFolder, ".
*\\.txt");
        for (String filePath : filesPathList) {
            log.info("The following file exist on S3: {}", filePath);
        }
    }

    private void readFile() throws IOException {
        InputStream fileContentInputStream = storageManager.getFile(StorageManager.DEFAULT_S3_BUCKET,
s3FilePath);
        String fileContent = toString(fileContentInputStream);
        log.info("Test file successfully downloaded. File content: {}", fileContent);
    }

    public static String toString(InputStream inputStream) throws IOException {
        ByteArrayOutputStream result = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        for (int length; (length = inputStream.read(buffer)) != -1; ) {
            result.write(buffer, 0, length);
        }
        return result.toString(StandardCharsets.UTF_8.name());
    }
}
```

Служба уведомлений

Обзор

Интерфейс службы уведомлений предоставляет доступ к действиям уведомлений, таким как отправка сообщений, вложений и шаблонов по каналу.

Канал

Каналы используются для определения получателей уведомления и типа доставки.

Текущая версия поддерживает два типа каналов - Email и Slack. Их конфигурация хранится в разделе конфигурации CS под ключом notification.channels.config.

```
{
  "email": {
    "default-encoding": "utf-8",
    "host": "mailin.iba",
    "port": "25",
    "protocol": "smtp",
    "mail.smtp.auth": "false",
    "mail.smtp.starttls.enable": "true",
    "mail.smtp.ssl.trust": "mailin.iba",
    "from": "test@ibagroup.eu"
  },
  "slack": {
    "token": "xoxb-1289667556037-2162753905523-oeC78BlZFBZHYvNhxnsfrFjx"
  }
}
```

Настройка канала электронной почты

Все параметры конфигурации показаны выше. Например, конфигурация Gmail описана здесь.

Параметр "outputContentType" позволяет выбрать формат вывода сообщения между text и html. По умолчанию тип определяется автоматически.

Конфигурация Slack канала

1. Создать приложение Slack и добавить необходимые области Bot OAuth Scopes такие как chat:write. Ссылка на конфигурацию приложения Slack - https://api.slack.com/apps.
2. Перейти в *CS Configuration* *notification.channels.config* и добавить новую конфигурацию в формате - "<*config_name*>" : {"token" : "<*Bot User OAuth Token*>"}
3. Создать Slack канал и добавить в него свое приложение. Скопируйте идентификатор канала, поскольку он будет использоваться в дальнейшем (идентификатор начинается с xoxb-...).
4. Создать уведомление. Использовать свое config_name в качестве Config Name и Channel ID в качестве получателя.

Шаблон

Шаблоны можно использовать, если необходимо динамически вставлять некоторые данные в сообщение. Сервер управления поддерживает два механизма шаблонов:

- Thymeleaf
  В настоящее время реализованы обработчики ThymeleafHtml и ThymeleafText. Ключевое различие между режимом текстового шаблона и режимом разметки заключается в том, что в текстовом шаблоне нет меток, в которые можно было бы вставить логику в виде атрибутов, поэтому приходится полагаться на другие механизмы. Подробнее о режимах шаблонов читайте здесь. Также здесь есть полезная документация о диалекте Thymeleaf here.
- FreeMarker
  Apache FreeMarker - это двигатель шаблонов: библиотека Java для создания текстовых выходных данных (веб-страниц HTML, электронных писем, файлов конфигурации, исходного кода и т.д.) на основе шаблонов и изменяющихся данных.

Шаблоны написаны на языке шаблонов FreeMarker (FTL), который представляет собой простой специализированный язык
Руководство по началу работы находится здесь.

Все параметры передаются в механизм шаблонов в виде параметров Map<String, ?>

Существует пример шаблона FreeMarker для примера из библитеки RPA Debtors Analysis.

debtors_template.txt

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <style>
        body {
                background: #fafafa url(https://jackrugile.com/images/misc/noise-diagonal.png);
                color: #444;
                font: 100%/30px 'Helvetica Neue', helvetica, arial, sans-serif;
                text-shadow: 0 1px 0 #fff;
        }

        strong {
                font-weight: bold;
        }

        em {
                font-style: italic;
        }

        table {
                background: #f5f5f5;
                border-collapse: separate;
                box-shadow: inset 0 1px 0 #fff;
                font-size: 14px;
                line-height: 24px;
                margin: 30px auto;
                text-align: center;
                width: 800px;
        }

        caption {
                font-size: 18px;
        font-weight: bold;
        }

        th {
                background: url(https://jackrugile.com/images/misc/noise-diagonal.png), linear-gradient(#777,
#444);
                border-left: 1px solid #555;
                border-right: 1px solid #777;
                border-top: 1px solid #555;
                border-bottom: 1px solid #333;
                box-shadow: inset 0 1px 0 #999;
                color: #fff;
                font-weight: bold;
                padding: 10px 15px;
                position: relative;
                text-shadow: 0 1px 0 #000;
        }

        th:after {
                background: linear-gradient(rgba(255, 255, 255, 0), rgba(255, 255, 255, .08));
                content: '';
                display: block;
                height: 25%;
                left: 0;
                margin: 1px 0 0 0;
                position: absolute;
                top: 25%;
                width: 100%;
```

```
        }

        th:first-child {
                border-left: 1px solid #777;
                box-shadow: inset 1px 1px 0 #999;
        }

        th:last-child {
                box-shadow: inset -1px 1px 0 #999;
        }

        td {
                border-right: 1px solid #fff;
                border-left: 1px solid #e8e8e8;
                border-top: 1px solid #fff;
                border-bottom: 1px solid #e8e8e8;
                padding: 10px 15px;
                position: relative;
                transition: all 300ms;
        }

        td:first-child {
                box-shadow: inset 1px 0 0 #fff;
        }

        td:last-child {
                border-right: 1px solid #e8e8e8;
                box-shadow: inset -1px 0 0 #fff;
        }

        tr {
                background: url(https://jackrugile.com/images/misc/noise-diagonal.png);
        }

        tr:nth-child(odd) td {
                background: #f1f1f1 url(https://jackrugile.com/images/misc/noise-diagonal.png);
        }

        tr:last-of-type td {
                box-shadow: inset 0 -1px 0 #fff;
        }

        tr:last-of-type td:first-child {
                box-shadow: inset 1px -1px 0 #fff;
        }

        tr:last-of-type td:last-child {
                box-shadow: inset -1px -1px 0 #fff;
        }

        tbody:hover td {
                color: transparent;
                text-shadow: 0 0 3px #aaa;
        }

        tbody:hover tr:hover td {
                color: #444;
                text-shadow: 0 1px 0 #fff;
        }

        </style>
</head>
<body>

<div class="container">
        <table class="responsive-table">
                <caption>   </caption>
                <thead>
                <tr>
                        <th scope="col"></th>
                        <th scope="col"></th>
```

```
                        <th scope="col"></th>
                        <th scope="col"></th>
                        <th scope="col"></th>
                        <th scope="col"></th>
                </tr>
                </thead>
                <tfoot>
                <tr>
                        <td colspan="7" align="left">
                                Sources:
                                <#list links?keys as key>
                                        <a href="${links[key]}" rel="external">${key}</a>.
                                </#list>
                                Data is current as of ${date}.
                        </td>
                </tr>
                </tfoot>
                <tbody>
                <#list debtors as debtor>
                        <#if debtor??>
                                <tr>
                                        <td data-title="">${debtor.companyID}</td>
                                        <td data-title="">${debtor.companyName}</td>
                                        <#if debtor.customsFound??>
                                                <td data-title="" data-type="currency">${(debtor.customsFound=="
true") ? then("", "")}</td>
                                        </#if>
                                        <#if debtor.nationalBankFound??>
                                                <td data-title="" data-type="currency">${(debtor.
nationalBankFound=="true") ? then("", "")}</td>
                                        </#if>
                                        <#if debtor.taxesFound??>
                                                <td data-title="" data-type="currency">${(debtor.taxesFound=="
true") ? then("", "")}</td>
                                        </#if>
                                        <#if debtor.socialfundFound??>
                                                <td data-title="" data-type="currency">${(debtor.
socialfundFound=="true") ? then("", "")}</td>
                                        </#if>
                                </tr>
                        </#if>
                </#list>
                </tbody>
        </table>
</div>
</body>
</html>
```

Использование

1. Создать канал в разделе Управление уведомлениями. Имя конфигурации должно быть одним из ключей notification.channels.config. См. Создание нового канала.

2. Создать шаблон в разделе Управление уведомлениями. Он может быть проверен одним из объектов - процессом автоматизации, типом документа, узлом, расписанием. См. Создание нового шаблона.

3. Внедрить службу уведомлений в ApTask, а также имена каналов уведомлений и шаблонов.

ApTask configuration

```
@Inject
private NotificationService notificationService;

@Configuration(value = "notificationChannel", defaultValue = "Default Notification Channel")
private String notificationChannel;
```

```
@Configuration(value = "notificationTemplate", defaultValue = "Default Email Template")
private String notificationTemplate;
```

4. Использовать один из предоставленных методов интерфейса:

NotificationService.java

```
/**
* Evaluates template with provided parameters.
* @param templateName template name
* @param params params for template
* @return the array of byte for processed template
*/
byte[] evaluateTemplate(String templateName, Map<String, ?> params);

/**
* Evaluates template with provided parameters and sends result into specified channel.
* @param templateName template name
* @param params params for template
* @param channelName name of channel for sending
* @param attachments list of attachments for channel if needed
*/
void evaluateTemplateAndSend(String templateName, Map<String, ?> params, String channelName, List<File>
attachments);

/**
* Sends notification into specified channel.
* @param channelName name of channel for sending
* @param text body text to send
* @param attachments list of attachments for channel if needed
*/
void sendNotification(String channelName, String text, List<File> attachments);
```

Примеры

- Оценка и отправка шаблона для примера из библитеки RPA Debtors Analysis.

GenerateEmailReport.java

```
package com.iba.kanclerrpa.debtors.task;
import com.iba.kanclerrpa.debtors.domain.Debtor;
import com.iba.kanclerrpa.debtors.repository.DebtorRepository;
import com.iba.kanclerrpa.debtors.to.DebtorsResultTo;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Configuration;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.service.ConfigurationService;
import com.iba.kanclerrpa.engine.service.NotificationService;

import javax.inject.Inject;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@ApTaskEntry(name = "Generate Email Report")
public class GenerateEmailReport extends ApTask {

    @Input(ExtractDebtors.DEBTOR_KEY)
    @Output(ExtractDebtors.DEBTOR_KEY)
    private DebtorsResultTo debtorsResult;
```

```
    @Configuration(value = "notificationChannel", defaultValue = "Debtor Notification Channel")
    private String notificationChannel;

    @Configuration(value = "notificationTemplate", defaultValue = "Debtor Email Template")
    private String notificationTemplate;

    @Inject
    private DebtorRepository repo;

    @Inject
    private NotificationService notificationService;

    @Override
    public void execute() {

        List<Debtor> debtors = repo.fetchDebtors(debtorsResult.getDebtorIds());

        Map data = new HashMap<>();
        data.put("debtors", debtors);
        data.put("date", new SimpleDateFormat("MMMM dd, yyyy").format(new Date()));
        data.put("links", getLinks());
        notificationService.evaluateTemplateAndSend(notificationTemplate, data, notificationChannel, new
ArrayList<>());
        debtorsResult.setGenerateEmailReportComplete(true);
    }

    @Inject
    private ConfigurationService cfg;

    private Map getLinks() {
        HashMap<String, String> links = new HashMap<>();
        links.put("", cfg.get("gtk.client.url", "http://www.gtk.gov.by/ru/dolg-test-ru/"));
        links.put("", cfg.get("nbrb.client.url", "http://www.nbrb.by/system/banks/guaranteesregister/"));
        links.put("", cfg.get("court.client.url", "http://service.court.by/ru/juridical/judgmentresults
/"));
        links.put("", cfg.get("nalog.client.url", "http://www.portal.nalog.gov.by/debtor/"));
        links.put("", cfg.get("ssf.client.url", "https://ssf.gov.by/ru/debtors-ru/"));
        return links;
    }

}
```

- Отправка уведомления с вложением в образце SAP.

GenerateAndSendReportTask.java

```
package com.iba.kanclerrpa.ap.creatematerial.task;

import com.iba.kanclerrpa.ap.creatematerial.ApConstants;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Configuration;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.service.NotificationService;
import lombok.extern.slf4j.Slf4j;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import javax.inject.Inject;
import java.io.File;
import java.io.FileOutputStream;
import java.util.Collections;
import java.util.HashMap;

@ApTaskEntry(name = "Generate and send report")
@Slf4j
public class GenerateAndSendReportTask extends ApTask {
```

```
    @Input
    private String materialCode;

    @Inject
    private NotificationService notificationService;

    @Configuration(value = "notificationTemplate", defaultValue = "SAP Email Template")
    private String notificationTemplate;

    @Configuration(value = "notificationChannel", defaultValue = "SAP Notification Channel")
    private String notificationChannel;

    @Override
    public void execute() throws Exception {
        File attachment = File.createTempFile("execution_results_", ".xlsx");
        createSpreadsheet(attachment);
        notificationService.evaluateTemplateAndSend(notificationTemplate, new HashMap<>(),
notificationChannel, Collections.singletonList(attachment));
        attachment.delete();
    }

    private void createSpreadsheet(File saveTo) throws Exception {
        try (XSSFWorkbook workBook = new XSSFWorkbook(getClass().getResourceAsStream(ApConstants.
TEMPLATE_NAME))) {
            Sheet sheet = workBook.getSheet("Sheet1");
            Row row = sheet.createRow(1);
            row.createCell(0).setCellValue(this.materialCode);
            row.createCell(1).setCellValue(ApConstants.RPA_BASE_UNIT);
            row.createCell(2).setCellValue(String.format(ApConstants.RPA_DESCR_STRING_TEMPL, this.
materialCode));
            workBook.write(new FileOutputStream(saveTo));
        }
    }
}
```

Служба хранилища секретов

Служба хранилища секретов предоставляет доступ к зашифрованному хранилищу секретов.

Чтобы использовать службу хранилища секретов секретов, надо ввести объект службы хранилища секретов в задачу.

```
@ApTaskEntry(name = "Vault task")
@Slf4j
public class VaultTask extends ApTaskBase {
    @Inject
    VaultService vaultService;
}
```

В настоящее время служба хранилища секретов предоставляет только методы getter.

Хранение учетных данных

Обычно в службе хранилища секретов хранятся учетные данные пользователя. Чтобы поместить учетные данные в службу хранилища секретов локально (Standalone конфигурация), они должны быть предоставлены в виде декодированной карты Base64 JSON, содержащей ключи пользователя и пароля, например:

Чтобы сохранить учетные данные с именем пользователя "admin" и паролем "123456", вам следует создать следующий JSON:

vault.properties

```
{"user": admin", "password": "123456"}
```

Затем указать какое-нибудь имя. Ваш файл "**resources/vault.properties**" будет содержать следующую пару ключ-значение:

vault.properties

```
mail.user={"user": admin", "password": "123456"}
```

Пример получения учетных данных пользователя:

```
SecretCredentials secrets = vaultService.getSecret("mail.user", SecretCredentials.class);
log.info("'mail.user' user:{} password:{}", secrets.getUser(), secrets.getPassword());
```

Хранение любой строки

Существует также возможность хранить любое строковое значение в секретном значении.

vault.properties

```
my.alias=SGVsbG8gZnJvbSBzZWNyZXQgdmF1bHQh
```

Пример извлечения строкового значения из службы хранилища секретов:

```
String myValue = vaultService.getSecret("my.alias", String.class);
```

В конфигурации узла секреты сохраняются в хранилище HashiCorp, поддерживаемом базой данных PostgresSQL .

В этом случае секретные записи добавляются вручную со страницы секретного хранилища секретов Сервера управления или с помощью API Сервера управления. См. Хранилище секретов.

Пример

VaultSampleAp.java

```
package org.example.ap;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

import org.example.tasks.VaultTask;

@Slf4j
@ApModuleEntry(name = "SecretVault Sample", description = "This process automates a lot of work...")
public class VaultSampleAp extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), VaultTask.class).get();
    }

}
```

VaultTask.java

```
package org.example.tasks;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import com.iba.kanclerrpa.engine.service.VaultService;
import lombok.extern.slf4j.Slf4j;

import javax.inject.Inject;

@ApTaskEntry(name = "Vault task")
@Slf4j
public class VaultTask extends ApTask {
    @Inject
    VaultService vaultService;

    @Override
    public void execute() {
        SecretCredentials secrets = vaultService.getSecret("mail.user", SecretCredentials.class);
        log.info("'mail.user' user:{} password:{}", secrets.getUser(), secrets.getPassword());

        String myValue = vaultService.getSecret("my.alias", String.class);
        log.info("my.alias:{}", myValue);
    }
}
```

ModuleLocalRun

```
package org.example.ap;

import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {
```

```
    public static void main(String[] args) {
        ApModuleRunner.localLaunch(VaultSampleAp.class, new DevelopmentConfigurationModule(args));
        //ApModuleRunner.localLaunch(VaultSampleAp.class);
    }
}
```

Создание процесса автоматизации

RPA на примере

Все примеры RPA процессов можно найти по следующей ссылке: https://<CS host>/nexus/#browse/browse:rpaplatform:com%2Fiba%2Fsamples

- Командная строка автоматизации
- Скачивание файла
- Автоматизация блокнота
- SAP
- Автоматизация на основе захвата образов на экране
- Создание скриншотов
- Загрузка файлов
- Oracle формы

Командная строка автоматизации

Ниже приведены 2 способа выполнения команд с помощью командной строки и получения результата:

1. Использовать **ProcessBuilder**

Run cmd commands

```
String[] commands = {"cmd.exe", "/c", "java -version"};
ProcessBuilder builder = new ProcessBuilder(commands);

//Redirect the process's standard error into standard output
builder.redirectErrorStream(true);
Process process = builder.start();

BufferedReader stdInput = new BufferedReader(new InputStreamReader(process.getInputStream()));

log.info("Here is the output of the command:\n");
String s;
while ((s = stdInput.readLine()) != null) {
        log.info(s);
}
```

2. Использовать **Runtime.getRuntime().exec()**

Run cmd commands

```
Runtime runtime = Runtime.getRuntime();
String[] commands = {"cmd.exe", "/c", "ping google.com"};
Process process = runtime.exec(commands);

BufferedReader stdInput = new BufferedReader(new InputStreamReader(process.getInputStream()));
BufferedReader stdError = new BufferedReader(new InputStreamReader(process.getErrorStream()));

// Read the output from the command
log.info("Here is the standard output of the command:\n");
String s;
while ((s = stdInput.readLine()) != null) {
        log.info(s);
}

// Read any errors from the attempted command
log.info("Here is the standard error of the command (if any):\n");
while ((s = stdError.readLine()) != null) {
        log.info(s);
}
```

Пример

CmdTask.java

```
package org.example.tasks;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import lombok.extern.slf4j.Slf4j;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
@ApTaskEntry(name = "Cmd Automation")
@Slf4j
public class CmdTask extends ApTask {

    @Override
    public void execute() {

        try {
            printJavaVersion();
            pingGoogle();
        } catch (Exception e) {
            //do something
        }
    }

    private void printJavaVersion() throws IOException {

        String[] commands = {"cmd.exe", "/c", "java -version"};
        ProcessBuilder builder = new ProcessBuilder(commands);

        //Redirect the process's standard error into standard output
        builder.redirectErrorStream(true);
        Process process = builder.start();

        BufferedReader stdInput = new BufferedReader(new InputStreamReader(process.getInputStream()));

        log.info("Here is the output of the command:\n");
        String s;
        while ((s = stdInput.readLine()) != null) {
            log.info(s);
        }
    }

    private void pingGoogle() throws IOException {

        Runtime runtime = Runtime.getRuntime();
        String[] commands = {"cmd.exe", "/c", "ping google.com"};
        Process process = runtime.exec(commands);

        BufferedReader stdInput = new BufferedReader(new InputStreamReader(process.getInputStream()));
        BufferedReader stdError = new BufferedReader(new InputStreamReader(process.getErrorStream()));

        // Read the output from the command
        log.info("Here is the standard output of the command:\n");
        String s;
        while ((s = stdInput.readLine()) != null) {
            log.info(s);
        }

        // Read any errors from the attempted command
        log.info("Here is the standard error of the command (if any):\n");
        while ((s = stdError.readLine()) != null) {
            log.info(s);
        }
    }
}
```

CmdDemoAp.java

```
package org.example.ap;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

import org.example.tasks.CmdTask;
```

```
@Slf4j
@ApModuleEntry(name = "Cmd Automation Demo", description = "This process automates a lot of work...")
public class CmdDemoAp extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), CmdTask.class).get();
    }
}
```

LocalRunner.java

```
package org.example.ap;

import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(CmdDemoAp.class, new DevelopmentConfigurationModule(args));
        //ApModuleRunner.localLaunch(CmdDemoAp.class);
    }
}
```

Скачивание файла

При скачивании файлов из интернета используя RPA часто возникает проблема: необходимо понять, когда загрузка завершена. Ниже приведен пример того, как такая функциональность может быть реализована.

Ниже приведен код метода **waitFile()** из класса **Task**. Этот метод, использующий NIO2 API, ожидает создания файла в каталоге для скачивания. Полный код класса, а также другие классы, необходимые для тестирования его работы, приведены ниже.

waitFile method

```
private String waitFile(int timeoutSeconds) throws IOException {

        Path directory = Paths.get(filePath);

        try (WatchService service = directory.getFileSystem().newWatchService()) {
                directory.register(service, StandardWatchEventKinds.ENTRY_CREATE);

                long timeout = TimeUnit.NANOSECONDS.convert(timeoutSeconds, TimeUnit.SECONDS);
                while (timeout > 0L) {
                        final long start = System.nanoTime();
                        WatchKey key = service.poll(timeout, TimeUnit.NANOSECONDS);
                        if (key != null) {
                                for (WatchEvent<?> event : key.pollEvents()) {
                                        Path path = (Path) event.context();

                                        if (path.toString().toLowerCase().endsWith(FILE_EXTENSION)) {
                                                return path.getFileName().toString();
                                        }
                                }
                                key.reset();
                                timeout -= System.nanoTime() - start;
                        } else {
                                break;
                        }
                }
        } catch (InterruptedException ignore) {
        }
        return null;
}
```

Используя класс **ChromeOptions**, есть возможность передать параметры, которые позволяют начать загрузку в указанный каталог без появления диалогового окна. Пример взаимодействия с таким окном можно найти в статье Загрузка файлов.

DownloadFile.java

```
import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Driver;
import com.iba.kanclerrpa.engine.annotation.DriverParameter;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.driver.DriverParams;
import lombok.extern.slf4j.Slf4j;
import org.example.WebApplication;
import org.example.page.MainPage;
import org.openqa.selenium.Capabilities;
import org.openqa.selenium.chrome.ChromeOptions;

import java.io.IOException;
import java.nio.file.*;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;
import java.util.function.Supplier;

@ApTaskEntry(name = "Download file")
```

```
@Slf4j
public class DownloadFile extends ApTask {

    private static final String FILE_EXTENSION = ".bin";

    @Driver(value = DriverParams.Type.BROWSER, param = {
            @DriverParameter(key = DriverParams.Browser.SELENIUM_NODE_CAPABILITIES, initializerName =
DriverParams.BrowserCapabilities.CHROME)})
    private BrowserDriver browserDriver;

    private String url;

    private static final String filePath = System.getProperty("user.home") + "\\Downloads"; // hetzner.file.path

    public static final class ChromeInitializer implements Supplier<Capabilities> {

        @Override
        public Capabilities get() {
            ChromeOptions chromeOptions = new ChromeOptions();

            HashMap<String, Object> chromePrefs = new HashMap<>();
            chromePrefs.put("profile.default_content_settings.popups", 0);
            chromePrefs.put("download.default_directory", filePath);

            chromeOptions.setExperimentalOption("prefs", chromePrefs);
            return chromeOptions;
        }
    }

    @AfterInit
    public void init() {
        url = getConfigurationService().get("hetzner.app.url", "https://speed.hetzner.de/");
    }

    @Override
    public void execute() {

        WebApplication webApplication = new WebApplication(browserDriver);
        MainPage mainPage = webApplication.open(url);

        mainPage.dovnloadTestFile();

        String downloadedFileName = null;

        try {
            downloadedFileName = waitFile(500);
        } catch (IOException e) {
            // do something
        }

        if (downloadedFileName != null) {
            log.info("Downloaded file name: " + downloadedFileName);
        } else {
            log.info("File wasn't downloaded! ");
        }
    }

    private String waitFile(int timeoutSeconds) throws IOException {

        Path directory = Paths.get(filePath);

        try (WatchService service = directory.getFileSystem().newWatchService()) {
            directory.register(service, StandardWatchEventKinds.ENTRY_CREATE);

            long timeout = TimeUnit.NANOSECONDS.convert(timeoutSeconds, TimeUnit.SECONDS);
            while (timeout > 0L) {
                final long start = System.nanoTime();
                WatchKey key = service.poll(timeout, TimeUnit.NANOSECONDS);
                if (key != null) {
                    for (WatchEvent<?> event : key.pollEvents()) {
                        Path path = (Path) event.context();
```

```
                    if (path.toString().toLowerCase().endsWith(FILE_EXTENSION)) {
                        return path.getFileName().toString();
                    }
                }
                key.reset();
                timeout -= System.nanoTime() - start;
            } else {
                break;
            }
        }
    } catch (InterruptedException ignore) {
    }
    return null;
  }
}
```

MainPage.java

```
import com.iba.kanclerrpa.engine.rpa.page.WebPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Wait;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class MainPage extends WebPage {

    @FindBy(xpath = "//a[@href='100MB.bin']")
    @Wait(waitFunc = Wait.WaitFunc.VISIBLE)
    private WebElement fileLink100mb;

    public void dovnloadTestFile() {
        fileLink100mb.click();
    }
}
```

Код автоматизированного процесса:

Module.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import org.example.tasks.DownloadFile;


@ApModuleEntry(name = "File Download Automation Demo", description = "This process automates a lot of work...")
public class Module extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), DownloadFile.class).get();
    }
}
```

Класс для работы с веб приложением:

WebApplication.java

```
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.element.BrowserElement;
import org.example.page.MainPage;

public class WebApplication extends Application<BrowserDriver, BrowserElement> {
    public WebApplication(BrowserDriver driver) {
        super(driver);
```

```
    }

    @Override
    public MainPage open(String... args) {
        String gmailUrl = args[0];
        getDriver().get(gmailUrl);
        return createPage(MainPage.class);
    }
}
```

Локальное средство запуска:

ModuleLocalRun.java

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(Module.class, new DevelopmentConfigurationModule(args));
        //ApModuleRunner.localLaunch(Module.class);
    }
}
```

Автоматизация блокнота

В этом примере показано, как автоматизировать приложение Windows с помощью **Desktop driver**.

Пример состоит из нескольких частей:

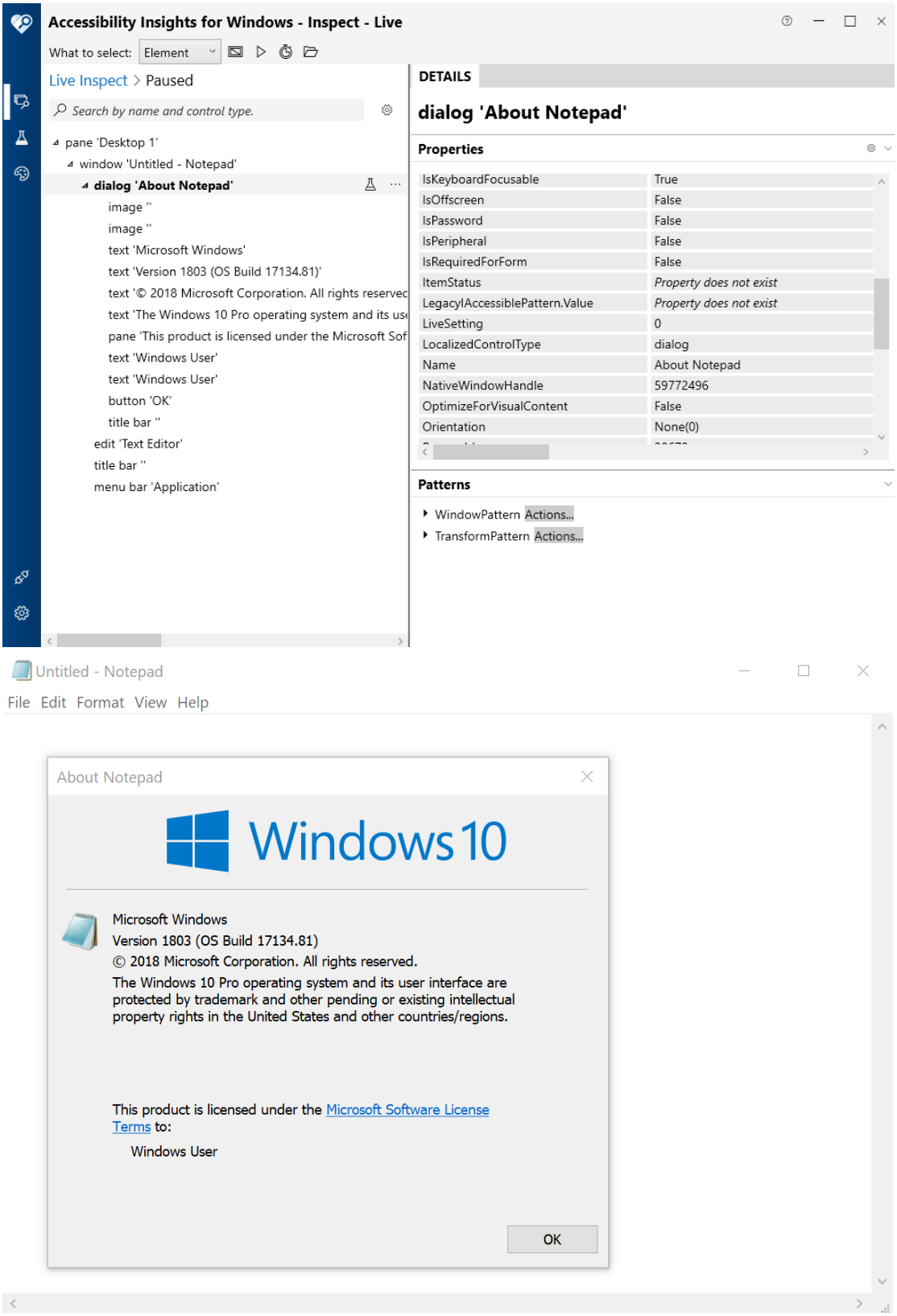
1. Открытие приложения **Notepad**
2. Открытие окна **About Notepad**
3. Получение информации о приложении
4. Сохранение полученной информации в **Notepad**
5. Сохранение файла - взаимодействие с окном **Save As**

Класс приложения открывает Блокнот и возвращает объект **MainPage**.

NotepadApplication.java

```
package com.iba.kanclerrpa.notepad;

import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.DesktopDriver;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import com.iba.kanclerrpa.notepad.page.MainPage;

public class NotepadApplication extends Application<DesktopDriver, DesktopElement> {

        public NotepadApplication(DesktopDriver driver) {
                super(driver);
        }

        @Override
        public MainPage open(String... args) {
                String appPath = args[0];
                getDriver().get(appPath);
                return createPage(MainPage.class);
        }
}
```

Окно **About Notepad** является внутренним элементом главного окна приложения. То же самое относится и к окну **Save As**, поэтому нам не нужно переключаться между окнами.



Класс **MainPage** содержит методы для работы с главным окном приложения: ввод текста, открытие окна **About Notepad**, сохранение документа, закрытие приложения.

MainPage.java

```
package com.iba.kanclerrpa.notepad.page;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import com.iba.kanclerrpa.engine.rpa.locator.By;
import com.iba.kanclerrpa.engine.rpa.locator.UIQuery;
import com.iba.kanclerrpa.engine.rpa.page.DesktopPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import org.apache.commons.io.FilenameUtils;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.Keys;

import java.io.File;

public class MainPage extends DesktopPage {

        @FindBy(name = "Help")
        @WithTimeout(time = 3)
        private DesktopElement helpMenu;

        @FindBy(name = "About Notepad")
        @WithTimeout(time = 3)
        private DesktopElement aboutNotepadMenuItem;

        private String title;

        public AboutNotepadPage openAboutWindow() {
                helpMenu.click();
                aboutNotepadMenuItem.click();
                return createPage(AboutNotepadPage.class);
        }

        @AfterInit
        public void init() {
                title = "Untitled - Notepad";
                this.switchToMainWindow();
        }

        public void switchToMainWindow() {
                getDriver().waitAndSwitchToWindow(title, 5);
        }

        public void printText(String text) {
                getDriver().findElement(By.desktopSearch(UIQuery.builder().name("Text Editor").build())).
sendKeys(text);
        }

        public void saveAs(File file) {
                getDriver().findElement(By.desktopSearch(UIQuery.builder().name("Text Editor").build())).
click();
                getDriver().getKeyboard().pressKey(Keys.LEFT_CONTROL);
                getDriver().getKeyboard().sendKeys("s");
                getDriver().getKeyboard().releaseKey(Keys.LEFT_CONTROL);
                getDriver().waitAndSwitchToWindow("#32770", "Save As", 5);
                DesktopElement textBox = getDriver().findElement(By.desktopSearch(UIQuery.builder().className
("Edit").name("File name:").build()));
                textBox.clear();
                textBox.sendKeys(file.getAbsolutePath());
                getDriver().findElement(By.desktopSearch(UIQuery.builder().name("Save").build())).click();
                title = FilenameUtils.removeExtension(file.getName()) + " - Notepad";
                switchToMainWindow();
        }

        public void close() {
                switchToMainWindow();
                this.getDriver().close();
        }
}
```

Окно **About Notepad** содержит несколько похожих элементов с текстовой информацией о приложении. Метод **getContent()** возвращает список строк с текстом из этих элементов. Метод **getContent()** закрывает окно.

AboutNotepadPage.java

```
package com.iba.kanclerrpa.notepad.page;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import com.iba.kanclerrpa.engine.rpa.page.DesktopPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import org.openqa.selenium.support.FindBy;

import java.util.List;
import java.util.stream.Collectors;

public class AboutNotepadPage extends DesktopPage {

        @FindBy(tagName = "Text")
        private List<DesktopElement> textContent;

        @FindBy(tagName = "Button", name = "OK")
        @WithTimeout(time = 3)
        private DesktopElement okBtn;

        @AfterInit
        public void init() {
                getDriver().waitAndSwitchToWindow("About Notepad", 5);
        }

        public List<String> getContent() {
                return this.textContent.subList(0, 1)
                        .stream().map(DesktopElement::getName).collect(Collectors.toList());
        }

        public void close() {
                this.okBtn.click();
        }
}
```

CreateNote - класс задачи, содержащий основной алгоритм.

CreateNote.java

```
package com.iba.kanclerrpa.notepad.task;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Driver;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.rpa.driver.DesktopDriver;
import com.iba.kanclerrpa.engine.rpa.driver.DriverParams;
import com.iba.kanclerrpa.notepad.NotepadApplication;
import com.iba.kanclerrpa.notepad.page.AboutNotepadPage;
import com.iba.kanclerrpa.notepad.page.MainPage;

import java.io.File;
import java.util.List;
import java.util.UUID;

@ApTaskEntry(name = "Get product list from InvoicePlane")
public class CreateNote extends ApTask {

        @Driver(DriverParams.Type.DESKTOP)
        private DesktopDriver desktopDriver;
```

```
        private String appPath;

        private String filePath;

        @AfterInit
        public void init() {
                appPath = getConfigurationService().get("notepad.app.path", "C:\\Windows\\system32\\notepad.
exe");
                filePath = getConfigurationService().get("notepad.file.path", System.getProperty("user.home"));
        }

        @Override
        public void execute() {

                NotepadApplication notepadApplication = new NotepadApplication(desktopDriver);
                MainPage mainPage = notepadApplication.open(appPath);
                AboutNotepadPage aboutNotepadPage = mainPage.openAboutWindow();
                List<String> content = aboutNotepadPage.getContent();
                aboutNotepadPage.close();
                mainPage.switchToMainWindow();
                for (String s : content) {
                        mainPage.printText(s + "\n");
                }

                File file = new File(filePath + "\\" + UUID.randomUUID().toString() + ".txt");
                mainPage.saveAs(file);
                mainPage.close();
        }
}
```

Процесс автоматизации и локальное средство запуска:

NotepadAp.java

```
package com.iba.kanclerrpa.notepad;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.notepad.task.CreateNote;

@ApModuleEntry(name = "Notepad Automation Demo")
public class NotepadAp extends ApModule {

        public TaskOutput run() throws Exception {
                return execute(getInput(), CreateNote.class).get();
        }
}
```

LocalRunner.java

```
package com.iba.kanclerrpa.notepad;

import com.iba.kanclerrpa.engine.boot.ApModuleRunner;

public class LocalRunner {

        public static void main(String[] args) {
                ApModuleRunner.localLaunch(NotepadAp.class);
        }
}
```

SAP

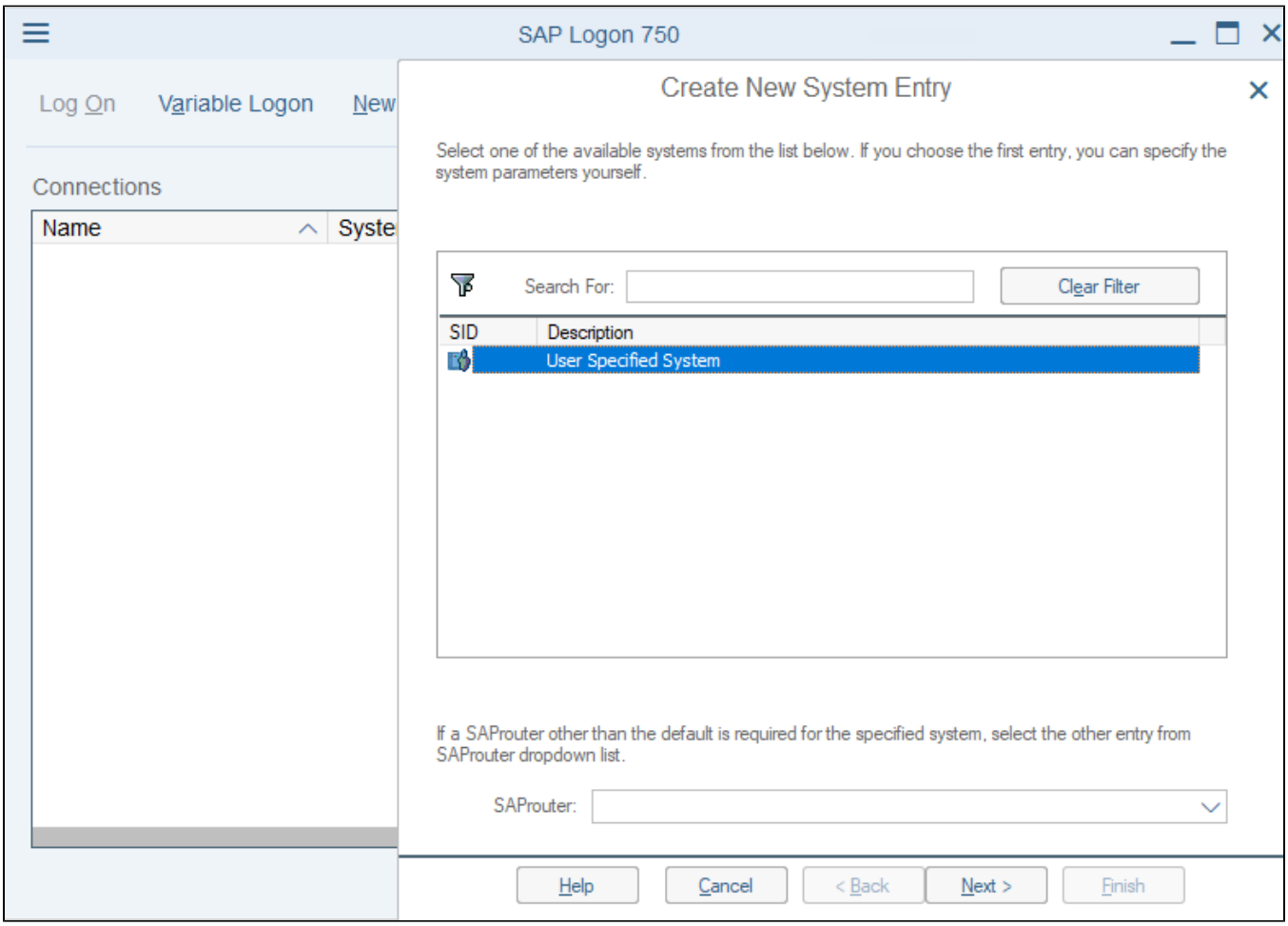
Подготовка соединения с клиентом SAP

Автоматизированный процесс должен использовать **WinappDriver** на той же машине, где запущен **NodeAgent** и настроена среда **SAP**. Главное окно SAP должно содержать все элементы без необходимости растягивания или увеличения окна.

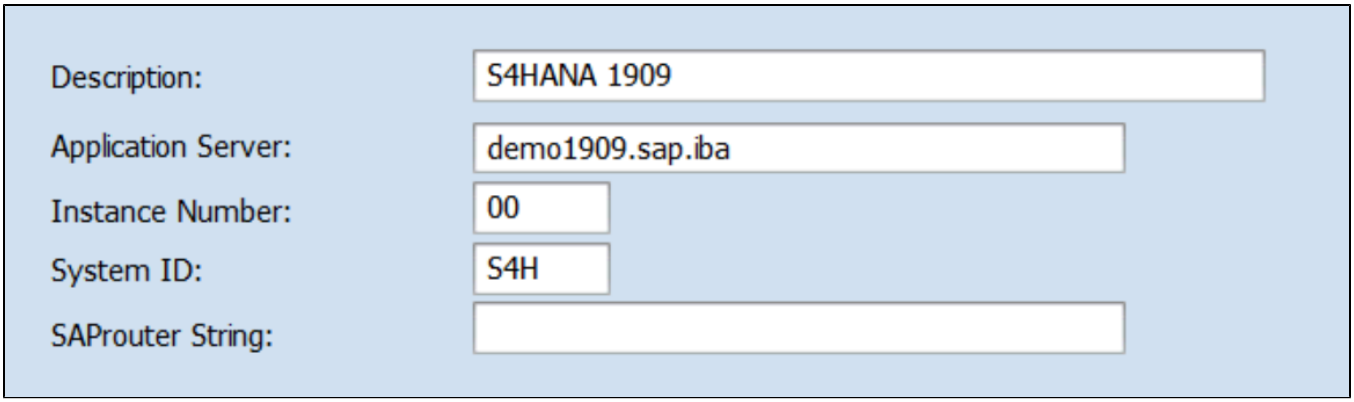
Запустите **Sap Logon** и создайте новое подключение.

> Шаги настройки будут представлены для демонстрационного примера Канцлер RPA.

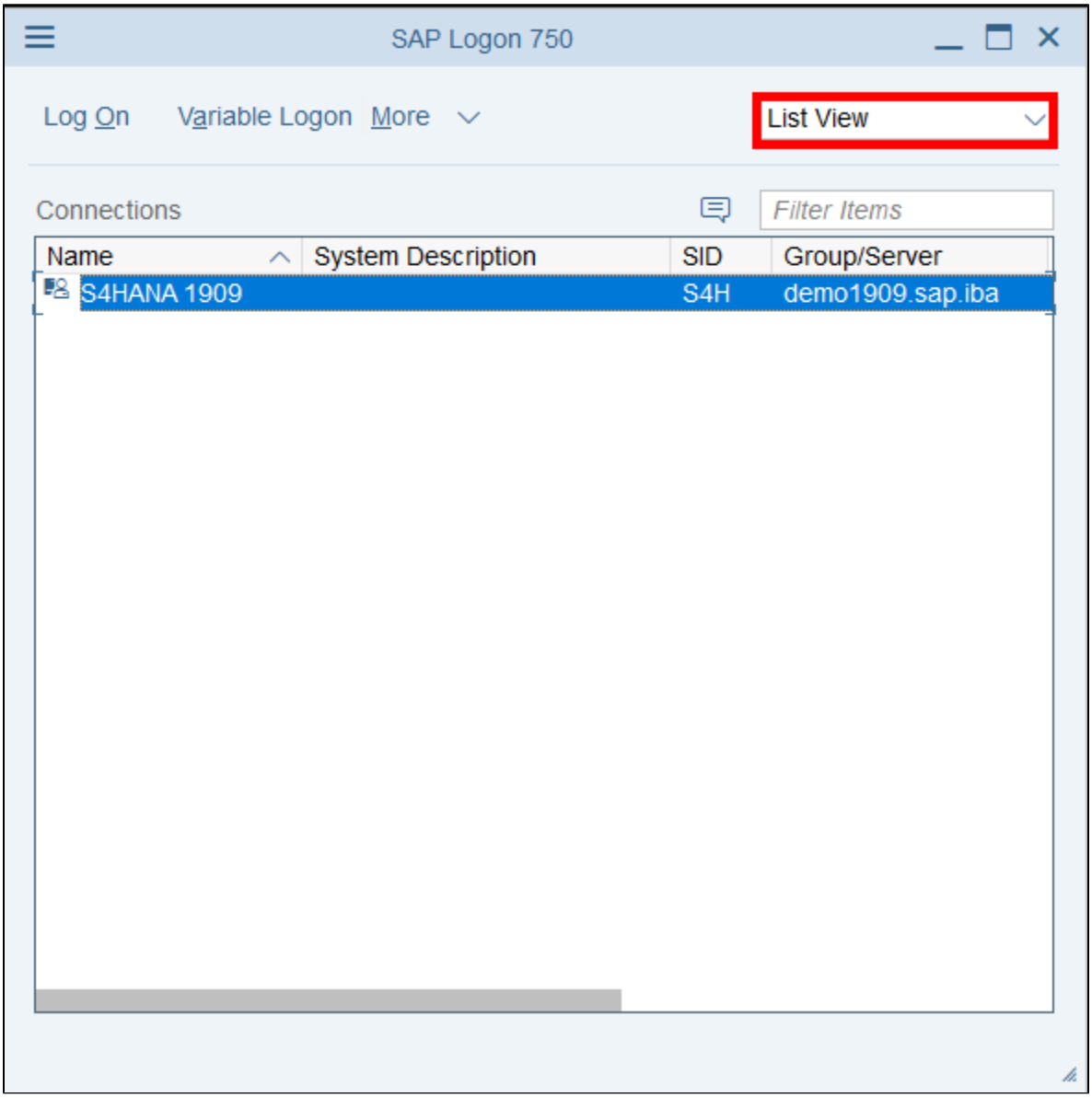
Нажмите **New Item** в верхнем меню:



Нажмите **Next** и укажите **Description**, **Application Server**, **Instance Number** и **System ID**:



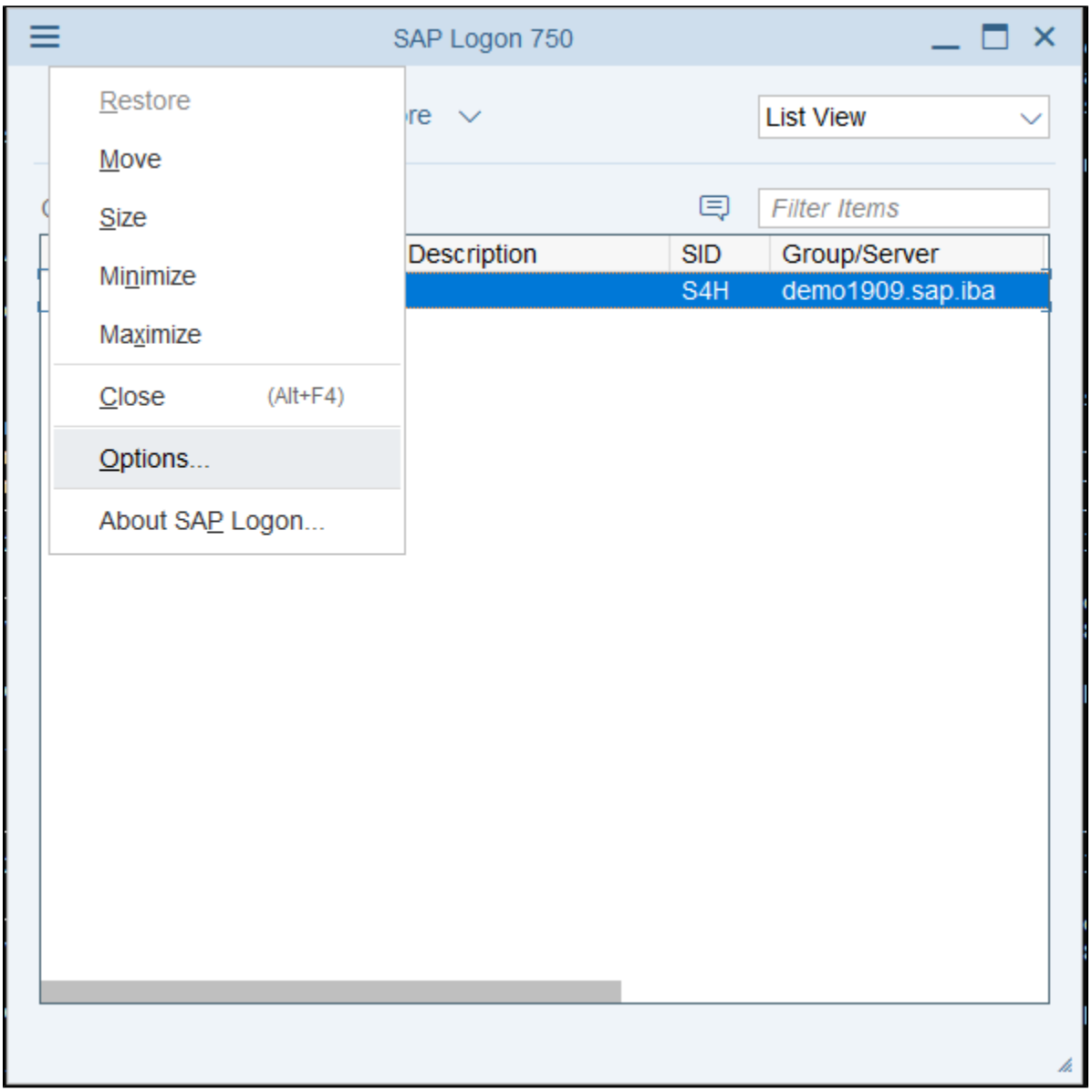
Наконец, переключитесь в режим **List View**. Это предотвратит скрытие записи подключения при случайном переключении вкладок.

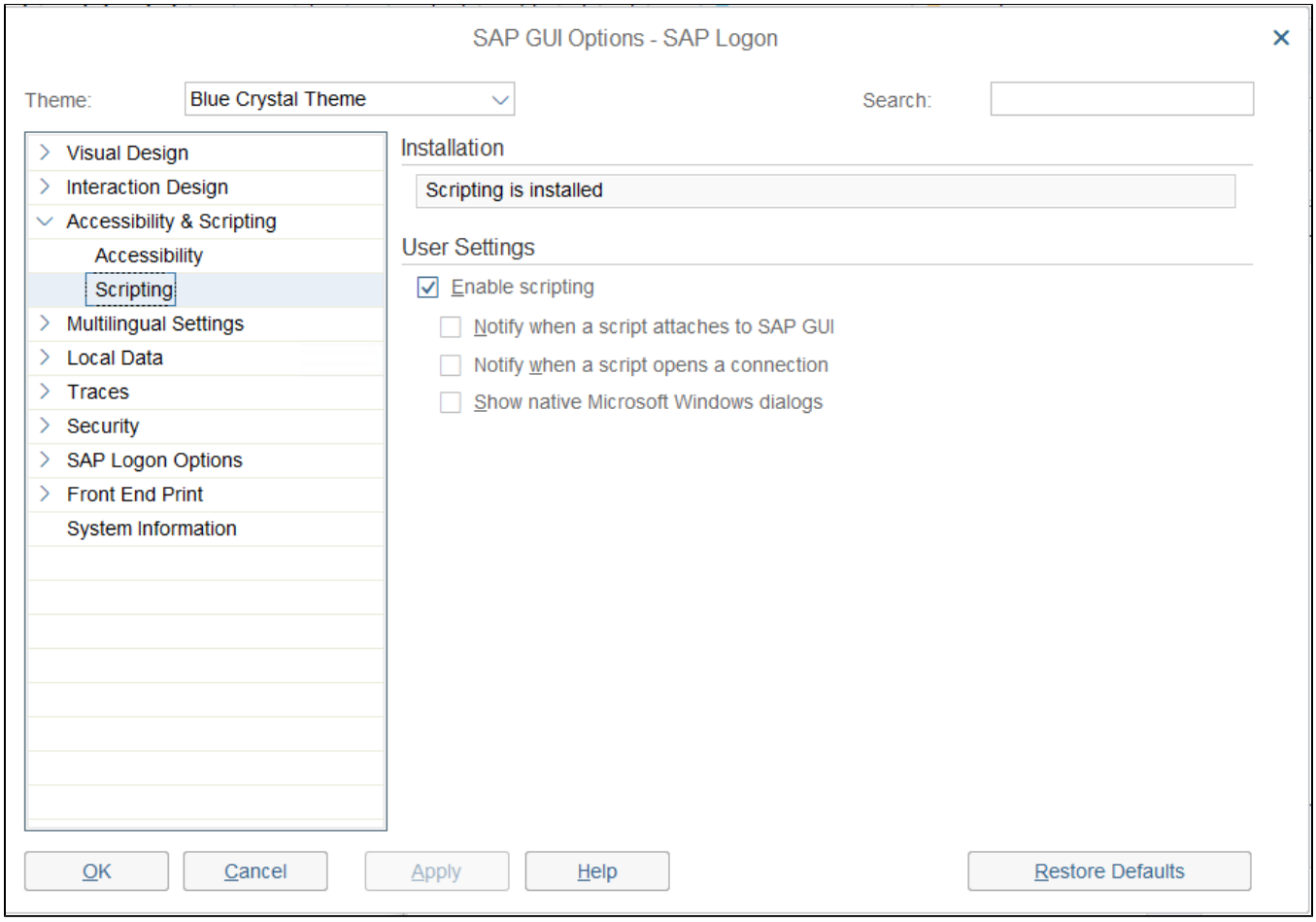


Активация scripting

Рекомендуется скачать **Scripting Tracker** с https://tracker.stschnell.de/. После выполнения этого шага необходимо иметь возможность запускать **Scripting Tracker** для **SAP Frontend** и видеть его дерево объектов.

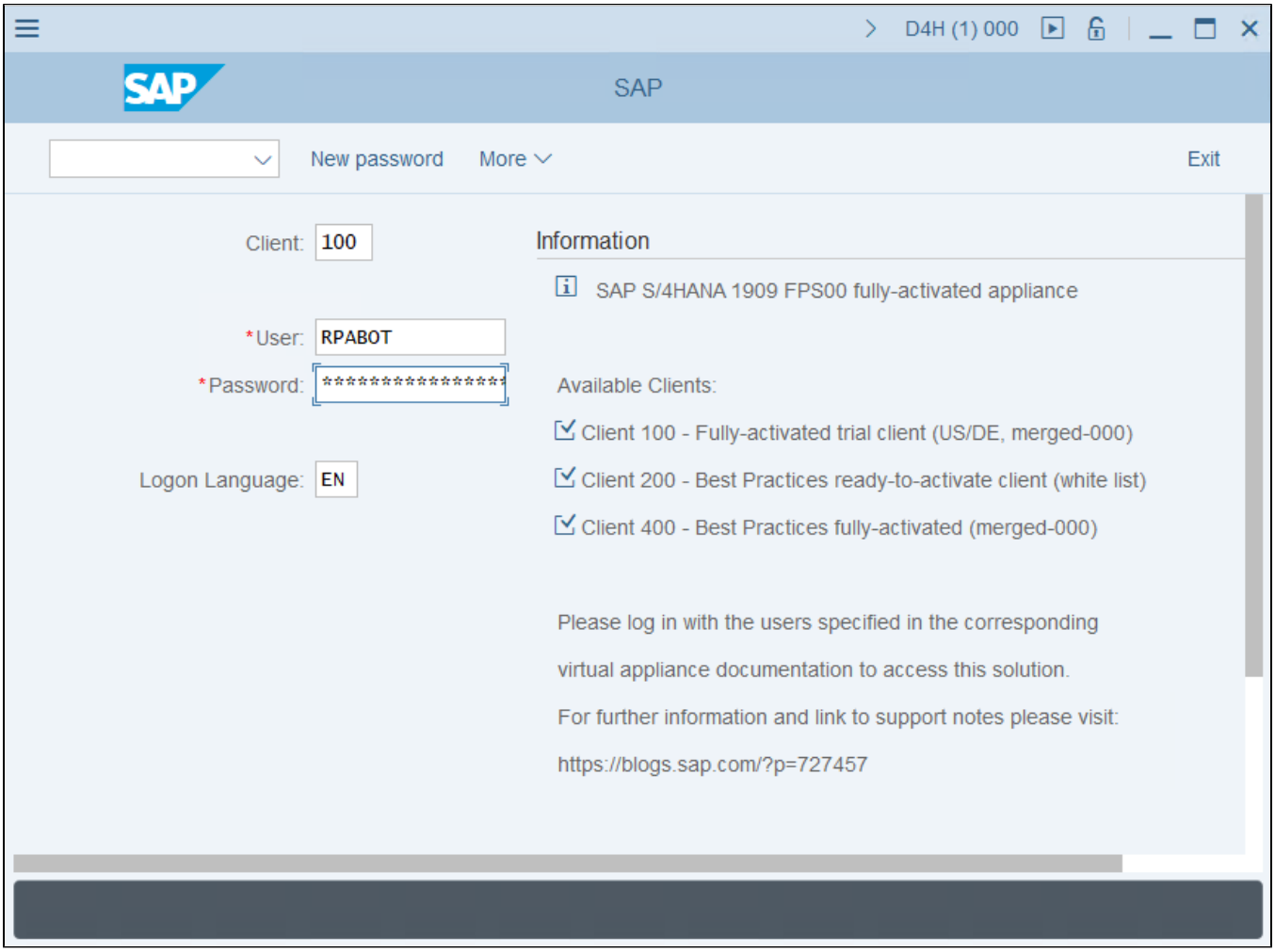
Активируйте чекбокс **Enable Scripting** на стороне клиента, отключите уведомления:



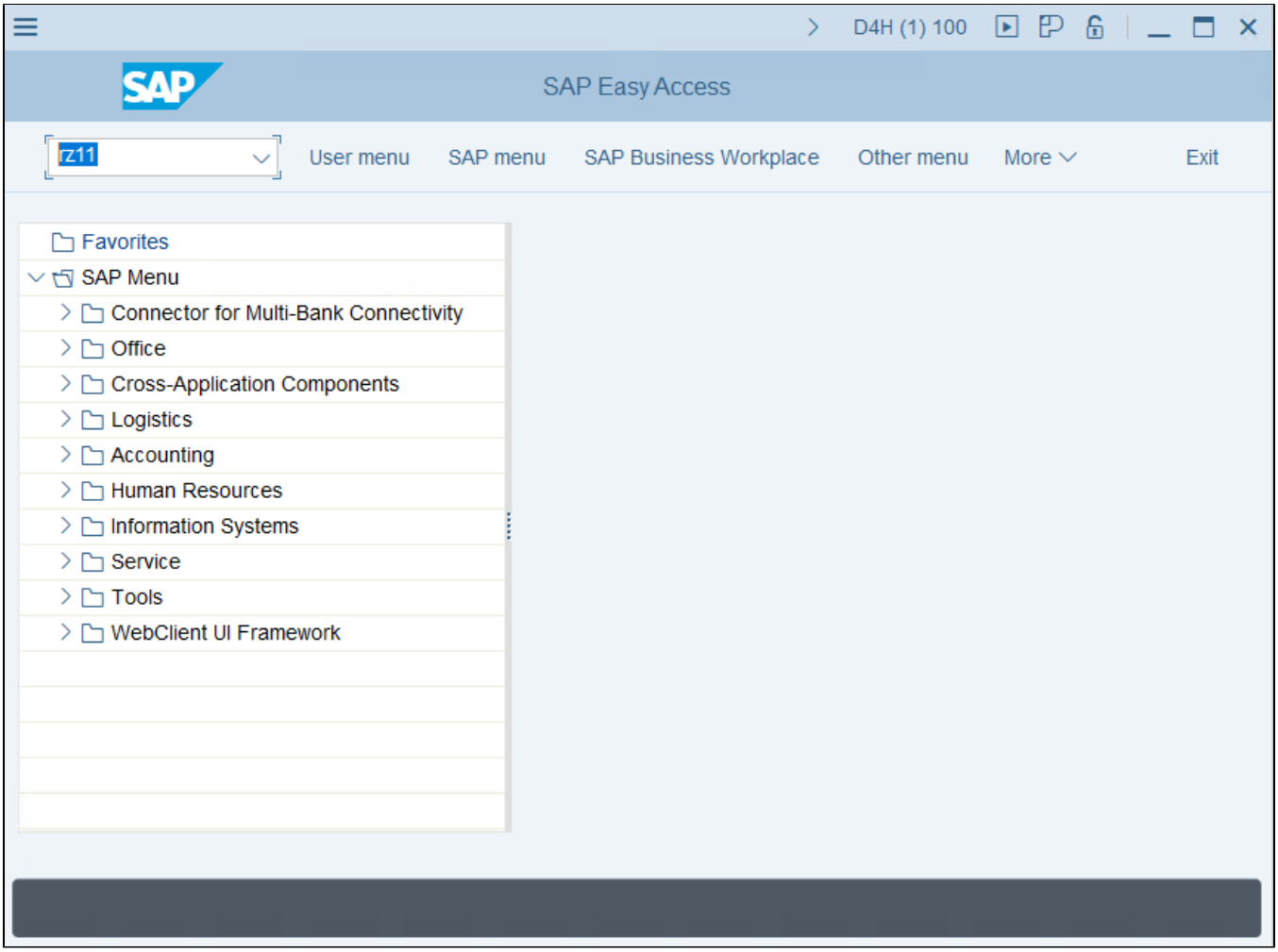


Активация scripting на сервере

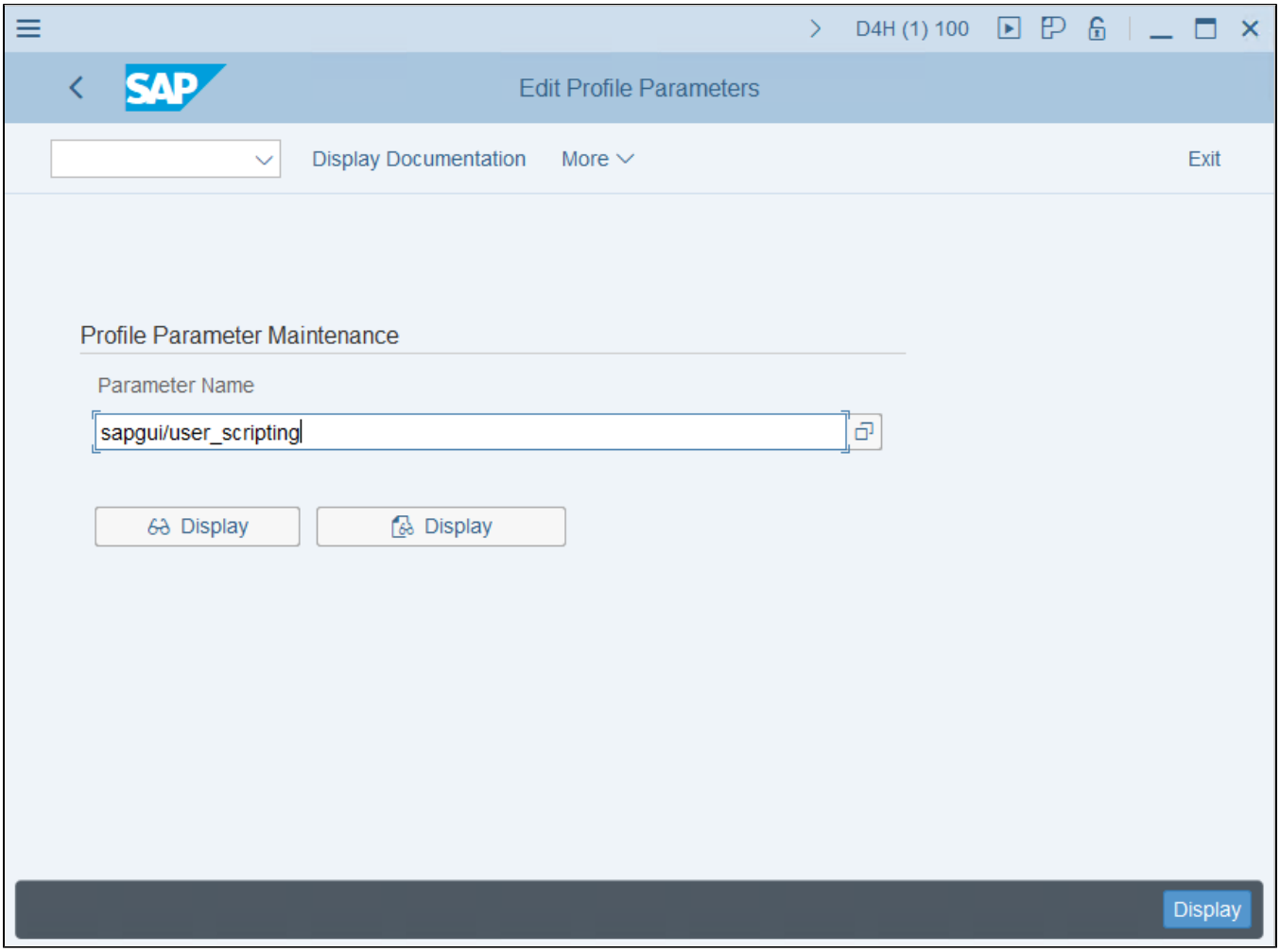
Войдите в созданное соединение, используя свои учетные данные:



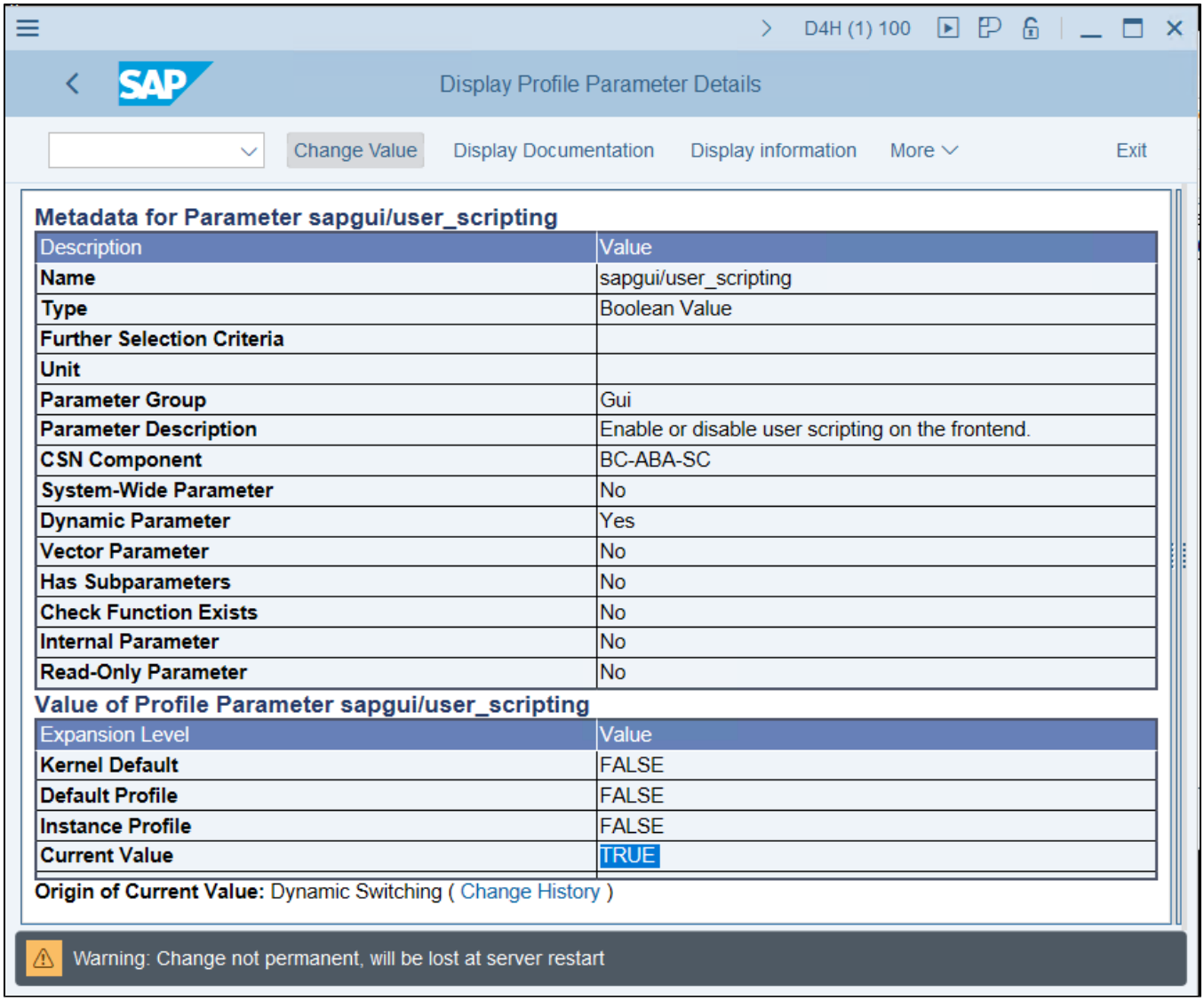
Запустите код транзакции **rz11**:



Введите `sapgui/user_scripting` в поле **Parameter Name** и нажмите Enter:



Убедитесь, что текущее значение установлено на **TRUE**, в противном случае измените его с помощью верхнего меню:



Пример

Краткий пример SAP состоит из следующих шагов:

1. Запуск сеанса SAP с определенной транзакцией с помощью sapshcut.exe.
2. Выполнение простых действий, таких как клики и ввод текста.
3. Выход из системы и завершение сеанса.

apm_run.properties файл хранит настройки, необходимые для запуска приложения.

apm_run.properties

```
sap.path=C:/Program Files (x86)/SAP/FrontEnd/SAPgui/saplogon.exe
sap.connection=S4HANA 1909
notificationChannel=SAP Notification Channel
```

vault.properties хранит учетные данные пользователя SAP, закодированные как base64

Подробнее о секретном хранилище читайте в Служба хранилища секретов.

vault.properties

```
sap.user=eyAidXNlciI6ICJSUEFCT1QiLCAicGFzc3dvcmQiOiAiUGFzc3dvcmQxIiB9
```

Класс **SapFrontendApplication** предоставляет метод для управления окнами SAP:

- `open(String... args)` - инициирует сеанс SAP Frontend, запуская исполняемый файл **sapshcut** с указанными выше настройками.

SapFrontendApplication.java

```
package com.iba.kanclerrpa.system.sap;

import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.SapDriver;
import com.iba.kanclerrpa.engine.rpa.element.SapElement;
import com.iba.kanclerrpa.system.sap.page.SapLoginPage;

public class SapFrontendApplication extends Application<SapDriver, SapElement> {

    public SapFrontendApplication(SapDriver driver) {
        super(driver);
    }

    @Override
    public SapLoginPage open(String... args) {
        getDriver().get("SAP");
        return createPage(SapLoginPage.class);
    }
}
```

Полный пример готового RPA процесса для SAP можно найти по следующей ссылке: SAP.

Автоматизация на основе захвата образов на экране

Это пример автоматизированного процесса, который проводит манипуляции с калькулятором Windows 10 с использованием драйвера захвата образов на экране.

Шаги включают в себя:

1. Открытие калькулятора через меню Пуск.
2. Развертывание окна калькулятора и переключение в научный режим.
3. Вычисление логарифма.
4. Извлечение результата с экрана с помощью OCR.
5. Закрытие окна калькулятора.

Класс приложения возвращает объект WindowsPage, который обеспечивает доступ к рабочему столу Windows 10.

WindowsScreenApplication .java

```
package com.iba.kanclerrpa.calculator;

import com.iba.kanclerrpa.calculator.page.WindowsPage;
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.ScreenDriver;
import com.iba.kanclerrpa.engine.rpa.element.ImageElement;

public class WindowsScreenApplication extends Application<ScreenDriver, ImageElement> {

        public WindowsScreenApplication(ScreenDriver driver) {
                super(driver);
        }

        @Override
        public WindowsPage open(String... args) {
                return createPage(WindowsPage.class);
        }
}
```

WindowsPage содержит только один общедоступный метод **openCalcApp()** который открывает главное окно калькулятора.

WindowsPage.java

```
package com.iba.kanclerrpa.system.rdpcalc.page;

import com.iba.kanclerrpa.engine.rpa.element.ScreenElement;
import com.iba.kanclerrpa.engine.rpa.page.ScreenPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Image;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.Keys;

@Slf4j
public class WindowsPage extends ScreenPage {

    private static final int LOAD_TIMEOUT = 3000;

    private static final int ANIMATION_TIMEOUT = 3500;

    @Image(url = "images/desktop/windows-start-1.png")
    @WithTimeout(time = 3)
    private ScreenElement start;

    public CalculatorMainPage openCalcApp() {
        launchFromStart("calculator");
        try {
            Thread.sleep(LOAD_TIMEOUT);
        } catch (InterruptedException e) {
```

```
            log.error(e.getMessage(), e);
        }
        maximizeWindow();

        return createPage(CalculatorMainPage.class);
    }

    private void launchFromStart(String appName) {
        this.start.click();
        try {
            Thread.sleep(ANIMATION_TIMEOUT);
        } catch (InterruptedException e) {
            log.error(e.getMessage(), e);
        }

        getDriver().sendKeys(appName);
        try {
            Thread.sleep(ANIMATION_TIMEOUT);
        } catch (InterruptedException e) {
            log.error(e.getMessage(), e);
        }

        getDriver().sendKeys(Keys.ENTER);
    }

    public void maximizeWindow() {
        getDriver().getInputDevices().getKeyboard().pressKey(Keys.ALT);
        getDriver().sendKeys(Keys.SPACE);
        getDriver().getInputDevices().getKeyboard().releaseKey(Keys.ALT);
        try {
            Thread.sleep(ANIMATION_TIMEOUT);
        } catch (InterruptedException e) {
            log.error(e.getMessage(), e);
        }
        getDriver().sendKeys("x");
    }
}
```

CalculatorMainPage потенциально может предоставить дескрипторы для всех кнопок и меню, отображаемых в окне калькулятора, но мы ограничили их с демонстрационными целями.

CalculatorMainPage.java

```
package com.iba.kanclerrpa.system.rdpcalc.page;

import com.iba.kanclerrpa.engine.rpa.element.ScreenElement;
import com.iba.kanclerrpa.engine.rpa.page.ScreenPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Image;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.Keys;
import org.sikuli.script.Region;

@Slf4j
public class CalculatorMainPage extends ScreenPage {

    private static final double CALC_DISPLAY_HEIGHT_RATIO = 0.14;

    private static final double CALC_DISPLAY_WIDTH_RATIO = 0.85;

    private static final double CALC_DISPLAY_Y_RATIO = 0.12;

    private static final int HIGHLIGHT_SECONDS = 2;

    @Image(url = "images/calculator/navigation-btn-1.png")
    @WithTimeout(time = 3)
    private ScreenElement navigation;

    @Image(url = "images/calculator/four-btn-1.png")
```

```
    @WithTimeout(time = 3)
    private ScreenElement fourBtn;

    @Image(url = "images/calculator/six-btn-1.png")
    @WithTimeout(time = 3)
    private ScreenElement sixBtn;

    @Image(url = "images/calculator/eight-btn-1.png")
    @WithTimeout(time = 3)
    private ScreenElement eightBtn;

    @Image(url = "images/calculator/log-btn-1.png")
    private ScreenElement logBtn;

    public void four() {
        this.fourBtn.click();
    }

    public void six() {
        this.sixBtn.click();
    }

    public void eight() {
        this.eightBtn.click();
    }

    public void logarithm() {
        this.logBtn.click();
    }

    public CalculatorMenuPage openMenu() {
        this.navigation.click();
        return createPage(CalculatorMenuPage.class);
    }

    public String getDisplayText() {
        int h = (int) (CALC_DISPLAY_HEIGHT_RATIO * getDriver().getScreen().h);
        int w = (int) (CALC_DISPLAY_WIDTH_RATIO * getDriver().getScreen().w);
        int y = (int) (CALC_DISPLAY_Y_RATIO * getDriver().getScreen().h);
        int x = 5;

        log.debug("Reading region: x={} y={} w={} h={}", x, y, w, h);

        return getRegionText(x, y, w, h);
    }

    private String getRegionText(int x, int y, int w, int h) {
        Region regResult = getDriver().getScreen().newRegion(x, y, w, h);
        regResult.highlight(HIGHLIGHT_SECONDS);

        return regResult.text();
    }

    public void exit() {
        getDriver().getInputDevices().getKeyboard().pressKey(Keys.ALT);
        getDriver().getInputDevices().getKeyboard().sendKeys(Keys.F4);
        getDriver().getInputDevices().getKeyboard().releaseKey(Keys.ALT);
    }
}
```

getDisplayText() заслуживает особого упоминания, поскольку он использует OCR для чтения определенной области на экране. Эта область в основном представляет собой прямоугольник, определяемый верхней левой вершиной (x, y), высотой (h) и шириной (w).

Полный пример готового RPA процесса для Калькулятора можно найти по следующей ссылке: Image-based Calculator Demo.

Создание скриншотов

Существуют способы делать скриншоты в **любом** драйвере:

- ```
  byte[] bytes = driver.getScreenshotAsBytes();
  ```
- ```
  File file = driver.getScreenshotAsFile();
  ```
- ```
  String base64String = driver.getScreenshotAsBase64();
  ```

Пример

TakeScreenshot.java

```
@ApTaskEntry(name = "Take screenshot")
public class TakeScreenshot extends ApTask {

        @Driver(DriverParams.Type.DESKTOP)
        private DesktopDriver desktopDriver;

        @Override
        public void execute() throws IOException {
                byte[] bytes = desktopDriver.getScreenshotAsBytes();
                FileUtils.writeByteArrayToFile(new File("C:\\work\\screenshot.png"), bytes);
        }
}
```
```

Загрузка файлов

В этом примере показано, как автоматизировать загрузку файлов в веб-приложении, а также использовать два драйвера в одной задаче.

Класс приложения открывает веб-браузер и возвращает объект MainPage:

OpenFileApplication

```
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.DesktopDriver;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import org.example.page.OpenFilePage;

public class OpenFileApplication extends Application<DesktopDriver, DesktopElement> {

    public OpenFileApplication(DesktopDriver driver) {
        super(driver);
    }

    @Override
    public OpenFilePage open(String... args) {
        return createPage(OpenFilePage.class);
    }
}
```

ViljamisApplication.java

```
import com.iba.kanclerrpa.engine.rpa.Application;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.element.BrowserElement;
import org.example.page.MainPage;

public class ViljamisApplication extends Application<BrowserDriver, BrowserElement> {

    public ViljamisApplication(BrowserDriver driver) {
        super(driver);
    }

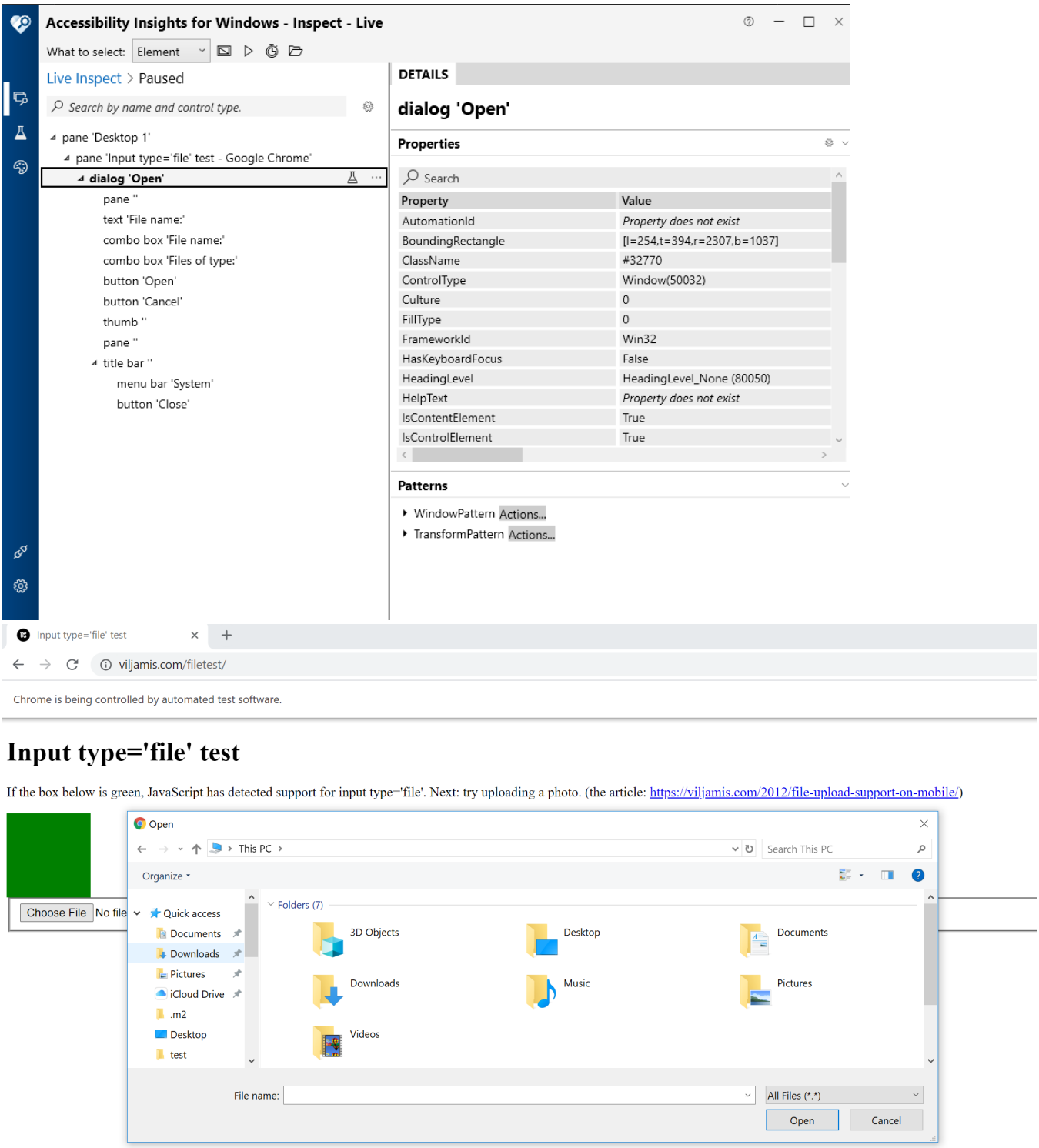
    @Override
    public MainPage open(String... args) {
        String gmailUrl = args[0];
        getDriver().get(gmailUrl);
        return createPage(MainPage.class);
    }
}
```

Класс MainPage содержит метод openFileWindow(DesktopDriver). Этот метод открывает окно **Open**, поэтому он должен возвращать объект класса **OpenFilePage**. Однако **OpenFilePage** ссылается на другое приложение, которое отвечает за его создание. Поэтому метод получает в качестве параметра драйвер для работы с открываемым им приложением, затем выполняет действия по открытию окна **Open**, создает необходимое приложение и возвращает нужную страницу, вызывая соответствующий метод из приложения.

MainPage.java

```
import com.iba.kanclerrpa.engine.rpa.driver.DesktopDriver;
import com.iba.kanclerrpa.engine.rpa.page.WebPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.Wait;
import org.example.application.OpenFileApplication;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.FindBy;
```

```
public class MainPage extends WebPage {

    @FindBy(xpath = "//input[@name='image']")
    @Wait(waitFunc = Wait.WaitFunc.VISIBLE)
    private WebElement chooseFileInput;

    @FindBy(xpath = "//input[@type='submit']")
    @Wait(waitFunc = Wait.WaitFunc.VISIBLE)
    private WebElement uploadInput;

    public OpenFilePage openFileWindow(DesktopDriver desktopDriver) {
        Actions builder = new Actions(getDriver());
        builder.moveToElement(chooseFileInput).click().perform();

        OpenFileApplication openFileApplication = new OpenFileApplication(desktopDriver);
        return openFileApplication.open();
    }

    public void upload() {
        uploadInput.click();
    }
}
```

Структурно окно открытия файла представляет собой не отдельное окно, а элемент окна браузера.



Поэтому для доступа к элементам окна **Open** нам нужно переключить **DesktopDriver** на главное окно браузера.

Класс **OpenFilePage** хранит строку с именем окна и переключается в методе **init()**. При создании объекта этот метод выполняется и переключает драйвер в окно браузера, предоставляя доступ к его элементам. Метод **ChooseFile** предоставляет путь к файлу и нажимает кнопку **Open**.

OpenFilePage.java

```
import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.rpa.element.DesktopElement;
import com.iba.kanclerrpa.engine.rpa.page.DesktopPage;
import com.iba.kanclerrpa.engine.rpa.po.annotation.WithTimeout;
import org.openqa.selenium.support.FindBy;

public class OpenFilePage extends DesktopPage {

    @FindBy(className = "Edit", name = "File name:")
    @WithTimeout(time = 3)
    private DesktopElement fileNameInput;

    @FindBy(className = "Button", name = "Open")
    @WithTimeout(time = 3)
    private DesktopElement openBtn;

    @AfterInit
    public void init() {
        String title = "Input type='file' test - Google Chrome";
        getDriver().waitAndSwitchToWindow(title, 5); // switch to the root Chrome window first
        getDriver().waitAndSwitchToWindow("Open", 5);
    }

    public void chooseFile(String fullFilePath) {
        fileNameInput.sendKeys(fullFilePath);
        openBtn.click();
    }
}
```

UploadFile — класс, содержащий основной алгоритм выполнения задачи:

UploadFile.java

```
import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Driver;
import com.iba.kanclerrpa.engine.annotation.DriverParameter;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.rpa.driver.BrowserDriver;
import com.iba.kanclerrpa.engine.rpa.driver.DesktopDriver;
import com.iba.kanclerrpa.engine.rpa.driver.DriverParams;
import org.example.application.ViljamisApplication;
import org.example.page.MainPage;
import org.example.page.OpenFilePage;

@ApTaskEntry(name = "Upload File")
public class UploadFile extends ApTask {

    @Driver(value = DriverParams.Type.BROWSER, param = {
            @DriverParameter(key = DriverParams.Browser.SELENIUM_NODE_CAPABILITIES, initializerName =
DriverParams.BrowserCapabilities.CHROME) })
    private BrowserDriver browserDriver;

    @Driver(DriverParams.Type.DESKTOP)
    private DesktopDriver desktopDriver;

    private String viljamisUrl;

    private String filePath;

    private String fileName;

    @AfterInit
    public void init() {
        viljamisUrl = getConfigurationService().get("viljamis.app.url", "https://viljamis.com/filetest/");
        filePath = getConfigurationService().get("viljamis.file.path", System.getProperty("user.home"));
        fileName = getConfigurationService().get("viljamis.file.name", "AboutNotepad.txt");
    }
```

```
    @Override
    public void execute() {
        ViljamisApplication viljamisApplication = new ViljamisApplication(browserDriver);
        MainPage mainPage = viljamisApplication.open(viljamisUrl);

        OpenFilePage openFilePage = mainPage.openFileWindow(desktopDriver);

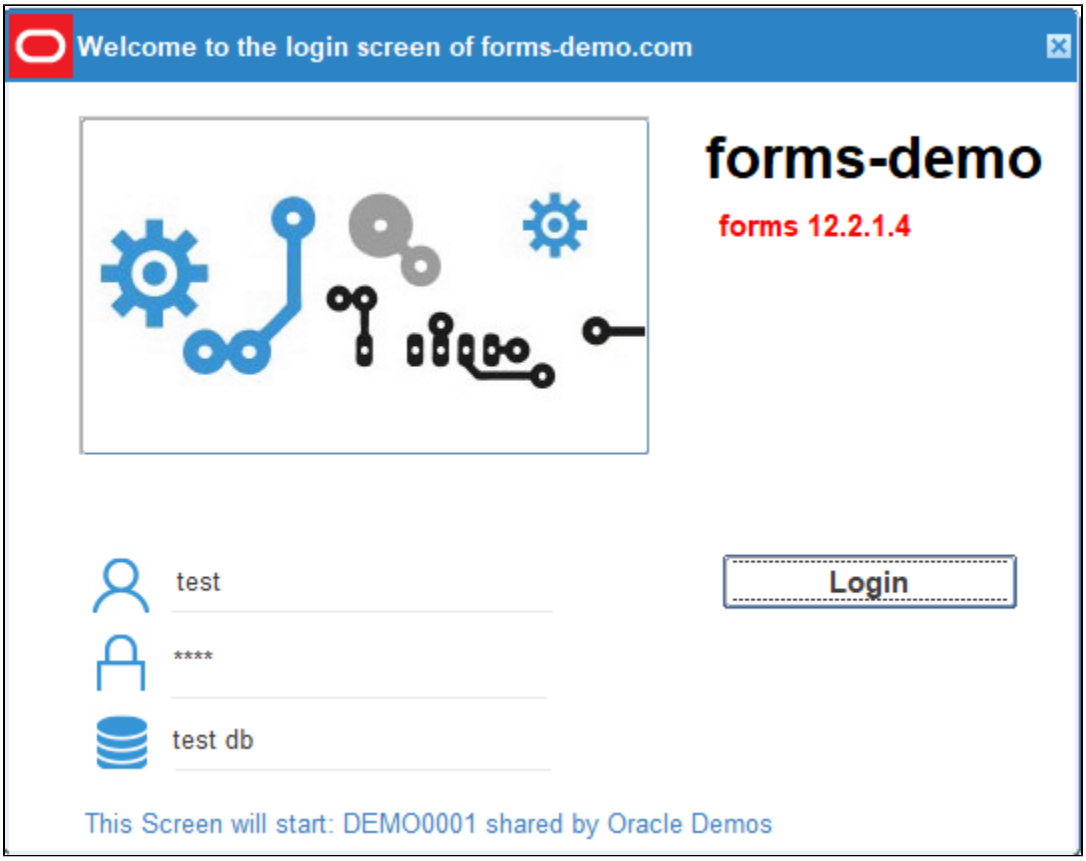
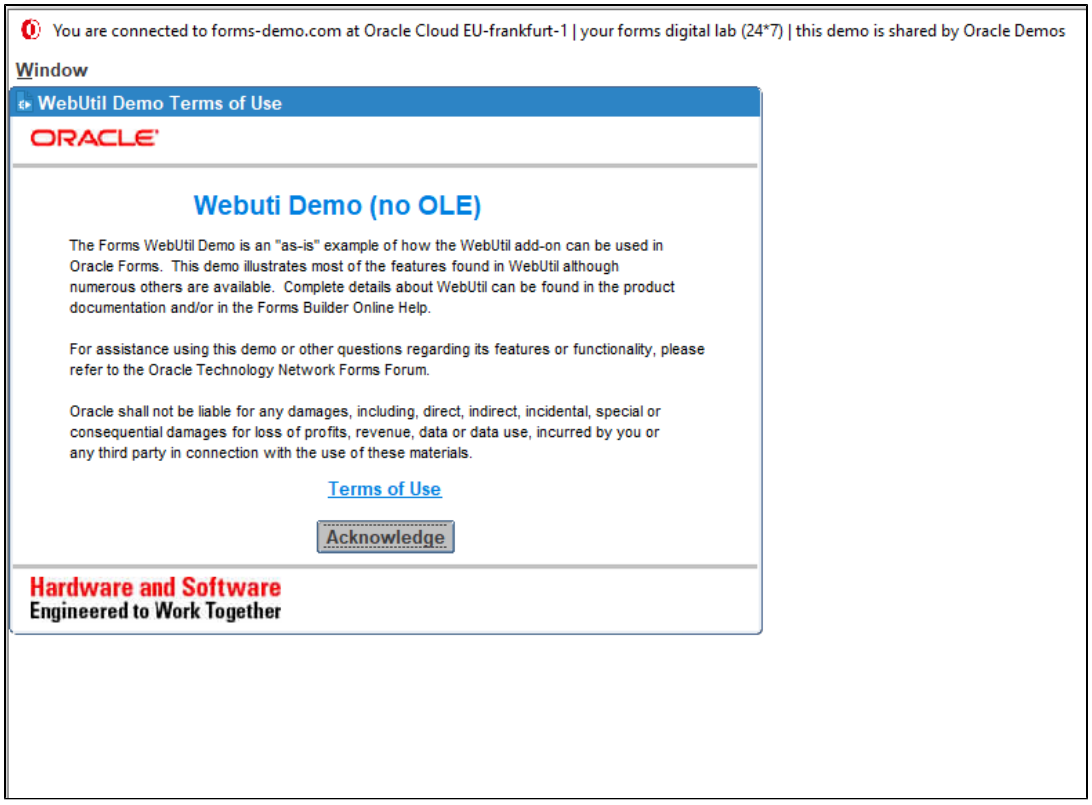
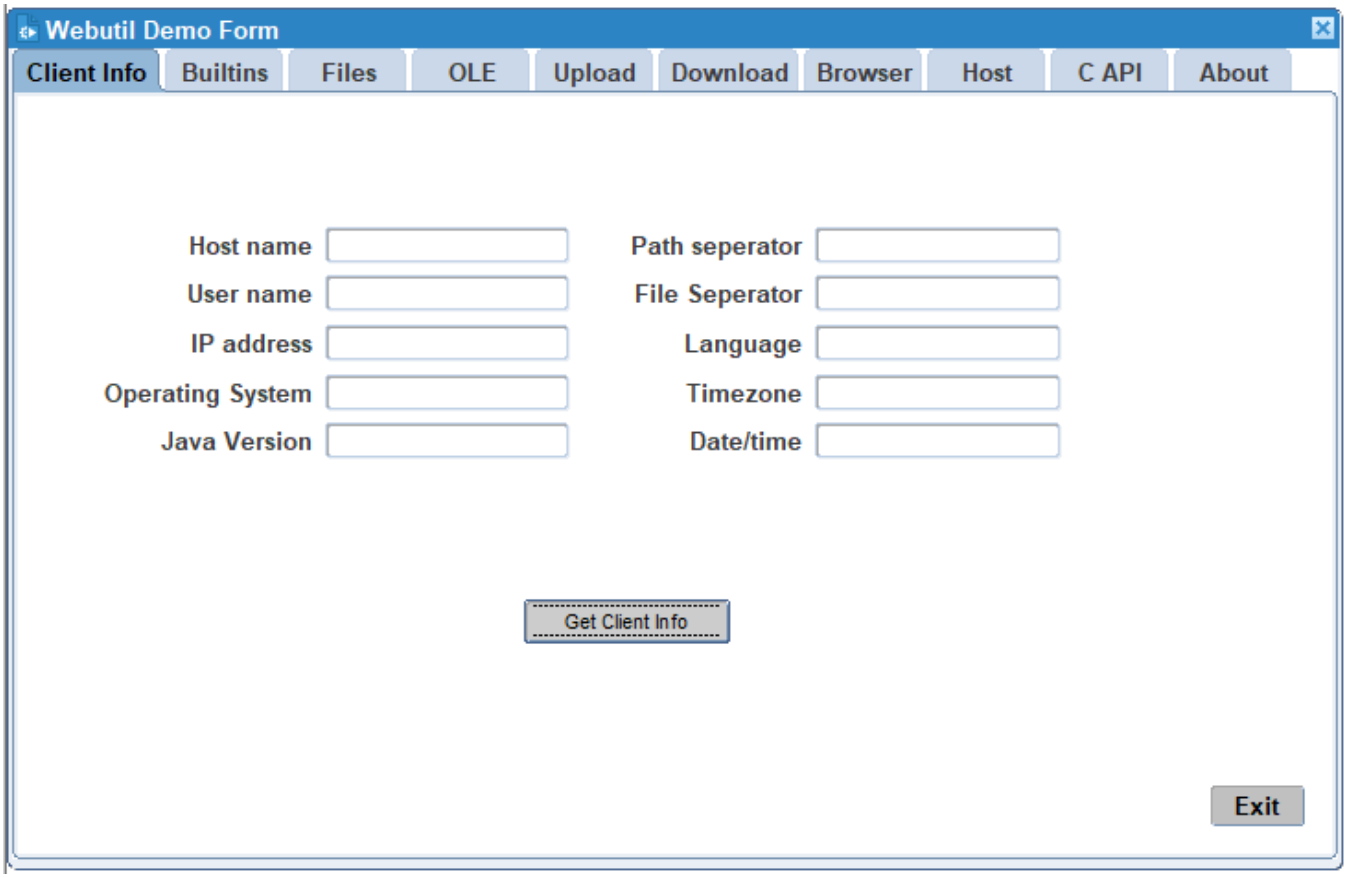
        openFilePage.chooseFile(filePath + "\\" + fileName);
        mainPage.upload();
    }
}
```

Автоматизированый процесс и локальное средство запуска:

UploadFileAp .java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

import org.example.tasks.UploadFile;

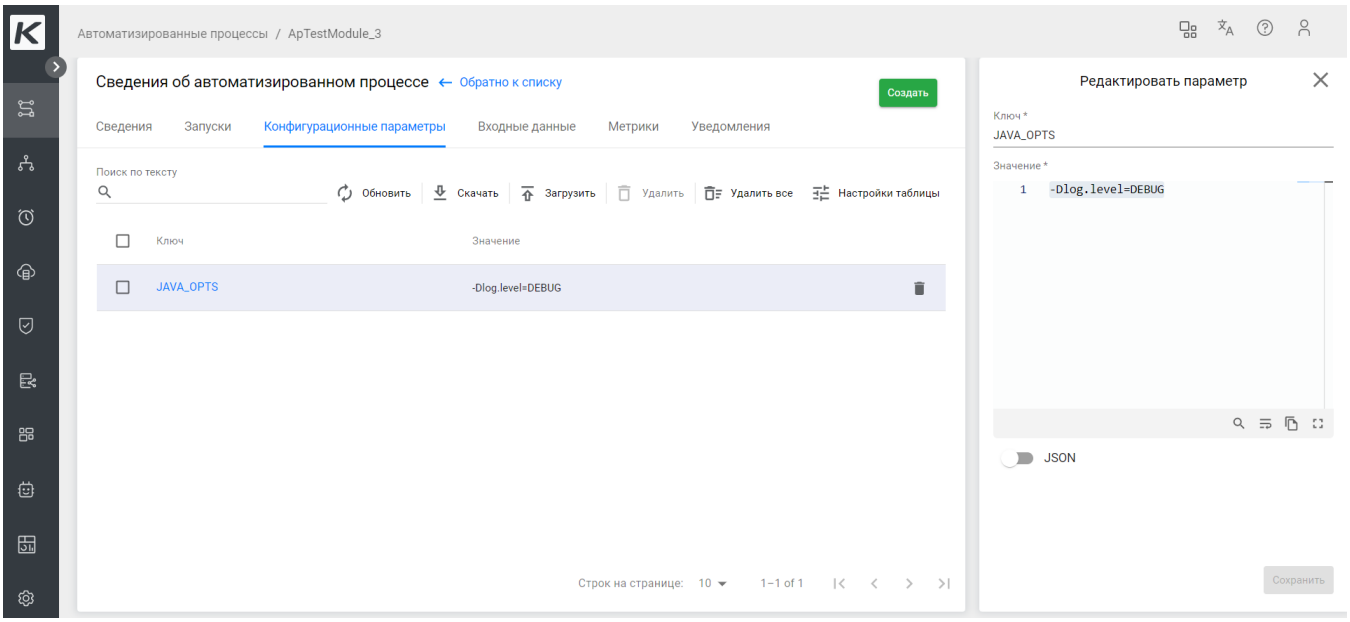
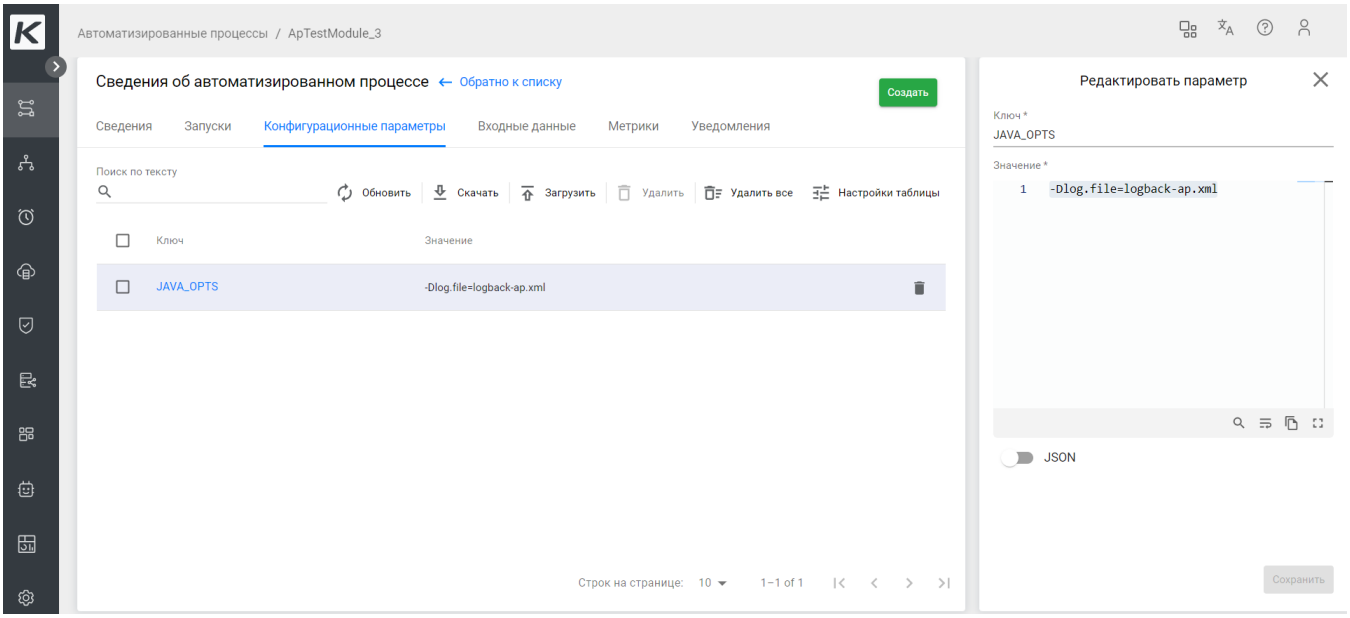
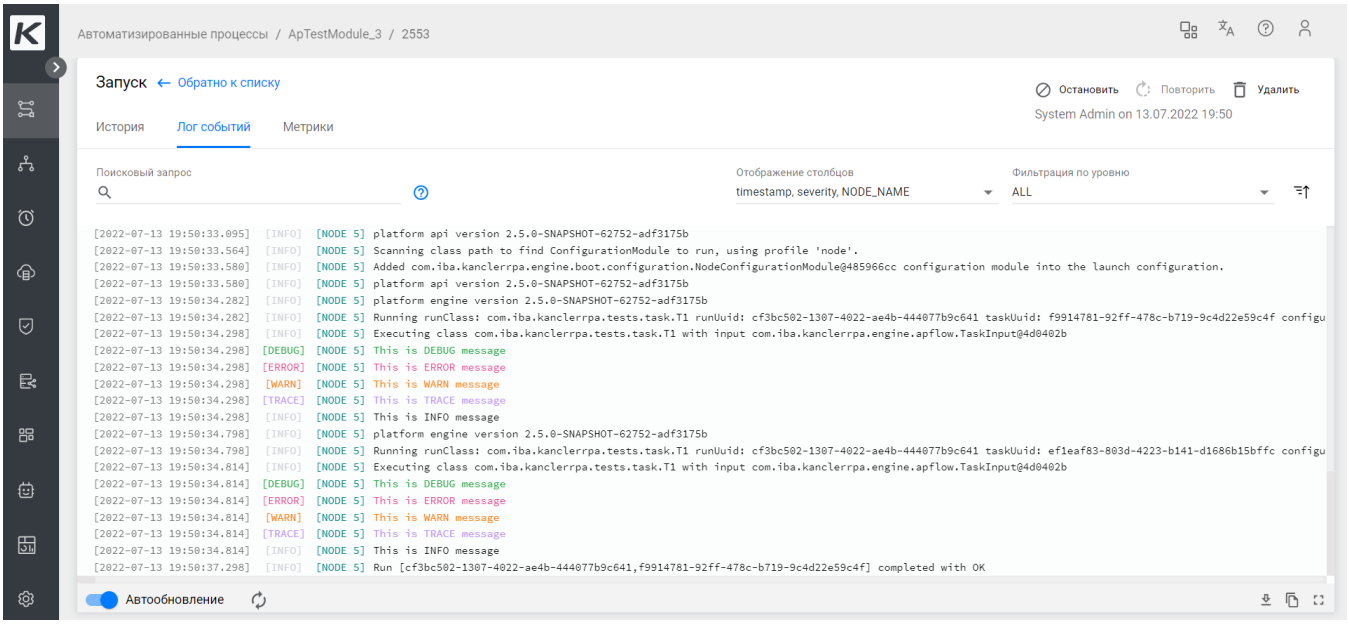
@Slf4j
@ApModuleEntry(name = "Upload file demo", description = "This process automates a lot of work...")
public class UploadFileAp extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), UploadFile.class).get();
    }
}
```

LocalRunner.java

```
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(UploadFileAp.class, new DevelopmentConfigurationModule(args));
        //ApModuleRunner.localLaunch(CmdDemoAp.class);
    }
}
```

Oracle формы

Введение

Этот пример демонстрирует автоматизацию приложения Oracle Forms. Пример реализации автоматизации Oracle формы можно найти в вашем локальном Nexus.

Предпосылки

Параметры конфигурации:

Ключ	Значение по умолчанию	Обязательность	Описание
app.path	app/start-oracle-forms-app.bat	нет	Полный путь к исполняемому файлу Oracle Forms: start-oracle-forms-app.bat

Шаги

Бот открывает ссылку на Oracle Forms demo в Internet Explorer и через некоторое время появляется окно входа:



Бот нажимает кнопку **Login**, оставляя остальные поля без изменений:



Далее на экране условий использования бот нажимает кнопку **Acknowledge***:*



Затем на экране информации о клиенте бот нажимает **Get Client Info**, считывает данные из полей, печатает их в журнал и, наконец, нажимает **Exit**.

Управление логами

Введение

Канцлер RPA предоставляет инфраструктуру ведения журналов, поддерживаемую java-библиотекой logback. В этом разделе мы объясним, как разработчик может использовать компоненты системы ведения логов.

Определение регистратора

Все системные события логируются с использованием интерфейса Logger, который перенаправляет их дальше в соответствии с конфигурацией.

Чтобы вывести что-то в логи разработчик должен объявить ссылку на логи и вызвать соответветствующий метод.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

class LocalRunner {
    private static final Logger log = LoggerFactory.getLogger(LocalRunner.class);

    public static void main(String[] args) {
        log.debug("This message will be logged");
        log.debug("This message includes arguments: {} and {}", "Europe", "Asia");
    }
}
```


Объявление регистратора можно заменить аннотацией @Slf4j

При локальном запуске приведенный выше код выводит на консоль:

```
18:03:37.511 [main] DEBUG LocalRunner - This message will be logged
18:03:37.515 [main] DEBUG LocalRunner - This message includes arguments: Europe and Asia
```

Профили регистрации

В зависимости от того, какая конфигурация используется, существуют разные способы настройки профилей ведения журнала:

- StandaloneConfigurationModule - нет никаких ограничений на использование любого файла конфигурации журнала, потому что журналы не должны храниться в сервере управления.
- DevelopmentConfigurationModule - логи хранятся на сервере управления и loggback должен расширить конфигурацию платформы.

Настройка Logback с помощью StandaloneConfigurationModule

В системе StandaloneConfiguration используется файл logback-rpaplatform-standalone.xml для конфигурации журнала. Вот вариант по умолчанию:

```
<!-- Logback configuration file for AP  -->
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{ISO8601} %-5level %logger{36} [%thread] [%X{RUN_UUID}] [%X{TASK_UUID}] - %msg%n<
/pattern>
        </encoder>
    </appender>
    <root level="${log.level:-INFO}">
        <appender-ref ref="CONSOLE" />
    </root>
    <logger name="com.iba" level="${log.level:-INFO}" additivity="false">
        <appender-ref ref="CONSOLE" />
    </logger>
</configuration>
```

Чтобы переопределить файл по умолчанию, поместите файл в ресурсы вашего проекта.

Шаблон

Шаблон вывода по умолчанию настроен как `%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n`

Это можно перевести как:

- `%d{HH:mm:ss.SSS}` текущая дата в формате HH:mm:ss.SSS
- `[%thread]` название темы в квадратных скобках
- `%-5level` log уровень (дополняется до 5 символов)
- `%logger{36}` имя регистратора (макс. 36 символов)
- `%msg` зарегистрированное сообщение
- `%n` специальный символ новой строки

Именно это показал логгер в нашем примере. Шаблон вывода, а также другие настройки регистратора можно изменить в файле logback.xml. Этот файл загружается из папки ресурсов при запуске Java-приложения. Помимо уже знакомого шаблона, в файле появились две новые функции: appender и level.

Appender

Appender обрабатывает событие, отправленное в логгер.

CONSOLE appender (с сопутствующим классом *ch.qos.logback.core.ConsoleAppender*) перенаправляет наши журналы на стандартный вывод, это консоль.
Logback также позволяет использовать другие выходные данные, например *FileAppender* и т. д. Подробнее о них можно прочитать в официальной документации.

Level

Level это то, что составляет иерархию событий. Это, в свою очередь, позволяет отфильтровать события по их серьезности. Logback на настоящий момент предоставляет пять уровней логирования: `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`.

Теперь рассмотрим пример:

```
log.trace("This is the trace message");
```

При запуске этого примера вывод ничего не показывает. Это связано с тем, что приоритет уровня трассировки ниже, чем уровень отладки, настроенный в logback.xml.

Давайте изменим корневой уровень в logback.xml следующим образом: `<root level="DEBUG">` и вернемся к примеру.

На это раз лог здесь есть

```
18:06:19.292 [main] TRACE LocalRunner - This is the trace message
```

Более подробное представление об уровнях серьезности логов можно найти в официальной документации.

Configuring Logback с DevelopmentConfigurationModule

В DevelopmentConfigurationModule все логи хранятся на сервере управления, и изменение конфигурации логов платформы может привести к проблемам с логами, вы можете только добавлять новые логгеры и изменять уровни логирования.

Измение уровня логирования

Уровень лога по умолчанию для логгера ROOT — INFO. Это означает, что только ERROR, WARN, INFO будут регистрироваться в консоли или в логе процесса автоматизации. Вы можете изменить уровень с помощью системного свойства java в процессе автоматизации:

logback.xml

```
-Dlog.level=DEBUG
```

На сервере управления необходимо добавить/обновить параметр **JAVA_OPTS** вв модуле **Настройки сервера** или во вкладке **Конфигурационные параметры** в модуле **Автоматизированные процессы** или во вкладке **Конфигурационные параметры** в модуле **Узлы:**



Определить уровни логгера

Вы можете определить уровень логгера для каждого уровня. Для этого вам нужно добавить файл ресурсов в ваш проект процесса автоматизации, используйте следующий файл в качестве шаблона:

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<included>
    <logger name="org.apache.http" level="INFO" />
    <logger name="com.iba.kanclerrpa.tests" level="ALL" />
</included>
```

Добавьте определенную конфигурацию в logback configuration , добавив следующую системную переменную java:

logback.xml

```
-Dlog.file=logback-ap.xml
```

В Сервер управления необходимо добавить/обновить параметр **JAVA_OPTS** в модуле **Настройки сервера** или во владке **Конфигурационные параметры** в модуле **Автоматизированные процессы**:



Где log.file относится к файлу ресурсов из вашей коллекции.

Получение логов

Обычно разработчик работает автономно - запускает код в IDE и сразу же видит результат в консоли, но в сервере управления все немного иначе. Сервер управления хранит свои логи в Elastic Search, то же самое делает Агент узла, на котором работает автоматизированный процесс. Самый простой способ проверить логи автоматизированного процесса —открыть вкладку **Лог событий** запуска. См. Лог событий запуска автоматизированного процесса.



Рекомендации

В этом разделе описаны методы, которые считаются стандартным способом работы и позволяют поддерживать качество разработанного кода.

Избегайте использования Thread.sleep

Вам всегда следует избегать использования метода Thread.sleep() во время реализации RPA, поскольку он останавливает выполнение потока в течение указанного времени.

Вместо этого вы должны использовать объект ожидания драйвера, когда вам нужно дождаться появления на экране какого-либо элемента UI. Ожидание драйвера — это гибкий инструмент для управления ожиданием, игнорированием исключений, контролем интервала опроса и т.д.

Пример

Вместо:

Bad practice

```
driver.get("...");
Thread.sleep(2000);
driver.find(By.id("id1")).click();
Thread.sleep(2000);
```

Вам следует сделать:

Best practice

```
driver.get("...");
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("id1")));
driver.find(By.id("id1")).click();
```

Установите для драйвера по умолчанию ожидание 0 секунд

Мы рекомендуем всегда устанавливать неявное ожидание драйвера на 0 секунд. В документации Явные и неявные ожидания че тко указано, что:

> **Warning**
>
> Не смешивайте неявные и явные ожидания. Это может привести к непредсказуемому по времени ожиданию. Например, установка неявного ожидания в 10 секунд и явного ожидания в 15 секунд может привести к тайм-ауту через 20 секунд.

Часть проблемы заключается в том, что неявные ожидания часто (но не всегда!) реализуются на «удаленной» стороне системы драйверов. Это означает, что они "встроены" в chromedriver.exe. Явные ожидания реализованы исключительно в привязках «локального» языка.

Best practice

```
driver.manage().timeouts().implicitlyWait(0, TimeUnit.SECONDS)
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("id1")));
```

Используйте шаблон проектирования Page Object

В качестве стандартной практики разработки рекомендуется отделить бизнес-логику от логики страницы какого-либо UI-приложения. В случае RPA отлично подходит шаблон проектирования Page Object. Подробная информация доступна на странице Шаблон проектирования Page Object.

Утилиты RPA

Полезная библиотека, доступная в Канцлер RPA Nexus:

```
<dependency>
  <groupId>com.iba</groupId>
  <artifactId>kanclerrpa-utils</artifactId>
  <version>${version}</version>
</dependency>
```

Библиотека Email

Email утилиты — набор утилит для работы с различными почтовыми протоколами. Он позволяет отправлять и получать электронную почту, а также управлять почтовыми ящиками, используя следующие протоколы:

- **SMTP**
- **POP3**
- **Exchane**

Ниже приведены три примера использования этой библиотеки.

> **Важно**
>
> В этой статье многие параметры и учетные данные объявлены в коде. Для работы с ними в проектах настоятельно рекомендуется использовать **ConfigurationService** и **VaultService**. См. Служба настроек и Служба хранилища секретов.

1. **Отправка электронного письма через SMTP.**

SendSimpleEmailTask.java

```
package org.example.tasks;

import com.iba.kanclerrpa.email.RobotEmail;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import lombok.extern.slf4j.Slf4j;

@ApTaskEntry(name = "Send Email using EmailUtils")
@Slf4j
public class SendSimpleEmailTask extends ApTask {

    private static final String RECIPIENTS = "irina.nesterch@gmail.com";
    private static final String SUBJECT = "Test email";
    private static final String EMAIL_SERVICE = "smtp.gmail.com:587"; //gmail service for example
    private static final String EMAIL_SERVICE_PROTOCOL = "smtp_over_tsl";
    private static final String SENDER_NAME = "KanclerRPA Bot";
    private static final String BODY = "This message was sent by KanclerRPA Bot.";

    @Override
    public void execute() {

        SecretCredentials secret = new SecretCredentials("botmailbox@gmail.com", "password");

        RobotEmail emailSender = new RobotEmail();
        emailSender.setCredentials(secret);
        emailSender.setEmailService(EMAIL_SERVICE);
        emailSender.setEmailServiceProtocol(EMAIL_SERVICE_PROTOCOL);
        emailSender.setSender(secret.getUser());

        emailSender
                .subject(SUBJECT)
                .recipients(RECIPIENTS)
                .body(BODY)
                .setSenderName(SENDER_NAME);
        emailSender.send();
    }
}
```

2. **Создание таблицы в теле сообщения с помощью ftl-template, css и отправка по SMTP.**

   В этом подходе вы должны создать класс робота, расширяющий **RobotEmail**, переопределить метод **beforeSend()**, настроить робота в этом методе, а затем вызвать метод **send()** из его объекта.

RobotEmail использует FreeMarker для генерации кода тела сообщения. Подробнее о шаблонах FreeMarker и ftl читайте здесь.

SendFtlEmailTask.java

```
package org.example.tasks;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import org.example.entities.Book;
import org.example.robot.SendFtlEmailRobot;

import java.util.Arrays;
import java.util.List;

@ApTaskEntry(name = "Generate Email Report")
public class SendFtlEmailTask extends ApTask {

    // prepare test data
    public List<Book> getBooks() {
        Book thinkingInJava = new Book("1", "Thinking in Java", "Bruce Eckel");
        Book cipollino = new Book("2", "Le avventure di Cipollino", "Giovanni Francesco Rodari");
        Book wadAndPeace = new Book("3", "War and Peace", "Lev Tolstoy");
        List<Book> books = Arrays.asList(thinkingInJava, cipollino, wadAndPeace);
        return books;
    }

    @Override
    public void execute() {
        try {
            List<Book> books = getBooks();
            SendFtlEmailRobot robot = new SendFtlEmailRobot(new SecretCredentials("test@gmail.com",
"password"));
            robot.setDebtorsList(books).send();
        } catch (Exception e) {
            handleStepError(e);
        }
    }
}
```

SendFtlEmailRobot.java

```
package org.example.robot;

import com.google.common.io.Resources;
import com.iba.kanclerrpa.email.RobotEmail;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import org.example.entities.Book;
import java.io.IOException;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.List;

public class SendFtlEmailRobot extends RobotEmail {

    private static final String TEMPLATE_FILE_PATH = "books.email.tpl/books.ftl";

    private static final String RECIPIENTS = "recipient@gmail.com";

    private static final String SUBJECT = "Book list";

    private static final String EMAIL_SERVICE = "smtp.gmail.com:587";

    private static final String EMAIL_SERVICE_PROTOCOL = "smtp_over_tsl";

    private static final String SENDER_NAME = "RPA Bot";
```

```
    // typeName is a prefix for the parameters of a specific robot in the
    // configuration file
    private static final String TYPE_NAME = "books_email";

    private List<Book> books;

    private SecretCredentials secret;

    public SendFtlEmailRobot(SecretCredentials secret) {
        super();
        this.secret = secret;
        setTypeName(TYPE_NAME);
    }

    public SendFtlEmailRobot setDebtorsList(List<Book> books) {
        this.books = books;
        return this;
    }

    @Override
    protected void beforeSend() {
        setRecipients(Arrays.asList(RECIPIENTS));
        setCredentials(secret);
        subject(SUBJECT);
        setEmailService(EMAIL_SERVICE);
        setEmailServiceProtocol(EMAIL_SERVICE_PROTOCOL);
        sender(secret.getUser());
        senderName(SENDER_NAME);

        URL url = Resources.getResource(TEMPLATE_FILE_PATH);
        String body = null;
        try {
            body = Resources.toString(url, StandardCharsets.UTF_8);
        } catch (IOException e) {
            // do something
        }
        body(body);
        property("books", books);
    }
}
```

Исходный код шаблона и css:

books.ftl

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <#include "books.email.tpl/books.css">
</head>
<body>

<div class="container">
  <table class="responsive-table">
    <caption>Books</caption>
    <thead>
      <tr>
        <th scope="col">Name</th>
        <th scope="col">Author</th>
      </tr>
    </thead>
    <tbody>
      <#list books as book>
      <tr>
        <td data-title="Name">${book.name}</td>
        <td data-title="Author">${book.author}</td>
      </tr>
```

```
      </#list>
    </tbody>
  </table>
</div>

</body>
</html>
```

books.css

```
<style>
body {
        background: #fafafa url(https://jackrugile.com/images/misc/noise-diagonal.png);
        color: #444;
        font: 100%/30px 'Helvetica Neue', helvetica, arial, sans-serif;
        text-shadow: 0 1px 0 #fff;
}

strong {
        font-weight: bold;
}

em {
        font-style: italic;
}

table {
        background: #f5f5f5;
        border-collapse: separate;
        box-shadow: inset 0 1px 0 #fff;
        font-size: 14px;
        line-height: 24px;
        margin: 30px auto;
        text-align: center;
        width: 800px;
}

caption {
     font-size: 18px;
     font-weight: bold;
     }

th {
        background: url(https://jackrugile.com/images/misc/noise-diagonal.png), linear-gradient(#777,
#444);
        border-left: 1px solid #555;
        border-right: 1px solid #777;
        border-top: 1px solid #555;
        border-bottom: 1px solid #333;
        box-shadow: inset 0 1px 0 #999;
        color: #fff;
  font-weight: bold;
        padding: 10px 15px;
        position: relative;
        text-shadow: 0 1px 0 #000;
}

th:after {
        background: linear-gradient(rgba(255,255,255,0), rgba(255,255,255,.08));
        content: '';
        display: block;
        height: 25%;
        left: 0;
        margin: 1px 0 0 0;
        position: absolute;
        top: 25%;
        width: 100%;
}
```

```
th:first-child {
        border-left: 1px solid #777;
        box-shadow: inset 1px 1px 0 #999;
}

th:last-child {
        box-shadow: inset -1px 1px 0 #999;
}

td {
        border-right: 1px solid #fff;
        border-left: 1px solid #e8e8e8;
        border-top: 1px solid #fff;
        border-bottom: 1px solid #e8e8e8;
        padding: 10px 15px;
        position: relative;
        transition: all 300ms;
}

td:first-child {
        box-shadow: inset 1px 0 0 #fff;
}

td:last-child {
        border-right: 1px solid #e8e8e8;
        box-shadow: inset -1px 0 0 #fff;
}

tr {
        background: url(https://jackrugile.com/images/misc/noise-diagonal.png);
}

tr:nth-child(odd) td {
        background: #f1f1f1 url(https://jackrugile.com/images/misc/noise-diagonal.png);
}

tr:last-of-type td {
        box-shadow: inset 0 -1px 0 #fff;
}

tr:last-of-type td:first-child {
        box-shadow: inset 1px -1px 0 #fff;
}

tr:last-of-type td:last-child {
        box-shadow: inset -1px -1px 0 #fff;
}

tbody:hover td {
        color: transparent;
        text-shadow: 0 0 3px #aaa;
}

tbody:hover tr:hover td {
        color: #444;
        text-shadow: 0 1px 0 #fff;
}

</style>
```

Результат:

3. **Получение почты и список папок почтового ящика по протоколу POP3.**

ReadEmailTask.java

```
package org.example.tasks;

import com.iba.kanclerrpa.email.EmailClientProvider;
import com.iba.kanclerrpa.email.message.EmailMessage;
import com.iba.kanclerrpa.email.service.javax.ImapPop3EmailClient;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.model.SecretCredentials;
import com.iba.kanclerrpa.engine.service.ConfigurationService;
import lombok.extern.slf4j.Slf4j;

import javax.mail.search.AndTerm;
import javax.mail.search.ComparisonTerm;
import javax.mail.search.ReceivedDateTerm;
import javax.mail.search.SearchTerm;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

@ApTaskEntry(name = "Read Email using EmailUtils")
@Slf4j
public class ReadEmailTask extends ApTask {

    private static final String EMAIL_SERVICE = "imap.gmail.com:993";

    private static final String EMAIL_SERVICE_PROTOCOL = "imap";

    @Override
    public void execute() {
        ConfigurationService cfg = getConfigurationService();
        SecretCredentials credentials = new SecretCredentials("test@gmail.com", "password");

        EmailClientProvider emailScanner = new EmailClientProvider(cfg, credentials);
        emailScanner.setEmailService(EMAIL_SERVICE);
        emailScanner.setEmailServiceProtocol(EMAIL_SERVICE_PROTOCOL);

        ImapPop3EmailClient client = (ImapPop3EmailClient) emailScanner.getClient();

        // Now you can implement any search logic and call ImapPop3EmailClient method
        // e.g.
        // Get list of folders
        log.info("List of folders: " + client.fetchFolderList().toString());
        // Fetch all messages from the 'INBOX' folder during the last 3 days
        Date currentDate = new Date();
        Calendar cal = Calendar.getInstance();
        cal.setTime(currentDate);
        cal.add(Calendar.DATE, -3);
        Date fromDate = cal.getTime();

        cal.setTime(currentDate);
        cal.add(Calendar.DATE, 1);
```

```
        Date toDate = cal.getTime();
        SearchTerm olderThanTerm = new ReceivedDateTerm(ComparisonTerm.LE, toDate);
        SearchTerm newerThanTerm = new ReceivedDateTerm(ComparisonTerm.GE, fromDate);

        SearchTerm dateRangeTerm = new AndTerm(olderThanTerm, newerThanTerm);

        List<EmailMessage> messages = client.searchMessages("INBOX", dateRangeTerm);
        for (EmailMessage msg : messages) {
            log.info(msg.getSubject());
        }

    }
}
```

Другие файлы проекта, необходимые для запуска примера:

SendEmailsAp

```
package org.example.ap;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import lombok.extern.slf4j.Slf4j;

import org.example.tasks.SendSimpleEmailTask;

@Slf4j
@ApModuleEntry(name = "Test EmailUtils", description = "This process automates a lot of work...")
public class SendEmailAp extends ApModule {

    public TaskOutput run() throws Exception {
        return execute(getInput(), SendSimpleEmailTask.class).get();
    }
}
```

LocalRunner.java

```
package org.example.ap;

import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;

public class ModuleLocalRun {

    public static void main(String[] args) {
        //ApModuleRunner.localLaunch(SendEmailAp.class, new DevelopmentConfigurationModule(args));
        ApModuleRunner.localLaunch(SendEmailAp.class);
    }
}
```

Библиотека Excel

Excel утилиты — набор утилит для работы с электронными таблицами Excel. Он основан на библиотеке Apache POI и утилите cscript для запуска скриптов VBA.

Ниже приведен небольшой пример работы с существующим документом. Здесь мы создаем лист и заполняем его данными.

ExcelTask.java

```
package org.example.tasks;


import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.excel.SpreadsheetDocument;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.util.CellReference;

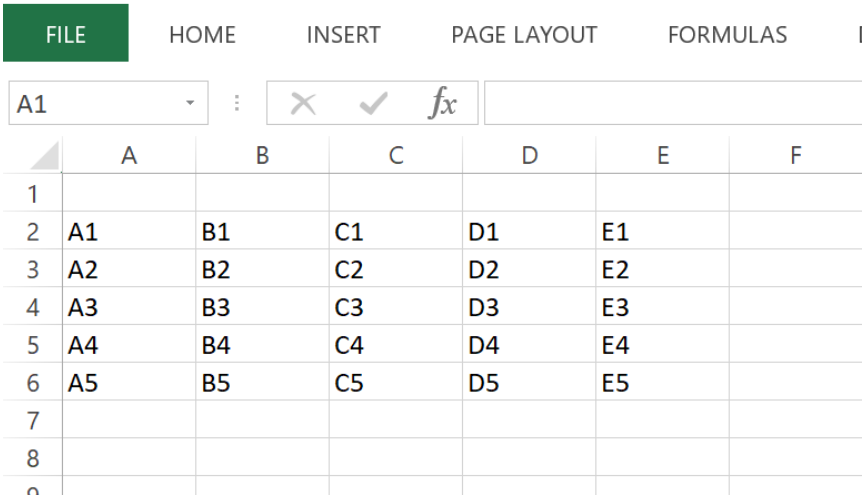
import java.io.FileInputStream;

@ApTaskEntry(name = "Create and read excel file")
public class ExcelTask extends ApTask {

    private static final String FILE_PATH = "C:\\Users\\Nestsiarchuk_IV\\ExcelLibrary\\app\\test.xlsx";

    @Override
    public void execute() {
        try {
            SpreadsheetDocument doc = new SpreadsheetDocument(new FileInputStream(FILE_PATH));
            Sheet numbersSheet = doc.createSheet("Numbers");

            for (int i = 0; i < 5; i++) {
                Row row = numbersSheet.createRow(i);
                for (int j = 0; j < 5; j++) {
                    Cell cell = row.createCell(j);
                    cell.setCellValue(String.format("%s%d", CellReference.convertNumToColString(j), i + 1));
                }
            }

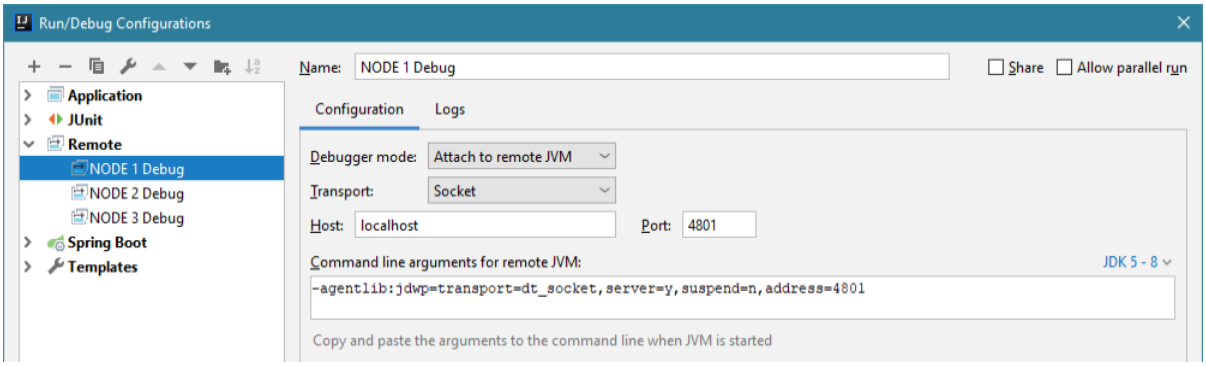
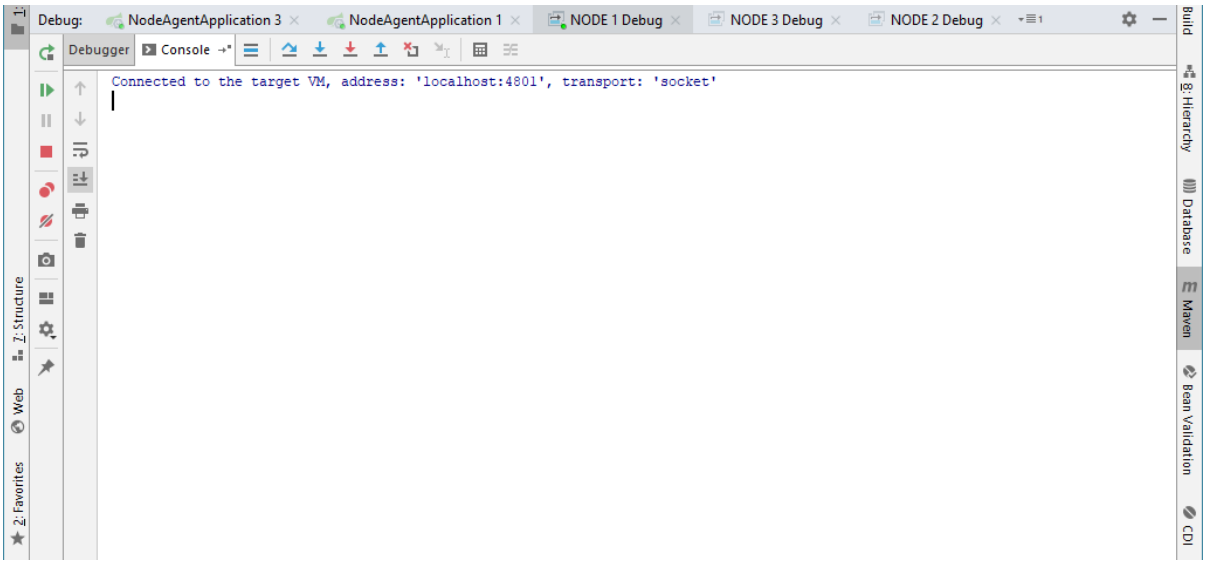
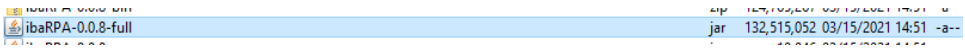
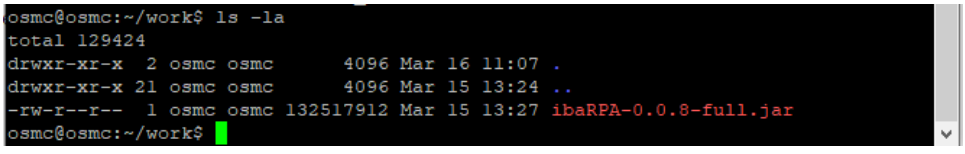
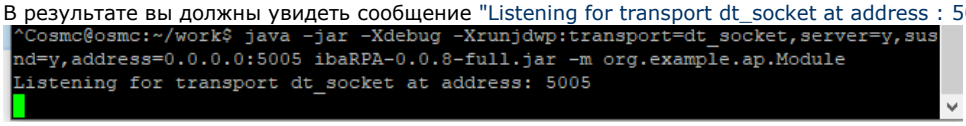
            doc.writeToFile(FILE_PATH);
        } catch (Exception e) {
            e.printStackTrace();
            // do something
        }
    }
}
```

В результате получаем лист:



ExcelSampleAp.java

```
package org.example.ap;

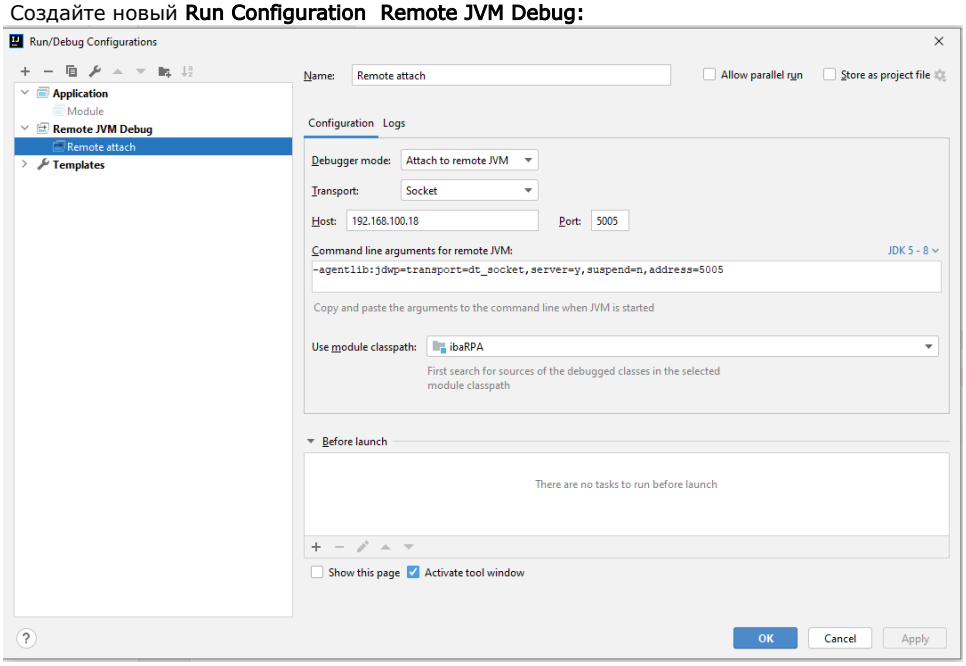
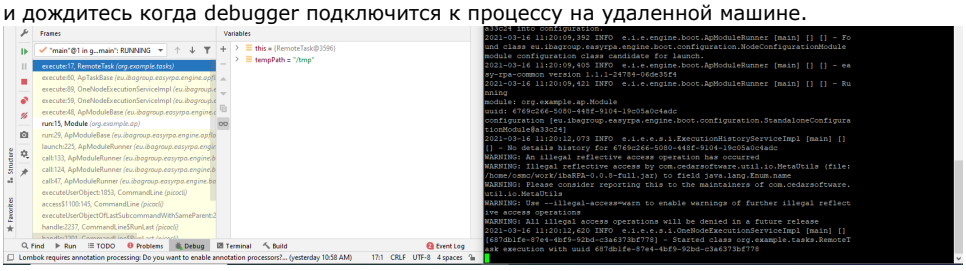
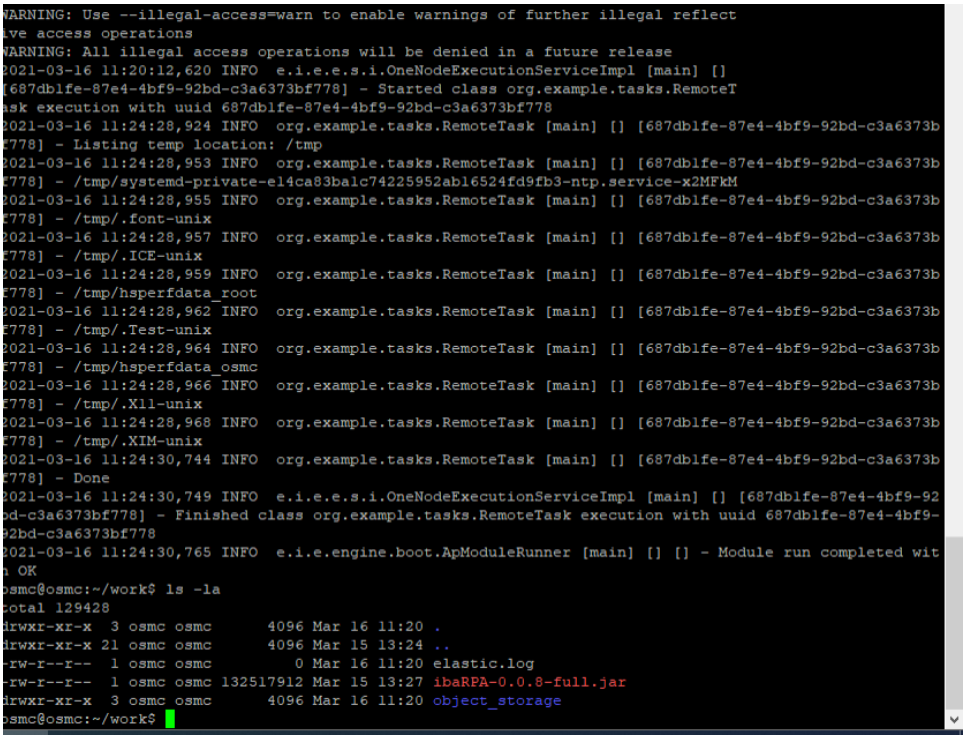
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import org.example.tasks.ExcelTask;

@ApModuleEntry(name = "Test ExcelUtils")
public class ExcelSampleAp extends ApModule {
    public TaskOutput run() throws Exception {
        return execute(getInput(), ExcelTask.class).get();
    }
}
```

LocalRunner.java

```
package org.example.ap;

import com.iba.kanclerrpa.engine.boot.ApModuleRunner;

public class ModuleLocalRun {

    public static void main(String[] args) {
        //ApModuleRunner.localLaunch(SendEmailAp.class, new DevelopmentConfigurationModule(args));
        ApModuleRunner.localLaunch(ExcelSampleAp.class);
    }
}
```

Отладка автоматизированного процесса на удаленном узле с сервером управления

Обзор

Во многих случаях, когда вы не можете настроить клиентские приложения на компьютере разработчика, можно проверить запуск автоматизированного процесса на удаленном узле. Узел запускает автоматизированный процесс как отдельный процесс Java, вследствие чего можно использовать стандартный подход отладки Java.

Указание флагов отладки на узле

Перейдите к конфигурации узла и настройте необходимые параметры:

Название	Параметр	Значение по умолчанию
JAVA_DEBUG	Включить или отключить отладку автоматизированного процесса	n
JAVA_DEBUG_PORT_BASE	Базовый порт отладки. Поскольку на узле можно запустить несколько JVM, это определяет базовый порт для отладки. Если порт занят, узел найдет первый доступный порт.	1044
JAVA_DEBUG_SUSPEND	Приостановить целевую JVM	n
JAVA_DEBUG_HOST	Хост для прослушивания. Локальный хост по умолчанию	

В журнале автоматизированного процесса появится следующая строка:

-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=**1050**

Подключение удаленного отладчика

В вашей среде IDE настройте удаленную конфигурацию запуска отладки Java. Для IDEA это выглядит так:



Просмотрите логи узла и, как только появится сообщение об ожидании соединения, запустите настройку отладчика:


Debug:
NodeAgentApplication 3
NodeAgentApplication 1
NODE 1 Debug
NODE 3 Debug
NODE 2 Debug
Debugger
Console
Connected to the target VM, address: 'localhost:4801', transport: 'socket'
Build
8: Hierarchy
Database
Maven
Bean Validation
CDI
1: Project
7: Structure
Web
2: Favorites

Отладка автоматизированного процесса на удаленном узле без сервера управления

Иногда разработчик RPA может начать разработку раньше, чем сервер управления Канцлер RPA и инфраструктура будут готовы. Можно запустить автоматизированный процесс на удаленном узле. Узел запускает автоматизированный процесс как отдельный процесс Java, вследствие чего можно использовать стандартный подход отладки Java.

Шаги

1. Создайте "uber" jar проекта автоматизированного процесса, например, используя плагин "shade":

pom.xml

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <configuration>
                <filters>
                        <filter>
                                <artifact>*:*</artifact>
                                <excludes>
                                        <exclude>module-info.class</exclude>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                </excludes>
                        </filter>
                </filters>
                <transformers>
                        <transformer
                                implementation="org.apache.maven.plugins.shade.resource.
ManifestResourceTransformer">
                                <mainClass>com.iba.kanclerrpa.engine.boot.ApModuleRunner</mainClass>
                        </transformer>
                </transformers>
        </configuration>
        <executions>
                <execution>
                <goals>
                        <goal>shade</goal>
                </goals>
                <configuration>
                        <shadedArtifactAttached>true</shadedArtifactAttached>
                        <shadedClassifierName>full</shadedClassifierName>
                        <transformers>
                                <transformer
                                        implementation="org.apache.maven.plugins.shade.resource.
ManifestResourceTransformer">
                                </transformer>
                        </transformers>
                </configuration>
                </execution>
        </executions>
</plugin>
```



2. Загрузите jar на удаленную машину. В приведенном ниже примере он был загружен в Linux Node Agent с помощью ssh:



3. Установите JRE на удаленную машину.

4. Запустите jar автоматизированного процесса как удаленный сервер, указав **параметры отладки**:

Command

```
java -jar -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=0.0.0.0:5005 ibaRPA-0.0.8-full.jar -m org.example.ap.Module
```

где "-m" - модуль для запуска

В результате вы должны увидеть сообщение "Listening for transport dt_socket at address : 5005":



5. На локальной машине в Intellij IDEA добавьте точка останова в коде:

```
@Slf4j
@ApTaskEntry(name = "Remote Task")
public class RemoteTask extends ApTaskBase {

    @Override
    public void execute() throws Exception {
        String tempPath = System.getProperty("java.io.tmpdir");
        log.info("Listing temp location: {}", tempPath);
        Files.list(Paths.get(tempPath)).limit(10).forEach(p -> log.info(p.toString()));
        log.info("Done");
    }
}
```

6. Создайте новый **Run Configuration Remote JVM Debug:**



7. Выберите опцию **Attach to remote JVM** в списке **Debugger mode**. Укажите адрес хоста и порт. Запустите **Run Configuration** и дождитесь когда debugger подключится к процессу на удаленной машине.



8. 

Для получения большей информации, ознакомьтесь со следующей ссылкой: Intellij IDEA Tutorial: Remote debug.

Гибридная автоматизация

Канцлер RPA реализует подход гибридной автоматизации, который позволяет людям и роботам RPA работать вместе, передавая друг другу задачи. Полученное комплексное решение RPA сочетает в себе эффективность и точность роботов с творческим потенциалом и гибкостью человека.

Пользовательская задача — это задача, в которой участвует пользователь. Довольно часто эта задача требует, чтобы человек взаимодействовал с другими службами, и, таким образом, становится задачей в рамках более крупной бизнес-цели.

Канцлер RPA предоставляет разработчикам автоматизированных процессов набор встроенных пользовательских задач и фреймворк для создания пользовательских задач.

Встроенные пользовательские задачи

Тип пользовательской задачи - это специальное небольшое веб-приложение, которое может считывать и анализировать конфигурацию вашего типа документа и знает, как отображать входные документы.

Тип документа - это конфигурация, которая определяет поля вывода, которые вы хотите извлечь из документа, и определяет дополнительную информацию, которую пользовательская задача использует для отображения входного документа.

Каждый **тип документа** относится к определенному **типу пользовательской задачи**.

Канцлер RPA предоставляет 4 типа встроенных пользовательских задач:

- Пользовательская задача по извлечению информации
- Пользовательская задача по извлечению информации из HTML
- Пользовательская задача по классификации документов
- Пользовательская задача заполнения формы

Пользовательская задача по извлечению информации

Извлечение информации - это процесс извлечения структурированной информации (или ключевых фактов) из неструктурированных и/или полуструктурированных документов (счета-фактуры, требования платежей и т. д.).

Пользовательская задача по извлечению информации может использоваться для:

- Сохранения и обработки данных, извлеченных пользователем в автоматизированном процессе.
- Подготовки тренировочного набора для обучения модели машинного обучения.
- Проверки данных, извлеченных моделью машинного обучения.

JSON-структура типа документа

Пример настроек JSON для задачи по извлечению информации:

Пример JSON настроек задачи извлечения информации

```
{
  "regexFunctions": {
    "numberMore10": "(value) => { return Number(value) > 10}",
    "lengthMore3": "(value) => { return value.length > 3}"
  },
  "taskInstructionText": "      ,   .",
  "taskInstructionLink": "https://172.20.194.57:8444/minio/data/ht_articles_classification/instruction.html",
  "taskTypeLabel": " ",
  "autoSave": false,
  "allowCustomValue": false,
  "appLanguage": "ru",
  "excludeUndefinedEntities": false,
  "categories": [
    {
      "name": "",
      "multiple": true,
      "regex": "^[0-9]*$"
    },
    {
      "name": "",
      "multiple": true
    },
    {
      "name": " ",
      "multiple": true,
      "required": true
    },
    {
      "name": " ",
      "multiple": true,
      "required": true,
      "helperText": "    10",
      "regexFnc": [
        "numberMore10",
        "lengthMore3"
      ]
    },
    {
      "name": " ",
      "multiple": true
    }
  ],
  "metadata": [
     {
```

```
  "name": "isInvalid",
  "markLabel": " ",
  "description":" ,     ."
 },
 {
  "name": "___12",
  "type": "input",
  "label": "   ",
  "mask": "+4\\9 99 999 99"
},
{
  "name": "__1",
  "type": "input",
  "label": "  ",
  "validationRegExp": "^[a-zA-Z]+$",
  "errorMessage": " "
},
{
  "name": "__10",
  "label": "   ",
  "type": "date",
  "format": "D MMM YYYY"
},
{
  "name": "__9",
  "label": "  ",
  "type": "date",
  "format": "MMMM DD"
},
{
  "name": "__8",
  "label": " ",
  "type": "date",
  "format": "YYYY"
},
{
  "name": "__7",
  "label": " ",
  "type": "date",
  "keyboard": true,
  "format": "YYYY"
},
{
  "name": "__6",
  "label": "_",
  "type": "date",
  "keyboard": true,
  "format": "MM"
},
{
  "name": "__5",
  "label": " .",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM"
},
{
  "name": "__4",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM.YY"
},
{
  "name": "__3",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM.YYYY"
},
{
```

```
  "name": "__2",
  "label": " --",
  "type": "date",
  "keyboard": true,
  "format": "MM-DD-YYYY"
},
{
  "name": "__1",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "MM.DD.YYYY",
  "description": "  .."
},
{
  "name": "__1",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "__2",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},     {
  "name": "_1",
  "type": "select",
  "required": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "_2",
  "type": "select",
  "required": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "__1",
      "label": "_1"
    },
    {
      "value": "__2",
      "label": "_2"
    },
```

```
      {
        "value": "__3"
      }
    ]
  },
  {
    "name": "__1",
    "type": "select",
    "required": true,
    "multiple": true,
    "autocomplete": true,
    "label": "  ",
    "description": "   ",
    "items": [
      "_1",
      "_2",
      "_3"
    ]
  },
  {
    "name": "__2",
    "type": "select",
    "required": true,
    "multiple": true,
    "autocomplete": true,
    "label": "  ",
    "description": "   ",
    "items": [
      {
        "value": "_1",
        "label": "_1"
      },
      {
        "value": "_2",
        "label": "_2"
      },
      {
        "value": "_3"
      }
    ]
  },
  {
    "name": "_1",
    "type": "select",
    "required": true,
    "autocomplete": true,
    "label": "  ",
    "description": "   ",
    "items": [
      "_1",
      "_2",
      "_3"
    ]
  },
  {
    "name": "_2",
    "type": "select",
    "required": true,
    "autocomplete": true,
    "label": "  ",
    "description": "   ",
    "items": [
      {
        "value": "_1",
        "label": "_1"
      },
      {
        "value": "_2",
        "label": "_2"
      },
      {
```

```
      "value": "_3"
    }
  ]
},
{
  "name": "_1",
  "type": "checkbox",
  "markLabel": " ",
  "label": " ",
  "description": "  "
},
{
  "name": "___1",
  "type": "input",
  "label": "   ",
  "mask": "+7 (999) 999-99-99"
},
{
  "name": "___2",
  "type": "input",
  "label": "    ",
  "mask": "9999-9999-9999-9999",
  "required": true,
  "description": "     "
},
{
  "name": "__1",
  "type": "input",
  "label": " ",
  "validationRegExp": "^[a-zA-Z]+$",
  "errorMessage": " "
},
{
  "name": "__1",
  "type": "number_input",
  "label": "  ",
  "description": "   "
},
{
  "name": "__1",
  "type": "textarea",
  "label": "  ",
  "description": "   "
},
{
  "name": "__2",
  "type": "textarea",
  "minRows": 5,
  "maxRows": 8
},
{
  "name": "__1",
  "type": "checkbox_group",
  "required": true,
  "label": "   ",
  "description": "    ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2",
      "disabled": true
    },
    {
      "value": "_3"
    }
  ]
},
```

```
    {
      "name": "__1",
      "type": "radio_group",
      "required": true,
      "label": "  ",
      "items": [
        "_1",
        "_2",
        "_3"
      ]
    },
    {
      "name": "__2",
      "type": "radio_group",
      "required": true,
      "label": "  ",
      "description": "   ",
      "items": [
        {
          "value": "_1",
          "label": "_1"
        },
        {
          "value": "_2",
          "label": "_2",
          "disabled": true
        },
        {
          "value": "_3"
        }
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,            .",
        ",       react-select  downshift"
      ]
    },     {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,            .",
        ",       react-select  downshift"
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": "             :          , ,       .       ,      , ,           ."
    }
  ]
}
```

Эти настройки содержат:

- **regexFunctions** (object) (необязательно) - специальный объект, который содержит пары "*key" "value*" для указания валидаторов пользовательских полей, где "*key"* - это имя вашей пользовательской функции, которое можно использовать в качестве ссылки в опции "***regexFnc***" настроек полей в "*categories*". "*value"* - это ваша пользовательская лямбда-функция JavaScript.
- **taskInstructionText** (string) (необязательно) - представляет текст инструкций, который появляется во всплывающем окне.
- **taskInstructionLink** (string) (необязательно) - представляет ссылку на удаленный источник с инструкцией. Наличие хотя бы одного из этих полей ***taskInstructionText*** или ***taskInstructionLink*** вызовет появление кнопки **Инструкция**.

- **taskTypeLabel** (string) (необязательно) - позволяет настроить название задачи. По умолчанию для него установлено значение "*Document Classification*".
- **autoSave** (boolean) (необязательно) - позволяет автоматически сохранять промежуточные результаты задач. По умолчанию этот параметр имеет значение "*false*".
- **allowCustomValue** (boolean) (необязательно) - позволяет вводить пользовательское значение без какого-либо подключения к изображению входного документа. Эта возможность полезна, когда *OCR* не удалось извлечь некоторый текст из документа.
- **appLanguage** (string) (необязательно) - позволяет пользователю настроить локализацию Пользовательской задачи. В настоящее время доступны опции "*en*" и "*ru*". По умолчанию отображается значение "*en*".
- **excludeUndefinedEntities** (boolean) (необязательно) - позволяет получать выходные данные только тех полей, которые настроены в параметре "*categories*". По умолчанию параметр имеет значение "*false*".
- **commentsSectionName** (string) (необязательно) - используется для переименования вкладки "***Подробности***". По умолчанию этот параметр имеет значение "*Подробности*".
- **categories** (list of objects) (обязательно) - список полей для извлечения из документа, где каждый элемент содержит:
    - **name** (string) (обязательно) - он служит ключом к получению данных из выходных данных пользовательской задачи.
    - **multiple** (boolean) (обязательно) - показывает, может ли это поле иметь несколько значений (используется, когда у вас есть список с подробными сведениями о некоторых элементах в документе, например, список продуктов).
    - **required** (boolean) (необязательно) - показывает, обязательно ли поле для извлечения, поэтому человек не сможет отправить задачу без указания значений для всех обязательных полей.
    - **helperText** (string) (необязательно) - дополнительный текст, который отображается, если значение недопустимо из-за дополнительных валидаторов, заданных параметрами *regexp*.
    - **regex** (string) (необязательно) - указывает функцию *regexp*, используемую для проверки значений.
    - **regexFnc** (list of string) (необязательно) - список названий функций валидатора, которые задаются в "***regexFunctions***".
    - **iconType** (string) (необязательно) - позволяет настроить иконку поля. В настоящее время доступны следующие варианты: *"date", "money", "multiple", "text"*. По умолчанию используется значок *"text"*.
    - **hotkey** (list of string) (необязательно) - список сочетаний клавиш, которые можно нажать, чтобы выбрать элемент в категории.
- **metadata** (list of objects) (необязательно) - используется для настройки вкладки "***Подробности***" в Рабочем пространстве и представления документов в Пакетах документов. Вкладка "***Подробности***" отключена, если ***metadata*** не определена. Флажок "Invalid Document" на вкладке "***Подробности***" можно настроить, но нельзя удалить. Если установлен флажок "Invalid Document", все проверки будут отключены.

Установленный по умолчанию флажок *"Invalid Document"* может быть настроен при помощи следующих настроек:

- **name** (string) (обязательно) - используется для объявления раздела, который используется для настройки названия и описания флажка "Invalid Document". Значением по умолчанию является "*isInvalid*" и его нельзя изменить, при его изменении параметры ***markLabel*** и ***description*** будут иметь значения по умолчанию.
- **markLabel** (string) (необязательно) - используется для определения имени флажка по умолчанию на вкладке "***Подробности***". По умолчанию этот параметр имеет значение *"Invalid Document"*.
- **description** (string) (необязательно) - позволяет добавить описание к флажку "*Invalid Document*" по умолчанию. По умолчанию описание не отображается.

Дополнительные поля можно добавить на вкладку ***"Подробности"*** с помощью следующих настроек:

- **name** (string) (обязательно) - он служит ключом к получению данных из вкладки ***"Подробности"*** из выходных данных Пользовательской задачи. Требуется для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **label** (string) (необязательно) - используется для задания названия поля.
- **type** (string) (обязательно) - задает тип поля. Должен быть один из поддерживаемых типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, info, date***.
- **required** (boolean) (необязательно) - показывает, является ли поле обязательным для заполнения во вкладке ***"Подробности"***, чтобы пользователь не сможг отправить Пользовательскую задачу без указания значений для всех обязательных полей на вкладке ***"Подробности"***, за исключением случая, когда установлен флажок *"Invalid Document"*.
- **multiple** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет выбрать несколько элементов в выпадающем списке. Может работать с настройкой ***autocomplete***.
- **autocomplete** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет осуществлять поиск по элементам в выпадающем списке. Может работать с настройкой ***multiple***.
- **mask** (string) (необязательно) - используется только если ***"type": "input"***. Позволяет задать строку символов, указывающую формат допустимых входных значений. Формат символов по умолчанию: "9" для символов *0-9*; "a" для символов *A-Z* и *a-z*; "" для симвоов *A-Z*, *a-z*, *0-9*. Если требуется иметь именно символ "9" (например, телефонный код), используйте "\\" перед символом (например, "+4\\9 99 999 99"). Если ***"required": true*** и ***mask*** установлена для ***"type": "input"***, входные данные должны быть заполнены полностью в соответствии с ***mask***.
- **minRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает минимальное количество строк, отображаемых в ***textarea***. По умолчанию этот параметр имеет значение "2".

- **maxRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает максимальное количество строк, отображаемых в текстовой области без появления прокрутки. Если количество строк больше параметра ***maxRows***, появится прокрутка. Значение по умолчанию отсутствует, и поле растет без ограничений.
- **disabled** (boolean) (необязательно) - предоставляет возможность отключения поля, тем самым не позволяет пользователю заполнить это поле.
- **validationRegExp** (string) (необязательно) -предоставляет возможность проверки заполненных значений с помощью регулярных выражений.
- **description** (string) (необязательно) - позволяет добавить описание поля. Может быть добавлено для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **errorMessage** (string) (необязательно) - используется для задания текста ошибки, если заполненные значения не соответствуют регулярному выражению в параметре ***validationRegExp***.
- **items** (list of objects) (обязательно только для "type": "checkbox_group", "radio_group", "select") - используется только если **"type": "checkbox_group", "radio_group","select"**. Задает возможные элементы для проверки. Каждый элемент в списке имеет следующие свойства:
  - **value** (string) (обязательно) - значение отображаемого элемента.
  - **label** (string) (необязательно) - метка отображаемого элемента. Если ***label*** не задан, вместо него отображается ***value***. Если установлены оба параметра, ***label*** отображается в разделе Пользовавтельская задача, а ***value*** задается в выходных данных Пользовательской задачи.
  - **disabled** (boolean) (необязательно) - предоставляет возможность отключить элемент, что помешает пользователю выбрать его.
- **markLabel** (string) (необязательно) - используется только если ***"type": "checkbox"***. Отображает метку флажка.
- **text** (string or array) (необязательно) - используется только если ***"type": "info"***. Отображает нередактируемый текст для информационного поля. Если текст содержит массив строк, каждая строка будет рассматриваться как новый абзац.
- **keyboard** (boolean) (необязательно) - используется только если ***"type": "date"***. Позволяет включить ручной ввод даты для формы ***datepicker***. По умолчанию параметр имеет значение *"false"*.
- **format** (string) (необязательно) - используется только если ***"type": "date"***. Позволяет установить пользовательский формат даты. По умолчанию этот параметр имеет значение "MM/DD/YYYY". Поддерживает следующие токены форматирования для дат:

	Токен	Выходные данные
Месяц	M	1 2 ... 11 12
	MM	01 02 ... 11 12
	MMM	Янв Фев ... Ноя Дек
	MMMM	Январь Февраль ... Ноябрь Декабрь
День	D	1 2 ... 30 31
	DD	01 02 ... 30 31
Год	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030

Горячие клавиши

Задача «Извлечение информации» имеет следующие горячие клавиши:

- **Tab** - для перехода к следующему полю на правой панели.
- **Shift + Tab** - для перехода к предыдущему полю на правой панели.
- **Shift + выделение области** в документе – чтобы добавить выделенные слова в конец текущего поля.
- **Shift + Enter** - добавить разрыв строки в режиме редактирования.
- **Двойной щелчок** по полю в документе, чтобы применить тег к текущему полю.
- **Ctrl + Z** - отмена последнего действия.
- **Ctrl + Shift + Z** - вернуть последнее отмененное действие обратно.
- **Ctrl + Колесо мыши** - увеличить/уменьшить масштаб документа.

Правила для настройки горячих клавиш:

- не должны быть заняты;
- не должны повторяться;
- не должны содержать несколько букв или цифр;
- должны содержать только существующие клавиши клавиатуры;
- могут быть в любом регистре;

- могут состоять из одной и больше кнопок.

Входные данные документа в виде JSON для обработанных PDF

Представляет результат распознавания документа в формате JSON, наложенный на исходное изображение документа. Генерируется автоматически при перезапуске OCR:

Пример ввода JSON для PDF-документов

```
{
        "images": [ // There should be an array to support multipage PDFs. Each page should be defined
individually in this case.
                {
                        "content": "https://some-provider.com/img.jpg",
                        "json": <OCR-JSON>,
                        "dimensions": { // The dimensions of the page in pixels.
                                "width": 2000,
                                "height": 3000
                        }
                }
        ]
}
```

- Входной JSON содержит:
    - **images** (list of objects) - корневой элемент, содержащий список конфигураций документа. Список обычно содержит только один элемент.
        - **content** (url or base64 string) - источник входного документа для отображения. Это может быть **URL** документа или содержание документа, закодированное в base64 (например, строковое значение "....").
        - **json** (**OCR-JSON** object) - предоставляет информацию OCR. Структура OCR-JSON описана ниже.
        - **dimensions** (object) - объект содержит параметры «**width**» и «**height**», которые представляют ширину и высоту исходного входного документа.
            - **width** (integer) - значение ширины исходного входного документа.
            - **height** (integer) - значение высоты исходного входного документа.

OCR-JSON имеет следующую структуру, которая генерируется компонентом OCR самостоятельно:

OCR-JSON

```
"json": {
        "pages": [
                "id": "page0",
                "areas": [
                        {
                                "id": "page0_area0",
                                "paragraphs": [
                                        {
                                                "id": "page0_area0_paragraph0",
                                                "lines": [
                                                        {
                                                                "id": "page0_area0_paragraph0_line0",
                                                                "words": [
                                                                        {
                                                                                "id":
"page0_area0_paragraph0_line0_word0",
                                                                                "text": "Advanced",
                                                                                "properties": {
                                                                                        "bbox": [

0.05999032414126754,

0.037290455011974,

0.14078374455732948,
```

```
0.046527540198426275
                                                                        ],
                                                                    "x_fsize": 0,
                                                                    "x_wconf": 96
                                                                }
                                                            },
                                                        }
                                                    ...
                                                    ]
                                                }
                                            ]
                                        }
                                    ]
                                }
```

Этот JSON содержит все слова, подвергнутые распознаванию, и имеет древовидную структуру: **pages areas paragraphs lines words**.

- **json** (object) - корневой элемент
  - **pages** (list of objects) - структура списка страниц. Каждая страница в списке имеет следующую структуру:
    - **id** (string) - ID страницы
    - **areas** (list of objects) - структура списка областей. Каждая область в списке имеет следующую структуру:
      - **id** (string) - ID области
      - **paragraphs** (list of objects) - структура списка абзацев. Каждый абзац в списке имеет следующую структуру:
        - **id** (string) - ID параграфа
        - **lines** (list of objects) - структура списка строк. Каждая строка в списке имеет следующую структуру:
          - **id** (string) - ID строки
          - **words** (list of objects) - структура списка слов. Каждое слово в списке имеет следующую структуру:
            - **id** (string) - ID слова
            - **text** (string) - исходный текст, извлеченный механизмом OCR
            - **properties** (object) - объект свойства со следующей структурой:
              - **bbox** (list of integers) - верхняя левая и нижняя правая координаты прямоугольника вокруг слова в исходном документе. Координаты нормализуются от 0 до 1 относительно исходного размера документа
            - **x_fsize** (integer) - это размер шрифта, специфичный для механизма OCR
            - **x_wconf** (integer) - Достоверность работы OCR для всей подстроки. Более высокие значения выражают более высокую достоверность.

Во входном JSON можно использовать следующие дополнительные настройки:

- **overrides** (необязательно) - позволяет изменить поле и его параметры из настроек типа документа, если оно используется в качестве ввода по умолчанию. Поле, которое нужно изменить, должно быть объявлено в блоке по параметру **name**. Затем необходимо объявить и определить настройки изменяемого поля. Обратите внимание, что «name» и «type» не могут быть изменены.

  ```
  "overrides": {
    "ошибка_ввода_1": {
      "label": "Обновленная обычная ошибка ввода"
    }
  ```
  } - параметр "label" для поля с параметром "name": "ошибка_ввода_1" будет измнен на установленное значение.

- **messages** (необязательно) - позволяет отобразить блок с сообщением или несколькими сообщениями. Блок можно настроить с различными параметрами серьезности: "**info**", "**warning**", "**error**".

  - **"messages": ["Информационное сообщение"]** - отобразить блок с сообщением «Информационное сообщение» с уровнем серьезности по умолчанию «info»;
  - **"messages": [{ "severity": "info", "text": "Информационное сообщение" }]** - отобразить информационное сообщение "Информационное сообщение";
  - **"messages": [{ "severity": "warning", "text": "Предупреждающее сообщение" }]** - отобразить предупреждающее сообщение "Предупреждающее сообщение";
  - **"messages": [{ "severity": "error", "text": "Сообщение об ошибке" }]** - отобразить сообщение об ошибке "Сообщение об ошибке";
  - ```
    "messages":  [{
        "severity": "error",
        "text": "Сообщение об ошибке"
      },
    ```
    ```

{
"severity": "info",
"text": "Информационное сообщение"
},
{
"severity": "warning",
"text": "Предупреждающее сообщение"
}
], - чтобы отобразить несколько сообщений разной степени серьезности.

Результат работы пользователя в виде JSON для обработанных PDF

В качестве результата работы над пользовательской задачей по извлечению информации из PDF создается следующий JSON:

Пример вывода JSON для PDF-документов


```
{
  "entities": [
    {
      "content": "NASH JEFFREY M",
      "name": "Reporter Name",
      "words": [
        {
          "content": "NASH",
          "bbox": [
            0.06562091503267974,
            0.17454545454545456,
            0.1126797385620915,
            0.18383838383838383
          ],
          "id": "page0_area17_paragraph0_line7_word0",
          "page": 0
        },
        {
          "content": "JEFFREY",
          "bbox": [
            0.11764705882352941,
            0.17474747474747473,
            0.18797385620915033,
            0.18383838383838383
          ],
          "id": "page0_area17_paragraph0_line7_word1",
          "page": 0
        },
        {
          "content": "M",
          "bbox": [
            0.19294117647058823,
            0.17474747474747473,
            0.20784313725490197,
            0.18363636363636363
          ],
          "id": "page0_area17_paragraph0_line7_word2",
          "page": 0
        }
      ],
      "index": 0,
      "multiple": false
    }
  ],
  "target": "metamask-inpage",
  "data": {
    "name": "metamask-provider",
    "data": {
      "method": "metamask_chainChanged",
      "params": {
```

```
        "chainId": "0x38",
        "networkVersion": "56"
      }
    }
  },
  "metadata": {
    "input_custom_error_1": "test",
    "datepicker_5": "2 Apr 2022",
    "datepicker_4": "April 03",
    "datepicker_3": "2022",
    "select_multiple_1": [
      "string_1",
      "string_2"
    ],
    "select_2": "real_value_1",
    "autocomplete_multiple_1": [
      "string_1"
    ],
    "checkbox_1": true,
    "number_input_1": "7",
    "radio_group_1": "real_value_1"
  }
}
```

Он имеет следующую структуру:

- **entities** (list of objects) -корневой элемент, который содержит список извлеченных сущностей. Каждый объект в этом списке имеет структуру, описанную ниже.:
  - **content** (string) - окончательный выходной текст извлеченной сущности.
  - **name** (string) - название извлеченной сущности.
  - **words** (list of objects) - список словесных объектов. Если извлеченный текст состоит из нескольких слов, этот список будет содержать несколько словесных объектов следующим образом:
    - **content** (string) - исходный текст из входного документа.
    - **bbox** (list of integers) - верхняя левая и нижняя правая координаты прямоугольника, окружающего слово в исходном документе. Координаты нормализуются в диапазоне от 0 до 1 относительно исходного размера документа.
    - **id** (string) - ID слова.
    - **page** (integer) - номер страницы исходного документа, на которой встречается это слово.
  - **index** (integer) - индекс сущности.
  - **multiple** (boolean) - показывает, может ли это поле иметь несколько значений.
- **metadata** (list of objects) - существует только в том случае, если поля из раздела «Подробности/переименованные метаданные» были заполнены. Представлен в виде структуры ключ-значение, где ключ — это имя поля, определенное в настройках типа документа, а значение — это значение, помещенное в поле.

Пользовательская задача по извлечению информации из HTML

Извлечение информации - это процесс извлечения структурированной информации (или ключевых фактов) из неструктурированных и/или полуструктурированных документов (счета-фактуры, требования, новости о дивидендах и т. д.).

Пользовательская задача по извлечению информации может использоваться для:

- Сохранения и обработки данных, извлеченных пользователем в автоматизированном процессе
- Подготовка тренировочного набора для обучения модели машинного обучения
- Проверки данных, извлеченных моделью машинного обучения

JSON-структура типа документа

Пример настроек JSON для задачи по извлечению информации:

Пример JSON настроек задачи извлечения информации

```
{
  "regexFunctions": {
    "numberMore10": "(value) => { return Number(value) > 10}",
    "lengthMore3": "(value) => { return value.length > 3}"
  },
  "taskInstructionText": "      ,   .",
  "taskInstructionLink": "https://172.20.194.57:8444/minio/data/ht_articles_classification/instruction.html",
  "taskTypeLabel": " ",
  "autoSave": false,
  "allowCustomValue": false,
  "appLanguage": "ru",
  "excludeUndefinedEntities": false,
  "categories": [
    {
      "name": "",
      "multiple": true,
      "regex": "^[0-9]*$"
    },
    {
      "name": "",
      "multiple": true
    },
    {
      "name": " ",
      "multiple": true,
      "required": true
    },
    {
      "name": " ",
      "multiple": true,
      "required": true,
      "helperText": "    10",
      "regexFnc": [
        "numberMore10",
        "lengthMore3"
      ]
    },
    {
      "name": " ",
      "multiple": true
    }
```

```
],
"metadata": [
   {
    "name": "isInvalid",
    "markLabel": " ",
    "description":" ,     ."
   },
   {
    "name": "___12",
    "type": "input",
    "label": "   ",
    "mask": "+4\\9 99 999 99"
  },
  {
    "name": "__1",
    "type": "input",
    "label": "  ",
    "validationRegExp": "^[a-zA-Z]+$",
    "errorMessage": " "
  },
  {
    "name": "__10",
    "label": "   ",
    "type": "date",
    "format": "D MMM YYYY"
  },
  {
    "name": "__9",
    "label": "  ",
    "type": "date",
    "format": "MMMM DD"
  },
  {
    "name": "__8",
    "label": " ",
    "type": "date",
    "format": "YYYY"
  },
  {
    "name": "__7",
    "label": " ",
    "type": "date",
    "keyboard": true,
    "format": "YYYY"
  },
  {
    "name": "__6",
    "label": "_",
    "type": "date",
    "keyboard": true,
    "format": "MM"
  },
  {
    "name": "__5",
    "label": " .",
    "type": "date",
    "keyboard": true,
    "format": "DD.MM"
  },
  {
    "name": "__4",
    "label": " ..",
    "type": "date",
    "keyboard": true,
    "format": "DD.MM.YY"
  },
  {
    "name": "__3",
    "label": " ..",
    "type": "date",
    "keyboard": true,
```

```
  "format": "DD.MM.YYYY"
},
{
  "name": "__2",
  "label": " --",
  "type": "date",
  "keyboard": true,
  "format": "MM-DD-YYYY"
},
{
  "name": "__1",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "MM.DD.YYYY",
  "description": "  .."
},
{
  "name": "__1",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "__2",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},     {
  "name": "_1",
  "type": "select",
  "required": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "_2",
  "type": "select",
  "required": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "__1",
      "label": "_1"
    },
    {
```

```
      "value": "__2",
      "label": "_2"
    },
    {
      "value": "__3"
    }
  ]
},
{
  "name": "__1",
  "type": "select",
  "required": true,
  "multiple": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "__2",
  "type": "select",
  "required": true,
  "multiple": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},
{
  "name": "_1",
  "type": "select",
  "required": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "_2",
  "type": "select",
  "required": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
```

```
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},
{
  "name": "_1",
  "type": "checkbox",
  "markLabel": " ",
  "label": " ",
  "description": "  "
},
{
  "name": "___1",
  "type": "input",
  "label": "   ",
  "mask": "+7 (999) 999-99-99"
},
{
  "name": "___2",
  "type": "input",
  "label": "    ",
  "mask": "9999-9999-9999-9999",
  "required": true,
  "description": "     "
},
{
  "name": "__1",
  "type": "input",
  "label": " ",
  "validationRegExp": "^[a-zA-Z]+$",
  "errorMessage": " "
},
{
  "name": "__1",
  "type": "number_input",
  "label": "  ",
  "description": "   "
},
{
  "name": "__1",
  "type": "textarea",
  "label": "  ",
  "description": "   "
},
{
  "name": "__2",
  "type": "textarea",
  "minRows": 5,
  "maxRows": 8
},
{
  "name": "__1",
  "type": "checkbox_group",
  "required": true,
  "label": "   ",
  "description": "    ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2",
      "disabled": true
    },
    {
      "value": "_3"
```

```
        }
      ]
    },
    {
      "name": "__1",
      "type": "radio_group",
      "required": true,
      "label": "  ",
      "items": [
        "_1",
        "_2",
        "_3"
      ]
    },
    {
      "name": "__2",
      "type": "radio_group",
      "required": true,
      "label": "  ",
      "description": "   ",
      "items": [
        {
          "value": "_1",
          "label": "_1"
        },
        {
          "value": "_2",
          "label": "_2",
          "disabled": true
        },
        {
          "value": "_3"
        }
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,            .",
        ",       react-select  downshift"
      ]
    },     {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,            .",
        ",       react-select  downshift"
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": "             :           , ,       .       ,      , ,            ."
    }
  ]
}
```

Эти настройки содержат:

- **regexFunctions** (object) (необязательно) - специальный объект, который содержит пары "*key" "value*" для указания валидаторов пользовательских полей, где "*key"* - это имя вашей пользовательской функции, которое можно использовать в качестве ссылки в опции "***regexFnc***" настроек полей в "*categories*". "*value"* - это ваша пользовательская лямбда-функция JavaScript.

- **taskInstructionText** (string) (необязательно) - представляет текст инструкций, который появляется во всплывающем окне.
- **taskInstructionLink** (string) (необязательно) - представляет ссылку на удаленный источник с инструкцией. Наличие хотя бы одного из этих полей ***taskInstructionText*** или ***taskInstructionLink*** вызовет появление кнопки Инструкция.
- **taskTypeLabel** (string) (необязательно) - позволяет настроить название задачи. По умолчанию для него установлено значение "*Document Classification*".
- **autoSave** (boolean) (необязательно) - позволяет автоматически сохранять промежуточные результаты задач. По умолчанию этот параметр имеет значение "*false*".
- **allowCustomValue** (boolean) (необязательно) - позволяет вводить пользовательское значение без какого-либо подключения к изображению входного документа. Эта возможность полезна, когда *OCR* не удалось извлечь некоторый текст из документа.
- **appLanguage** (string) (необязательно) - позволяет пользователю настроить локализацию Пользовательской задачи. В настоящее время доступны опции "*en*" и "*ru*". По умолчанию отображается значение "*en*".
- **excludeUndefinedEntities** (boolean) (необязательно) - позволяет получать выходные данные только тех полей, которые настроены в параметре "*categories*". По умолчанию параметр имеет значение "*false*".
- **commentsSectionName** (string) (необязательно) - используется для переименования вкладки "***Подробности***". По умолчанию этот параметр имеет значение "*Подробности*".
- **categories** (list of objects) (обязательно) - список полей для извлечения из документа, где каждый элемент содержит:
    - **name** (string) (обязательно) - он служит ключом к получению данных из выходных данных пользовательской задачи.
    - **multiple** (boolean) (обязательно) - показывает, может ли это поле иметь несколько значений (используется, когда у вас есть список с подробными сведениями о некоторых элементах в документе, например, список продуктов).
    - **required** (boolean) (необязательно) - показывает, обязательно ли поле для извлечения, поэтому человек не сможет отправить задачу без указания значений для всех обязательных полей.
    - **helperText** (string) (необязательно) - дополнительный текст, который отображается, если значение недопустимо из-за дополнительных валидаторов, заданных параметрами *regexp*.
    - **regex** (string) (необязательно) - указывает функцию *regexp*, используемую для проверки значений.
    - **regexFnc** (list of string) (необязательно) - список названий функций валидатора, которые задаются в "***regexFunctions***".
    - **iconType** (string) (необязательно) - позволяет настроить иконку поля. В настоящее время доступны следующие варианты: *"date", "money", "multiple", "text"*. По умолчанию используется значок *"text"*.
    - **hotkey** (list of string) (необязательно) - список сочетаний клавиш, которые можно нажать, чтобы выбрать элемент в категории.
- **metadata** (list of objects) (необязательно) - используется для настройки вкладки "***Подробности***" в Рабочем пространстве и представления документов в Пакетах документов. Вкладка "***Подробности***" отключена, если ***metadata*** не определена. Флажок "Invalid Document" на вкладке "***Подробности***" можно настроить, но нельзя удалить. Если установлен флажок "Invalid Document", все проверки будут отключены.

Установленный по умолчанию флажок *"Invalid Document"* может быть настроен при помощи следующих настроек:

- **name** (string) (обязательно) - используется для объявления раздела, который используется для настройки названия и описания флажка "Invalid Document". Значением по умолчанию является "*isInvalid*" и его нельзя изменить, при его изменении параметры ***markLabel*** и ***description*** будут иметь значения по умолчанию.
- **markLabel** (string) (необязательно) - используется для определения имени флажка по умолчанию на вкладке "***Подробности***". По умолчанию этот параметр имеет значение *"Invalid Document"*.
- **description** (string) (необязательно) - позволяет добавить описание к флажку "*Invalid Document*" по умолчанию. По умолчанию описание не отображается.

Дополнительные поля можно добавить на вкладку ***"Подробности"*** с помощью следующих настроек:

- **name** (string) (обязательно) - он служит ключом к получению данных из вкладки ***"Подробности"*** из выходных данных Пользовательской задачи. Требуется для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **label** (string) (необязательно) - используется для задания названия поля.
- **type** (string) (обязательно) - задает тип поля. Должен быть один из поддерживаемых типов: ***input***, ***number_input***, ***textarea***, ***date***, ***radio_group***, ***select***, ***checkbox***, ***checkbox_group***, ***info***, ***date***.
- **required** (boolean) (необязательно) - показывает, является ли поле обязательным для заполнения во вкладке ***"Подробности"***, чтобы пользователь не сможг отправить Пользовательскую задачу без указания значений для всех обязательных полей на вкладке ***"Подробности"***, за исключением случая, когда установлен флажок *"Invalid Document"*.
- **multiple** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет выбрать несколько элементов в выпадающем списке. Может работать с настройкой ***autocomplete***.
- **autocomplete** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет осуществлять поиск по элементам в выпадающем списке. Может работать с настройкой ***multiple***.
- **mask** (string) (необязательно) - используется только если ***"type": "input"***. Позволяет задать строку символов, указывающую формат допустимых входных значений. Формат символов по умолчанию: "9" для символов *0-9*; "a" для символов *A-Z* и *a-z*; "" для симвоов *A-Z*, *a-z*, *0-9*. Если требуется иметь именно символ "9" (например, телефонный код), используйте "\\" перед символом (например, "+4\\9 99 999 99"). Если ***"required": true*** и ***mask*** установлена для ***"type": "input"***, входные данные должны быть заполнены полностью в соответствии с ***mask***.

- **minRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает минимальное количество строк, отображаемых в ***textarea***. По умолчанию этот параметр имеет значение "2".
- **maxRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает максимальное количество строк, отображаемых в текстовой области без появления прокрутки. Если количество строк больше параметра ***maxRows***, появится прокрутка. Значение по умолчанию отсутствует, и поле растет без ограничений.
- **disabled** (boolean) (необязательно) - предоставляет возможность отключения поля, тем самым не позволяет пользователю заполнить это поле.
- **validationRegExp** (string) (необязательно) -предоставляет возможность проверки заполненных значений с помощью регулярных выражений.
- **description** (string) (необязательно) - позволяет добавить описание поля. Может быть добавлено для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **errorMessage** (string) (необязательно) - используется для задания текста ошибки, если заполненные значения не соответствуют регулярному выражению в параметре ***validationRegExp***.
- **items** (list of objects) (обязательно только для "type": "checkbox_group", "radio_group", "select") - используется только если **"type": "checkbox_group", "radio_group","select"**. Задает возможные элементы для проверки. Каждый элемент в списке имеет следующие свойства:
  - **value** (string) (обязательно) - значение отображаемого элемента.
  - **label** (string) (необязательно) - метка отображаемого элемента. Если ***label*** не задан, вместо него отображается ***value***. Если установлены оба параметра, ***label*** отображается в разделе Пользовавтельская задача, а ***value*** задается в выходных данных Пользовательской задачи.
  - **disabled** (boolean) (необязательно) - предоставляет возможность отключить элемент, что помешает пользователю выбрать его.
- **markLabel** (string) (необязательно) - используется только если ***"type": "checkbox"***. Отображает метку флажка.
- **text** (string or array) (необязательно) - используется только если ***"type": "info"***. Отображает нередактируемый текст для информационного поля. Если текст содержит массив строк, каждая строка будет рассматриваться как новый абзац.
- **keyboard** (boolean) (необязательно) - используется только если ***"type": "date"***. Позволяет включить ручной ввод даты для формы ***datepicker***. По умолчанию параметр имеет значение *"false"*.
- **format** (string) (необязательно) - используется только если ***"type": "date"***. Позволяет установить пользовательский формат даты. По умолчанию этот параметр имеет значение "MM/DD/YYYY". Поддерживает следующие токены форматирования для дат:

Таблица форматов дат

	Токен	Выходные данные
Месяц	M	1 2 ... 11 12
	MM	01 02 ... 11 12
	MMM	Янв Фев ... Ноя Дек
	MMMM	Январь Февраль ... Ноябрь Декабрь
День	D	1 2 ... 30 31
	DD	01 02 ... 30 31
Год	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030

Горячие клавиши

Задача «Извлечение информации» имеет следующие горячие клавиши:

- **Tab** - для перехода к следующему полю на правой панели.
- **Shift + Tab** - для перехода к предыдущему полю на правой панели.
- **Shift + выделение области** в документе – чтобы добавить выделенные слова в конец текущего поля.
- **Shift + Enter** - добавить разрыв строки в режиме редактирования.
- **Двойной щелчок** по полю в документе, чтобы применить тег к текущему полю.
- **Ctrl + Z** - отмена последнего действия.
- **Ctrl + Shift + Z** - вернуть последнее отмененное действие обратно.
- **Ctrl + Колесо мыши** - увеличить/уменьшить масштаб документа.

Правила для настройки горячих клавиш:

- не должны быть заняты;

- не должны повторяться;
- не должны содержать несколько букв или цифр;
- должны содержать только существующие клавиши клавиатуры;
- могут быть в любом регистре;
- могут состоять из одной и больше кнопок.

Входные данные в виде JSON для обработанных файлов HTML

Представляет обработанный документ HTML в формате JSON. Генерируется автоматически при перезапуске документа:

Пример ввода JSON для HTML-документов

```
{
    "html": "<HTML >"
}
```

Входной JSON для текстовых документов содержит:

- **html** (string) - HTML для обработки. Обратите внимание, что HTML не должен содержать внешних ссылок на стили или ресурсы.

Во входном JSON можно использовать следующие дополнительные настройки:

- **overrides** (необязательно) - позволяет изменить поле и его параметры из настроек типа документа, если оно используется в качестве ввода по умолчанию. Поле, которое нужно изменить, должно быть объявлено в блоке по его имени. Затем необходимо объявить и определить настройки изменяемого поля. Обратите внимание, что «имя» и «тип» не могут быть изменены.

"overrides": {
"ошибка_ввода_1": {
"label": "Обновленная обычная ошибка ввода"
}
} - параметр "label" для поля с параметром "name": "ошибка_ввода_1" будет измнен на установленное значение.

- **messages** (необязательно) - позволяет отобразить блок с сообщением или несколькими сообщениями. Блок можно настроить с различными параметрами серьезности: "**info**", "**warning**", "**error**".
  - **"messages": ["Информационное сообщение"]** - отобразить блок с сообщением «Информационное сообщение» с уровнем серьезности по умолчанию «info»;
  - **"messages": [{ "severity": "info", "text": "Информационное сообщение" }]** - отобразить информационное сообщение "Информационное сообщение";
  - **"messages": [{ "severity": "warning", "text": "Предупреждающее сообщение" }]** - отобразить предупреждающее сообщение "Предупреждающее сообщение";
  - **"messages": [{ "severity": "error", "text": "Сообщение об ошибке" }]** - отобразить сообщение об ошибке "Сообщение об ошибке";
  - **"messages": [{**
    "severity": "error",
    "text": "Сообщение об ошибке"
    },
    {
    "severity": "info",
    "text": "Информационное сообщение"
    },
    {
    "severity": "warning",
    "text": "Предупреждающее сообщение"
    }
    ], - чтобы отобразить несколько сообщений разной степени серьезности.

Результат работы пользователя в виде JSON для обработанных HTML

В результате работы над пользовательской задачей по извлечению информации из HTML создается следующий JSON:

Пример вывода JSON для HTML-документов


```
{
  "html": "<HTML          >",
  "metadata": {
    "__10": "22  2022",
    "__1": "test",
    "__1": "07.16.2022",
    "__9": " 24",
    "__1": [
      "_3",
      "_1"
    ],
    "_2": "__2",
    "__2": [
      "_3",
      "_1"
    ],
    "__1": "-1",
    "__1": "_1",
    "__1": [
      "_1"
    ],
    "___1": "+7 (222) 222-22-22"
  }
}
```


Он имеет следующую структуру:

- **html** (string) - HTML с добавленными метками к значениям, помеченным пользователем/моделью.
- **metadata** (list of objects) - существует только в том случае, если поля из раздела «Подробности/переименованные метаданные» были заполнены. Представлен в виде структуры ключ-значение, где ключ — это имя поля, определенное в настройках типа документа, а значение — это значение, помещенное в поле.

Пользовательская задача по классификации документов

Пользовательская задача по классификации документов используется для отнесения документа к одному или нескольким классам или категориям.

Пользовательская задача по классификации документов может быть настроена для **классификации с одной меткой** (1 документ относится только к одной категории) или **классификации с несколькими метками** (1 документ может относиться к нескольким категориям)

Пользовательская задача по классификации документов может использоваться для:

- Сохранения и обработки категории класифицированной пользователем в автоматизироавнном процессе
- Сбор тренировочного набора для обучения модели машинного обучения
- Проверки данных, извлеченных моделью машинного обучения

JSON-структура типа документа

Ниже приведен пример JSON настроек для задачи классификации:

Пример JSON настроек задачи классификации

```
{
  "autoSave": false,
  "taskInstructionText": "     ,   .",
  "taskInstructionLink": "https://172.20.194.57:8444/minio/data/ht_articles_classification/instruction.html",
  "taskTypeLabel": " ",
  "appLanguage": "ru",
  "multipleChoice": false,
  "metadata": [
     {
      "name": "isInvalid",
      "markLabel": " ",
      "description":" ,     ."
     },
     {
      "name": "___12",
      "type": "input",
      "label": "   ",
      "mask": "+4\\9 99 999 99"
    },
    {
      "name": "__1",
      "type": "input",
      "label": "  ",
      "validationRegExp": "^[a-zA-Z]+$",
      "errorMessage": " "
    },
    {
      "name": "__10",
      "label": "   ",
      "type": "date",
      "format": "D MMM YYYY"
    },
    {
      "name": "__9",
      "label": "  ",
      "type": "date",
```

```
  "format": "MMMM DD"
},
{
  "name": "__8",
  "label": " ",
  "type": "date",
  "format": "YYYY"
},
{
  "name": "__7",
  "label": " ",
  "type": "date",
  "keyboard": true,
  "format": "YYYY"
},
{
  "name": "__6",
  "label": "_",
  "type": "date",
  "keyboard": true,
  "format": "MM"
},
{
  "name": "__5",
  "label": " .",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM"
},
{
  "name": "__4",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM.YY"
},
{
  "name": "__3",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "DD.MM.YYYY"
},
{
  "name": "__2",
  "label": " --",
  "type": "date",
  "keyboard": true,
  "format": "MM-DD-YYYY"
},
{
  "name": "__1",
  "label": " ..",
  "type": "date",
  "keyboard": true,
  "format": "MM.DD.YYYY",
  "description": "  .."
},
{
  "name": "__1",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
```

```
  "name": "__2",
  "type": "select",
  "required": true,
  "multiple": true,
  "label": "  ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},     {
  "name": "_1",
  "type": "select",
  "required": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "_2",
  "type": "select",
  "required": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "__1",
      "label": "_1"
    },
    {
      "value": "__2",
      "label": "_2"
    },
    {
      "value": "__3"
    }
  ]
},
{
  "name": "__1",
  "type": "select",
  "required": true,
  "multiple": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "__2",
  "type": "select",
  "required": true,
  "multiple": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
```

```
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},
{
  "name": "_1",
  "type": "select",
  "required": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "_2",
  "type": "select",
  "required": true,
  "autocomplete": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2"
    },
    {
      "value": "_3"
    }
  ]
},
{
  "name": "_1",
  "type": "checkbox",
  "markLabel": " ",
  "label": " ",
  "description": "  "
},
{
  "name": "___1",
  "type": "input",
  "label": "   ",
  "mask": "+7 (999) 999-99-99"
},
{
  "name": "___2",
  "type": "input",
  "label": "    ",
  "mask": "9999-9999-9999-9999",
  "required": true,
  "description": "     "
},
{
  "name": "__1",
```

```
  "type": "input",
  "label": " ",
  "validationRegExp": "^[a-zA-Z]+$",
  "errorMessage": " "
},
{
  "name": "__1",
  "type": "number_input",
  "label": "  ",
  "description": "   "
},
{
  "name": "__1",
  "type": "textarea",
  "label": "  ",
  "description": "   "
},
{
  "name": "__2",
  "type": "textarea",
  "minRows": 5,
  "maxRows": 8
},
{
  "name": "__1",
  "type": "checkbox_group",
  "required": true,
  "label": "   ",
  "description": "    ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2",
      "disabled": true
    },
    {
      "value": "_3"
    }
  ]
},
{
  "name": "__1",
  "type": "radio_group",
  "required": true,
  "label": "  ",
  "items": [
    "_1",
    "_2",
    "_3"
  ]
},
{
  "name": "__2",
  "type": "radio_group",
  "required": true,
  "label": "  ",
  "description": "   ",
  "items": [
    {
      "value": "_1",
      "label": "_1"
    },
    {
      "value": "_2",
      "label": "_2",
      "disabled": true
    },
```

```
        {
          "value": "_3"
        }
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,           .",
        ",       react-select  downshift"
      ]
    },     {
      "type": "info",
      "label": "   ",
      "text": [
        "             :",
        "1.          , ,       .",
        "2.      ,      , ,           .",
        ",       react-select  downshift"
      ]
    },
    {
      "type": "info",
      "label": "   ",
      "text": "             :           , ,       .       ,      , ,           ."
    }   ],
  "categories": [
    "",
    "",
    "",
    ""
],
  "scoreThreshold": 0.1
}
```

- Эти настройки содержат:
  - **autoSave** (boolean) (необязательно) - позволяет автоматически сохранять промежуточные результаты задач. По умолчанию этот параметр имеет значение "*false*".
  - **taskInstructionText** (string) (необязательно) - представляет текст инструкций, который появится во всплывающем окне.
  - **taskInstructionLink** (string) (необязательно) - представляет ссылку на удаленный источник с инструкцией. Наличие хотя бы одного из этих полей ***taskInstructionText*** или ***taskInstructionLink*** вызовет появление кнопки Инструкция.
  - **taskTypeLabel** (string) (необязательно) - позволяет настроить название задачи. По умолчанию для него установлено значение "*Document Classification*".
  - **appLanguage** (string) (необязательно) - позволяет пользователю настроить локализацию Пользовательской задачи. В настоящее время доступны опции "*en*" и "*ru*". По умолчанию отображается значение "*en*".
  - **multipleChoice** (boolean) (необязательно) - позволяет включить режим множественного выбора, где вы можете выбрать несколько категорий вместо одной. По умолчанию этот параметр имеет значение "*false*".
  - **commentsSectionName** (string) (необязательно) - используется для переименования вкладки "***Подробности***". По умолчанию этот параметр имеет значение "*Подробности*".
  - **categories** (list of strings) (обязательно) - список предопределенных классов, к которым могут относиться ваши документы.
  - **scoreThreshold** (decimal) (необязательно) - используется если ***"multipleChoice": true***, чтобы автоматически выбрать категории после Машинного обучения. Когда ***"multipleChoice": false***, то ***scoreThreshold*** не имеет значения, так как автоматически выбирается категория с наибольшим баллом.
  - **metadata** (list of objects) (необязательно) - используется для настройки вкладки "***Подробности***" в Рабочем пространстве и представления документов в Пакетах документов. Вкладка "***Подробности***" отключена, если ***metadata*** не определена. Флажок "Invalid Document" на вкладке "***Подробности***" можно настроить, но нельзя удалить. Если установлен флажок "Invalid Document", все проверки будут отключены.

Установленный по умолчанию флажок *"Invalid Document"* может быть настроен при помощи следующих настроек:

- **name** (string) (обязательно) - используется для объявления раздела, который используется для настройки названия и описания флажка "Invalid Document". Значением по умолчанию является "*isI nvalid*" и его нельзя изменить, при его изменении параметры ***markLabel*** и ***description*** будут иметь значения по умолчанию.
- **markLabel** (string) (необязательно) - используется для определения имени флажка по умолчанию на вкладке "***Подробности***". По умолчанию этот параметр имеет значение *"Invalid Document".*
- **description** (string) (необязательно) - позволяет добавить описание к флажку "*Invalid Document*" по умолчанию. По умолчанию описание не отображается.

Дополнительные поля можно добавить на вкладку ***"Подробности"*** с помощью следующих настроек:

- **name** (string) (обязательно) - он служит ключом к получению данных из вкладки ***"Подробности"*** из выходных данных Пользовательской задачи. Требуется для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **label** (string) (необязательно) - используется для задания названия поля.
- **type** (string) (обязательно) - задает тип поля. Должен быть один из поддерживаемых типов: ***input***, ***number_input***, ***textarea***, ***date***, ***radio_group***, ***select***, ***checkbox***, ***checkbox_group***, ***info***, ***date***.
- **required** (boolean) (необязательно) - показывает, является ли поле обязательным для заполнения во вкладке ***"Подробности"***, чтобы пользователь не сможг отправить Пользовательскую задачу без указания значений для всех обязательных полей на вкладке ***"Подробности"***, за исключением случая, когда установлен флажок *"Invalid Document".*
- **multiple** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет выбрать несколько элементов в выпадающем списке. Может работать с настройкой ***autocomplete***.
- **autocomplete** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет осуществлять поиск по элементам в выпадающем списке. Может работать с настройкой ***multiple***.
- **mask** (string) (необязательно) - используется только если ***"type": "input"***. Позволяет задать строку символов, указывающую формат допустимых входных значений. Формат символов по умолчанию: "9" для символов *0-9*; "a" для символов *A-Z* и *a-z*; "" для симвоов *A-Z*, *a-z*, *0-9*. Если требуется иметь именно символ "9" (например, телефонный код), используйте "\\" перед символом (например, "+4\\9 99 999 99"). Если ***"required": true*** и ***mask*** установлена для ***"type": "input"***, входные данные должны быть заполнены полностью в соответствии с ***mask***.
- **minRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает минимальное количество строк, отображаемых в ***textarea***. По умолчанию этот параметр имеет значение "2".
- **maxRows** (number) (необязательно) - используется только если **"type": "textarea"**. Задает максимальное количество строк, отображаемых в текстовой области без появления прокрутки. Если количество строк больше параметра ***maxRows***, появится прокрутка. Значение по умолчанию отсутствует, и поле растет без ограничений.
- **disabled** (boolean) (необязательно) - предоставляет возможность отключения поля, тем самым не позволяет пользователю заполнить это поле.
- **validationRegExp** (string) (необязательно) -предоставляет возможность проверки заполненных значений с помощью регулярных выражений.
- **description** (string) (необязательно) - позволяет добавить описание поля. Может быть добавлено для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
- **errorMessage** (string) (необязательно) - используется для задания текста ошибки, если заполненные значения не соответствуют регулярному выражению в параметре ***validationRegExp***.
- **items** (list of objects) (обязательно только для "type": "checkbox_group", "radio_group", "select") - используется только если **"type": "checkbox_group", "radio_group","select"**. Задает возможные элементы для проверки. Каждый элемент в списке имеет следующие свойства:
  - **value** (string) (обязательно) - значение отображаемого элемента.
  - **label** (string) (необязательно) - метка отображаемого элемента. Если ***label*** не задан, вместо него отображается ***value***. Если установлены оба параметра, ***label*** отображается в разделе Пользовавтельская задача, а ***value*** задается в выходных данных Пользовательской задачи.
  - **disabled** (boolean) (необязательно) - предоставляет возможность отключить элемент, что помешает пользователю выбрать его.
- **markLabel** (string) (необязательно) - используется только если ***"type": "checkbox"***. Отображает метку флажка.
- **text** (string or array) (необязательно) - используется только если ***"type": "info"***. Отображает нередактируемый текст для информационного поля. Если текст содержит массив строк, каждая строка будет рассматриваться как новый абзац.
- **keyboard** (boolean) (необязательно) - используется только если ***"type": "date"***. Позволяет включить ручной ввод даты для формы ***datepicker***. По умолчанию параметр имеет значение "*false*".
- **format** (string) (необязательно) - используется только если ***"type": "date"***. Позволяет установить пользовательский формат даты. По умолчанию этот параметр имеет значение "MM/DD /YYYY". Поддерживает следующие токены форматирования для дат:

	Токен	Выходные данные

Месяц	M	1 2 ... 11 12
	MM	01 02 ... 11 12
	MMM	Янв Фев ... Ноя Дек
	MMMM	Январь Февраль ... Ноябрь Декабрь
День	D	1 2 ... 30 31
	DD	01 02 ... 30 31
Год	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030

JSON-структура входных данных PDF

Входные данные для пользовательской задачи классификации документов представляют собой JSON со структурой, основанной на типе обрабатываемого документа:

Входные данные JSON для обработанных файлов PDF

Представляет результат распознавания документа в формате JSON, наложенный на исходное изображение документа. Генерируется автоматически при перезапуске OCR:

Пример входных данных JSON для PDF-документов

```
{
        "images": [ // There should be an array to support multipage PDFs. Each page should be defined
individually in this case.
                {
                        "content": "https://some-provider.com/img.jpg",
                    "tessinput": "https://some-provider.com/tessinput.jpg"
                    "json": <OCR-JSON>,
                    "dimensions": { // The dimensions of the page in pixels.
                            "width": 2000,
                            "height": 3000
                    }
                }
        ]
}
```

- Входной JSON содержит:
    - **images** (list of objects) - корневой элемент, содержащий список конфигураций документа. Список обычно содержит только один элемент.
        - **content** (url or base64 string) - источник входного документа для отображения. Это может быть **URL** документа или содержание документа, закодированное в base64 (например, строковое значение "....").
        - **json** (**OCR-JSON** object) - предоставляет информацию OCR. Структура OCR-JSON описана ниже.
        - **dimensions** (object) - объект содержит параметры «**width**» и «**height**», которые представляют ширину и высоту исходного входного документа.
            - **width** (integer) - значение ширины исходного входного документа.
            - **height** (integer) - значение высоты исходного входного документа.

OCR-JSON имеет следующую структуру, которая генерируется компонентом OCR самостоятельно:

OCR-JSON

```
"json": {
        "pages": [
                "id": "page0",
                "areas": [
                        {
                                "id": "page0_area0",
```

```
                        "paragraphs": [
                                {
                                        "id": "page0_area0_paragraph0",
                                        "lines": [
                                                {
                                                        "id": "page0_area0_paragraph0_line0",
                                                        "words": [
                                                                {
                                                                        "id":
"page0_area0_paragraph0_line0_word0",
                                                                        "text": "Advanced",
                                                                        "properties": {
                                                                                "bbox": [

0.05999032414126754,

0.037290455011974,

0.14078374455732948,

0.046527540198426275
                                                                                ],
                                                                        "x_fsize": 0,
                                                                        "x_wconf": 96
                                                                }
                                                        },
                                                }
                                        ...
                                        ]
                                }
                        ]
                }
        ]
}
```


Этот JSON содержит все слова, подвергнутые распознаванию, и имеет древовидную структуру: **pages** **areas** **paragraphs** **lines** **words**.

- **json** (object) - корневой элемент
  - **pages** (list of objects) - структура списка страниц. Каждая страница в списке имеет следующую структуру:
    - **id** (string) - ID страницы
    - **areas** (list of objects) - структура списка областей. Каждая область в списке имеет следующую структуру:
      - **id** (string) - ID области
      - **paragraphs** (list of objects) - структура списка абзацев. Каждый абзац в списке имеет следующую структуру:
        - **id** (string) - ID параграфа
        - **lines** (list of objects) - структура списка строк. Каждая строка в списке имеет следующую структуру:
          - **id** (string) - ID строки
          - **words** (list of objects) - структура списка слов. Каждое слово в списке имеет следующую структуру:
            - **id** (string) - ID слова
            - **text** (string) - исходный текст, извлеченный механизмом OCR
            - **properties** (object) - объект свойства со следующей структурой:
              - **bbox** (list of integers) - верхняя левая и нижняя правая координаты прямоугольника вокруг слова в исходном документе. Координаты нормализуются от 0 до 1 относительно исходного размера документа
            - **x_fsize** (integer) - это размер шрифта, специфичный для механизма OCR
            - **x_wconf** (integer) - Достоверность работы OCR для всей подстроки. Более высокие значения выражают более высокую достоверность.

Входные данные JSON для текстовых документов

Представляет текст документа в формате JSON и имеет следующую структуру:

Пример ввода JSON для текстовых документов

```
{
        "text": [
                "Lorem Ipsum is simply dummy text of the printing and typesetting industry.",
                "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown
printer took a galley of type and scrambled it to make a type specimen book.",
                "It has survived not only five centuries, but also the leap into electronic typesetting,
remaining essentially unchanged.",
                "It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum
passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem
Ipsum."
        ]
}
```

Входной JSON для текстовых документов содержит:

- **text** (list of strings) - текст для классификации. Добавление нескольких строк в список приведет к разделению текста на абзацы в пользовательской задаче.

Во входном JSON можно использовать следующие дополнительные настройки:

- **overrides** (необязательно) - позволяет изменить поле и его параметры из настроек типа документа, если оно используется в качестве ввода по умолчанию. Поле, которое нужно изменить, должно быть объявлено в блоке по его имени. Затем необходимо объявить и определить настройки изменяемого поля. Обратите внимание, что «имя» и «тип» не могут быть изменены

"overrides": {
"ошибка_ввода_1": {
"label": "Обновленная обычная ошибка ввода"
}
} - параметр "label" для поля с параметром "name": "ошибка_ввода_1" будет измнен на установленное значение.

- **messages** (необязательно) - позволяет отобразить блок с сообщением или несколькими сообщениями. Блок можно настроить с различными параметрами серьезности: "**info**", "**warning**", "**error**".
    - **"messages": ["Информационное сообщение"]** - отобразить блок с сообщением «Информационное сообщение» с уровнем серьезности по умолчанию «info»;
    - **"messages": [{ "severity": "info", "text": "Информационное сообщение" }]** - отобразить информационное сообщение "Информационное сообщение";
    - **"messages": [{ "severity": "warning", "text": "Предупреждающее сообщение" }]** - отобразить предупреждающее сообщение "Предупреждающее сообщение";
    - **"messages": [{ "severity": "error", "text": "Сообщение об ошибке" }]** - отобразить сообщение об ошибке "Сообщение об ошибке";
    - **"messages": [{**
      "severity": "error",
      "text": "Сообщение об ошибке"
      },
      {
      "severity": "info",
      "text": "Информационное сообщение"
      },
      {
      "severity": "warning",
      "text": "Предупреждающее сообщение"
      }
      ], - чтобы отобразить несколько сообщений разной степени серьезности.

JSON-структура результатов обработки

В качестве выходных данных пользовательская задача по классификации документовt создает следующий JSON:

Пример результатов JSON

```
{
  "categories": [
    ""
```

```
  ],
  "scores": {
    "": 0.511058509349823,
    "": 0.3745160043239594,
    "": 0.06,
    "": 0.03
  },
  "metadata": {
    "__1": "test",
    "__10": "22  2022",
    "__4": "16.07.22",
    "__1": [
      "_1",
      "_3"
    ],
    "__2": [
      "_2"
    ],
    "_1": "_1",
    "_2": "__3",
    "__1": [
      "_2"
    ],
    "_2": "_1",
    "_1": "_2",
    "___2": "1111-1111-1111-1111",
    "__1": "_3",
    "__2": "_1"
  }
}
```

Он имеет следующую структуру:

- **categories** (list of strings) - категории, к которым относится входной документ. Результат может быть предоставлен либо моделью машинного обучения, либо пользователем. В списке может быть несколько значений, если для Типа документа значение MultipleChoice равно true, и только 1 значение, если оно равно false.
- **scores** (list of objects) - существует только в том случае, если результаты классификации предоставляются моделью машинного обучения. Значение оценки каждой категории представлено в виде структуры ключ-значение, где ключ — это имя категории, а значение — десятичное значение, представляющее оценку от 0 до 1.
- **metadata** (list of objects) - существует только в том случае, если поля из раздела «Подробности/переименованные метаданные» были заполнены. Представлен в виде структуры ключ-значение, где ключ — это имя поля, определенное в настройках типа документа, а значение — это значение, помещенное в поле.

Пользовательская задача заполнения формы

Пользовательская задача заполнения формы предоставляет возможность настроить пользовательскую задачу, которая содержит несколько элементов формы. Задача поддерживает следующий набор полей:

- Простой текст (однострочное поле ввода) (тип **input**);
- Многострочный текст (многострочное поле ввода) (тип **textarea**);
- Выбор даты (тип **date**);
- Радиокнопки (элементы одиночного выбора) (тип **radio_group**);
- Выбер выпадающих списков (один выбор из списка значений, лучше, если у вас есть длинный список для выбора) (тип **select**);
- Флажок для множественного выбора (множественный выбор из списка значений) (тип **checkbox_group**);
- Флажок для одиночного выбора (тип **checkbox**);
- Номер (тип **number_input**);
- Информационное поле (тип **info**)

Список вопросов можно разделить на отдельные блоки с помощью групп. Они задаются в настройках как массив полей ввода.

JSON-структура типа документа формы

Вот пример настроек, определяющего форму:

Пример JSON настроек типа документа формы

```
{
  "appLanguage": "ru",
  "groups": [
    {
      "groupTitle": " 1",
      "fields": [
        {
          "name": "Email",
          "label": "  email",
          "type": "input",
          "required": true
        },
        {
          "name": "",
          "label": " ",
          "type": "select",
          "required": true,
          "items": [
            "",
            "",
            ""
          ]
        },
        {
          "name": " ",
          "label": " ",
          "type": "date",
          "required": true
        },
        {
          "name": "1",
          "label": " ",
          "type": "checkbox_group",
          "required": true,
          "disabled": false,
          "items": [
            {
              "value": "_1",
```

```
            "label": "1",
            "disabled": false
          },
          {
            "value": "_2",
            "label": "2",
            "disabled": false
          },
          {
            "value": "_3",
            "label": "3",
            "disabled": true
          }
        ]
      },
      {
        "name": "1",
        "label": "   ",
        "type": "textarea",
        "required": true,
        "disabled": false
      },
      {
        "name": "",
        "label": "",
        "type": "radio_group",
        "required": false,
        "items": [
          {
            "value": "",
            "label": "",
            "disabled": false
          },
          {
            "value": "",
            "label": "",
            "disabled": false
          },
          {
            "value": "",
            "label": "",
            "disabled": true
          }
        ]
      }
    ]
  },
  {
    "groupTitle": " 2",
    "fields": [
      {
        "name": "",
        "label": "  ",
        "type": "input",
        "required": true,
        "validationRegExp": "[A-Za-z]+"
      },
      {
        "name": " ",
        "label": " ",
        "type": "date",
        "required": true
      },
      {
        "name": "2",
        "markLabel": "",
        "type": "checkbox",
        "required": false,
        "disabled": true,
         "label": " "
      }
```

```
      ]
    }
  ]
}
```

Эти настройки содержат:

- **autoSave** (boolean) (необязательно) - позволяет автоматически сохранять промежуточные результаты задач. По умолчанию этот параметр имеет значение "*false*".
- **appLanguage** (string) (необязательно) - позволяет пользователю настроить локализацию пользовательской задачи. В настоящее время доступны опции "*en*" и "*ru*". По умолчанию отображается значение "*en*".
- **showOrderNumber** (boolean) (необязательно) - позволяет отобразить положение элемента в группе. По умолчанию этот параметр имеет значение "*false*".
- **groups** (list of objects) (обязательно) - список настроек группы. Каждая группа в списке имеет следующую структуру:
  - **groupTitle** (string) (обязательно) - определяет название группы, отображаемое в пользовательской задаче.
  - **fields** (list of objects) (обязательно) - список настроек полей. Каждое поле в списке имеет следующую структуру:
    - **name** (string) (обязательно) - используется в качестве ключа для получения значения результата выходных данных. Оно никогда не отображается в пользовательской задаче и **должно быть уникальным для всей формы**. Обязательно для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
    - **label** (string) (обязательно) - метка для отображения на пользовательской задаче.
    - **type** (string) (обязательно) - задает тип поля. Должен быть один из поддерживаемых типов: ***input***, ***number_input***, ***textarea***, ***date***, ***radio_group***, ***select***, ***checkbox***, ***checkbox_group***, ***info***, ***date***.
    - **required** (boolean) (необязательно) - показывает, обязательно ли поле для заполнения, поэтому пользователь не сможет отправить пользовательскую задачу без указания значений для всех обязательных полей.
    - **multiple** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет выбрать несколько элементов в выпадающем списке. Может работать с настройкой ***autocomplete***.
    - **autocomplete** (boolean) (необязательно) - используется только если ***"type": "select"***. Позволяет осуществлять поиск по элементам в выпадающем списке. Может работать с настройкой ***multiple***.
    - **mask** (string) (необязательно) - используется только если ***"type": "input"***. Позволяет задать строку символов, указывающую формат допустимых входных значений. Формат символов по умолчанию: "9" для символов *0-9*; "a" для символов *A-Z* и *a-z*; "" для симвоов *A-Z*, *a-z*, *0-9*. Если требуется иметь именно символ "9" (например, телефонный код), используйте "\\" перед символом (например, "+4\\9 99 999 99"). Если ***"required": true*** и ***mask*** установлена для ***"type": "input"***, входные данные должны быть заполнены полностью в соответствии с ***mask***.
    - **minRows** (number) (необязательно) - используется только если ***"type": "textarea"***. Задает минимальное количество строк, отображаемых в ***textarea***. По умолчанию этот параметр имеет значение "2".
    - **maxRows** (number) (необязательно) - используется только если ***"type": "textarea"***. Задает максимальное количество строк, отображаемых в текстовой области без появления прокрутки. Если количество строк больше параметра ***maxRows***, появится прокрутка. Значение по умолчанию отсутствует, и поле растет без ограничений.
    - **disabled** (boolean) (необязательно) - предоставляет возможность отключения поля, тем самым не позволяет пользователю заполнить это поле.
    - **validationRegExp** (string) (необязательно) - предоставляет возможность проверки заполненных значений с помощью регулярных выражений.
    - **description** (string) (необязательно) - позволяет добавить описание поля. Может быть добавлено для следующих типов: ***input, number_input, textarea, date, radio_group, select, checkbox, checkbox_group, date***.
    - **errorMessage** (string) (необязательно) - используется для задания текста ошибки, если заполненные значения не соответствуют регулярному выражению в параметре ***validationRegExp***.
    - **items** (list of objects) (обязательно только для "type": "checkbox_group", "radio_group", "select") - используется только если ***"type": "checkbox_group"***, **"radio_group","select"**. Задает возможные элементы для проверки. Каждый элемент в списке имеет следующие свойства:
      - **value** (string) (обязательно) - значение отображаемого элемента.
      - **label** (string) (необязательно) - метка отображаемого элемента. Если ***label*** не задан, вместо него отображается ***value***. Если установлены оба параметра, ***label*** отображается в разделе Пользовавтельская задача, а ***value*** задается в выходных данных Пользовательской задачи.
      - **disabled** (boolean) (необязательно) - предоставляет возможность отключить элемент, что помешает пользователю выбрать его.
    - **markLabel** (string) (необязательно) - используется только если ***"type": "checkbox"***. Отображает метку флажка.

- **text** (string or array) (необязательно) - используется только если ***"type": "info"***. Отображает нередактируемый текст для информационного поля. Если текст содержит массив строк, каждая строка будет рассматриваться как новый абзац.
- **keyboard** (boolean) (необязательно) - используется только если ***"type": "date"***. Позволяет включить ручной ввод даты для формы ***datepicker***. По умолчанию параметр имеет значение "false".
- **format** (string) (необязательно) - используется только если ***"type": "date"***. Позволяет установить пользовательский формат даты. По умолчанию этот параметр имеет значение "MM/DD /YYYY". Поддерживает следующие токены форматирования для дат:

Месяц	M	1 2 ... 11 12
	MM	01 02 ... 11 12
	MMM	Янв Фев ... Ноя Дек
	MMMM	Январь Февраль ... Ноябрь Декабрь
День	D	1 2 ... 30 31
	DD	01 02 ... 30 31
Год	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030

JSON-структура входных данных

Входные данные JSON содержат структуру и описание полей настраиваемой формы, которая отображается в рабочем пространстве. Их можно настроить аналогично настройкам типа документа. Если Входной JSON не заполнен, поля из настроек типа документа будут отображаться по умолчанию. Во входном JSON можно использовать следующие дополнительные настройки:

- **overrides** (необязательно) - позволяет изменить поле и его параметры из настроек типа документа, если оно используется в качестве ввода по умолчанию. Поле, которое нужно изменить, должно быть объявлено в блоке по его имени. Затем необходимо объявить и определить настройки изменяемого поля. Обратите внимание, что «имя» и «тип» не могут быть изменены

```
"overrides": {
   "ошибка_ввода_1": {
     "label": "Обновленная обычная ошибка ввода"
   }
```

  } - параметр "label" для поля с параметром "name": "ошибка_ввода_1" будет измнен на установленное значение.

- **messages** (необязательно) - позволяет отобразить блок с сообщением или несколькими сообщениями. Блок можно настроить с различными параметрами серьезности: "**info**", "**warning**", "**error**".
  - **"messages": ["Информационное сообщение"]** - отобразить блок с сообщением «Информационное сообщение» с уровнем серьезности по умолчанию «info»;
  - **"messages": [{ "severity": "info", "text": "Информационное сообщение" }]** - отобразить информационное сообщение "Информационное сообщение";
  - **"messages": [{ "severity": "warning", "text": "Предупреждающее сообщение" }]** - отобразить предупреждающее сообщение "Предупреждающее сообщение";
  - **"messages": [{ "severity": "error", "text": "Сообщение об ошибке" }]** - отобразить сообщение об ошибке "Сообщение об ошибке";
  -
```
"messages":  [{
     "severity": "error",
     "text": "Сообщение об ошибке"
   },
   {
    "severity": "info",
    "text": "Информационное сообщение"
   },
   {
    "severity": "warning",
    "text": "Предупреждающее сообщение"
   }
```

  ], - чтобы отобразить несколько сообщений разной степени серьезности.

Пример входного JSON, в котором определены разделы **override** и **messages:**

Пример JSON входных данных

```
{
  "overrides": {
    "": {
      "label": "    ",
      "required": false
    }
  },
  "messages": [
    {
      "severity": "error",
      "text": "  "
    },
    {
      "severity": "info",
      "text": " "
    },
    {
      "severity": "warning",
      "text": " "
    }
  ]
}
```

JSON-структура результатов обработки

JSON-структура результатов обработки состоит из пар ключ-значение. Ключ — это уникальное имя поля, указанное в параметрах типа документа или входном JSON формы, а значение указывает значение, выбранное во время обработки формы.

Пример JSON результатов обработки

```
{
  "Email": "test@gmail.com",
  "": "",
  " ": "07/16/2022",
  "1": [
    "_1"
  ],
  "": "",
  "": "",
  " ": "07/17/2022",
  "1": ""
}
```

Создание пользовательской задачи

Выполнение пользовательской задачи

Вы отвечаете за контроль размера пула потоков

Поскольку автоматизированный процесс не завершен после создания пользовательской задачи, он ожидает результатов выполнения пользовательской задачи для продолжения обработки. Это означает, что в это время используется Java Thread.

Вы всегда должны разрабатывать свой автоматизированный процесс, помня о том, что у вас может быть несколько активных (незавершенных) пользовательских задач, поэтому вы должны брать в расчет возможное количество незавершенных пользовательских задач.

Если все потоки из пула потоков используются, автоматизированный процесс не сможет продолжать работу с различными шагами, поскольку все потоки будут заняты пользовательскими задачами.

Есть 2 параметра автоматизированного процесса, которыми вы можете управлять:

- **executorService.poolSize** - для указания размера пула потоков
- **remoteExecutionService.taskTimeout** - чтобы указать тайм-аут для истечения срока действия пользовательских задач.

Вы можете найти значения по умолчанию на странице **Администрирование Настройки сервера**. См. Настройки сервера.

Подготовка входных данных

Перед запуском пользовательской задачи вы должны подготовить объект **HumanTaskData** и предоставить этот объект в качестве вывода из класса **Task**.

В качестве примера рассмотрим приведенный ниже Java-класс **PrepareInputApTask**. Это простая **ApTask**, которая создает **HumanTaskData**.

```
@ApTaskEntry(name = "Prepare Human Task Input")
@Slf4j
public class PrepareInputApTask extends ApTask {

        @Output(HumanTask.HUMAN_TASK_DATA_KEY)
        private HumanTaskData humanTaskData;

        @Override
        public void execute(){
                this.humanTaskData = new HumanTaskData();
                this.humanTaskData.setInputJson(new HashMap<>());
                this.humanTaskData.setDescription("Human Task Example");
                this.humanTaskData.setDocumentType("example");
                this.humanTaskData.setName("Human Task");
                this.humanTaskData.setPriority(1);

                log.info("Injecting task into workspace {} ", humanTaskData);
}
```

Выполнение пользовательской задачи

Теперь давайте посмотрим, как подходовить входные данные выполнить **HumanTask** из вашего класса **ApModule**.

```
public TaskOutput run() throws Exception {
    return execute(getInput(), PrepareInputApTask.class)
```

```
                                .thenCompose(execute(HumanTask.class))
                                .thenCompose(execute(ProcessHumanTaskResultsTask.class)).get();
}
```

Работа с результатами пользовательской задачи

Когда пользовательская задача будет завершена, вам может понадобиться обработать полученные результаты. Ниже приведен пример получения результатов пользовательской задачи и работы с ними.

```
@ApTaskEntry(name = "Working with Human Task results.")
@Slf4j
@InputToOutput
public class ProcessHumanTaskResultsTask extends ApTask {

        @Input(HumanTask.HUMAN_TASK_DATA_KEY)
        private HumanTaskData humanTaskData;

        @Override
        public void execute() throws Exception {
                log.info("Received response from Human Task {} ", humanTaskData.getTaskUuid());
                Map outputJson = humanTaskData.getOutputJson();
                //work with output..
        }
}
```

Создание типа пользовательской задачи

Что такое тип пользовательской задачи?

Тип пользовательской задачи — это веб-приложение и часть Канцлер RPA, которое предоставляет пользовательский интерфейс для взаимодействия с работниками (подрядчиками, экспертами предметной области, сотрудниками) и автоматизированными процессами.

Базовая структура

Базовый тип пользовательской задачи состоит из 3 файлов:

- index.html — содержит основную логику и структуру HTML. Этот файл является точкой входа в пользовательскую задачу. Базовый пример структуры ниже:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Human Task Example</title>
</head>
<body>
    <div id="root">
                <!-- some html structure -->
    </div>
</body>
<script>
        //scripts with some rules, logic and etc.
</script>
</html>
```

- sample.json — этот файл нужен для Песочницы, где вы можете посмотреть и протестировать свой тип пользовательской задачи.

> ⚠ Требуются свойства inputJson, outputJson и settings.

```
{
  "inputJson": {
        ...
  },
  "outputJson": {
                ...
  },
  "settings": {
    ...
  }
}
```

- package.json — этот файл нужен, чтобы узнать название и версию типа пользовательской задачи для Канцлер RPA. Основной пример ниже:

```
{
  "name": "human-task-type-example",
  "version": "0.1.0"
}
```

Интеграция с рабочим пространством

Метод **window.postMessage()** обеспечивает связь между объектами **Window**. Когда пользовательская задача создается в рабочем пространстве, вам необходимо получить входные данные json из **процесса приложения**. Входные данные будут храниться в свойстве **data** полученного сообщения. Структура входных данных аналогична описанному выше *sample.json*.

Чтобы получить входные данные, вам нужно добавить следующий обработчик события в ваш *index.html* или ваш файл скрипта.

'validation-message' означает, что рабочий выполнил пользовательскую задачу и она готова к завершению. Здесь вы можете проверить обработанные данные, добавить их на **сервер управления** и завершить **пользовательскую задачу**.

```
window.addEventListener('message', message => {
    const { data } = message; //collect input data
    if (!data['validation-message']) {
        renderApp(data);
    } else if (data['validation-message']) {
        submit(data);
    }
});
```

Когда вы получили входные данные, вы можете использовать **'inputJson'** для заполнения некоторых полей или использовать **'settings'** для сбора некоторых внутренних настроек, например: labels, regexp, list of required fields и т. д.

Ниже приведен пример использования входных настроек и inputJson для визуализации пользовательской задачи и заполнения значений в полях ввода.

```
function renderApp(data) {
        const template = document.createElement('div');
        const root = document.getElementById('root');
        root.appendChild(template);

        const input = document.getElementById('main-input');
        input.value = data.inputJson.inputText;

        const inputLabel = document.getElementById('main-label');
        inputLabel.innerHTML = data.settings.input.label;
}
```

Для отправки данных используйте следующий пример:

```
function submit() {
        const inputValue = document.getElementById('main-input').value;
        const outputJson = { text: inputValue };
        window.parent.postMessage(outputJson, '*');

        const data = {
                'validation-message': true,
                result: true,
        };
        window.parent.postMessage(data, '*');
}
```

Загрузка на сервер управления

1. Чтобы загрузить тип пользовательской задачи на сервере управления, вам необходимо создать zip-архив, содержащий все файлы приложения.
2. Откройте страницу **Типы пользовательских задач** на сервере управления.
3. Нажмите **Создать**, заполните форму и загрузите подготовленный zip-архив.
4. Нажмите **Создать**.

См. Типы пользовательских задач.

Проверьте свой собственный тип пользовательской задачи

Песочница позволяет разработчику проверить, как пользовательская задача интегрируется в рабочее пространство.

Чтобы посмотреть, как созданный тип пользовательской задачи выглядит в Песочнице, сделайте следующее:

1. Откройте страницу **Типы пользовательских задач** на сервере управления.
2. Передите в созданную пользовательскую задачу на вкладку **Песочница**.
3. Измените **Входной документ JSON** и нажмите кнопку **ОТПРАВИТЬ ВХОДНЫЕ ДАННЫЕ**. Рабочая область запустит событие с новым данными для типа пользовательской задачи.
4. Чтобы убедиться, что тип пользовательской задачи правильно отправляет полученные данные на рабочее пространство, нажмите кнопку **ПОЛУЧИТЬ ВЫХОДНЫЕ ДАННЫЕ**. После этого раздел **Полученные данные** будет немедленно изменен.

Полный пример

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Human Task Example</title>
</head>
<body>
    <div id="root">
        <div class="app">
            <label for="main-input" id="main-label"></label>
            <input type="text" id="main-input">
        </div>
    </div>
</body>
<script>
    function submit() {
        const inputValue = document.getElementById('main-input').value;
        const outputJson = { text: inputValue };
        window.parent.postMessage(outputJson, '*');

        const data = {
            'validation-message': true,
            result: true,
        };
        window.parent.postMessage(data, '*');
    }

    function renderApp(data) {
        const template = document.createElement('div');
        const root = document.getElementById('root');
        root.appendChild(template);

        const input = document.getElementById('main-input');
        input.value = data.inputJson.inputText;

        const inputLabel = document.getElementById('main-label');
        inputLabel.innerHTML = data.settings.input.label;
    }

    window.addEventListener('message', message => {
        const { data } = message;
        if (!data['validation-message']) {
            renderApp(data);
        } else if (data['validation-message']) {
            submit(data);
        }
    });
</script>
</html>
```

sample.json

```
{
  "inputJson": {
    "inputText": "Text"
  },
  "outputJson": {

  },
  "settings": {
    "input": {
      "label": "Label"
    }
  }
}
```

package.json

```
{
  "name": "human-task-type-example",
  "version": "0.1.0"
}
```

Оцифровка документов (OCR)

Канцлер RPA поддерживает автоматизацию процесса, требующего преобразования изображений печатного текста в машинно-кодированный текст.

Платформа включает встроенный механизм OCR:

- Готовое решение: нет необходимости писать или поддерживать какой-либо собственный код.
- Нет ограничений на количество страниц OCR в год.
- Простая автоматизация работы с PDF документами.
- Основан на ведущем движке OCR с открытым исходным кодом.
- Может распознавать более 100 языков.

Кроме того, Канцлер RPA поддерживает интеграцию с программным обеспечением ABBYY FlexiCapture.

- Встроенный OCR
- Задача OCR
- Модуль интеграции ABBYY FlexiCapture
- Рекомендации по OCR

Встроенный OCR

Поток OCR

Оптическое распознавание символов, обычно называемое OCR, представляет собой процесс преобразования отсканированных изображений букв и слов в их электронные версии.

ImageMagick

Первый шаг — обработка ImageMagick. Он разбивает документ на страницы и преобразует их в изображения, используя предоставленные конфигурации. Результаты сохраняются в виде файлов .jpg.

Tesseract

На следующем шаге Tesseract преобразует изображения документа в текстовый и HOCR-форматы и сохраняет результаты.

PostProcessor

После окончания преобразования Tesseract результаты обрабатываются HocrPostProcessor. Разработчик процесса может дополнительно предоставить конфигурацию, которая задает пары распознаных и правильных слов. Компонент HocrPostProcessor перебирает слова HOCR, заменяет все слова, соответствующие распознанному образцу, на соответствующее правильное слово.

Сведения об OCR Container

Входные данные

Контейнер OCR принимает в качестве входных данных сообщение JSON. Пример представлен ниже:

```
{
        "taskUuid": "41125e8d-8b9e-4e42-b0ef-739594a21c50",
        "runUuid": "5f425d18-06ac-48f8-ad2b-5af4a24daca7",
        "documentLocation": "template1.pdf",
        "formats": ["hocr", "text", "json", "image"],
        "configuration": {
                "bucket": "ocr",
                "tesseractOptions": ["-l", "deu","--psm","3","--oem","3","--dpi", "800"],
                "imageMagickOptions": ["-resample", "450", "-density", "350", "-quality", "100", "-background"
, "white" , "-alpha" , "flatten", "-colorspace", "RGB"],
                "hocrFixWords": {
                        "INVOICE": "InvoiceFixed"
                }
        }
}
```

Результат работы

OCR Container возвращает список объектов OcrOutput (по одному на страницу), которые заполняются в зависимости от входного параметра "format".

- **image** - путь к изображению и размер изображения.
- **text** - путь к текстовому файлу Tesseract.
- **hocr** - путь к файлу Tesseract Hocr.
- **json** - возвращает структуру JSON, полученную из файла Hocr с помощью HocrJsonConverter.

Задача OCR

Обзор OCR

OCR (оптическое распознавание символов) – это электронное преобразование изображений печатного, рукописного или печатного текста в машинно-кодированный текст, например из отсканированного документа, фотографии документа или текста субтитров, наложенного на изображение.

Распознавание входных документов — это отдельный этап процесса разработки — обычно это конвертация из PDF в текстовые документы. **Качество исходных документов** во многом влияет на результат этого процесса. Канцлер RPA использует Tesseract OCR, который интегрирован в отдельную задачу и предлагает готовое решение: нет необходимости писать или поддерживать какой-либо собственный код.

Использование задачи OCR

Теперь давайте дадим вам несколько шагов, необходимых для реализации этого подхода.

1. Определите **OcrTaskData** в качестве вывода в классе **Task**:

   ```
   @Output(OcrTask.OCR_TASK_DATA_KEY)
   private OcrTaskData ocrTaskData;
   ```

   OcrTask.OCR_TASK_DATA_KEY - ключ ввода/вывода данных. *OcrTask* использует этот ключ для получения входных данных для OCR.

2. Подготовьте конфигурацию OCR, указав имя корзины S3, параметры Tesseract и параметры Image Magick:

   ```
   Map configuration = new HashMap<String, Object>();
   configuration.put("bucket", "data");
   configuration.put("tesseractOptions", Arrays.asList("-l", "eng","--psm","3","--oem","3","--dpi", "800"));
   configuration.put("imageMagickOptions", Arrays.asList("-resample", "450", "-density", "350", "-quality",
   "100", "-background" , "white" , "-alpha" , "flatten"));
   configuration.put("hocrFixWords", Map.of("S(?=[0-9]+)", "\\$"));
   ```

   document_bucket - имя корзины S3, которое OCR будет использовать для сохранения результатов.
   tesseractOptions – параметры командной строки Tesseract. «-l» и «eng» означают, что язык = eng и т. д. См. внешнюю документацию по использованию командной строки Tesseract.
   imageMagickOptions - OcrTask использует инструмент ImageMagick для разделения PDF на страницы и печати в виде изображений. Вы можете указать параметры командной строки. Пожалуйста, следуйте документации по командной строке ImageMagick.

   hocrFixWords – HocrPostProcessor использует эту конфигурацию для исправления ошибок OCR. Постобработчик принимает регулярное выражение и заменяет все совпадения по значению. Обратите внимание, что знак доллара и обратная косая черта (\ ) должны быть экранированы в строке значения.

3. Чтобы инициализировать OcrTaskData , вам необходимо передать путь S3 к конкретному документу, добавить форматы, которые будут использоваться для получения результатов OCR, и предоставить конфигурацию OCR:

   ```
   this.ocrTaskData = new OcrTaskData();
   ocrTaskData.setDocumentLocation("ie_demo_invoice/invoice101.pdf");
   ocrTaskData.getFormats().addAll(Arrays.asList(OcrFormats.TEXT, OcrFormats.HOCR, OcrFormats.JSON,
   OcrFormats.IMAGE));
   ocrTaskData.setConfiguration(configuration);
   ```

   document location -путь к документу в S3 Storage.
   OcrFormats — список форматов OCR, которые в настоящее время поддерживаются задачей OCR.

configuration - карта конфигурации, которую вы должны установить из 2-го пункта.

4. Вызов **OcrTask**:

```
TaskOutput ocrInput = execute(getInput(), PrepareOcrTask.class).get();
TaskOutput ocrOutput = execute(ocrInput, OcrTask.class).get();
```

5. Чтобы получить OCR, используйте следующий пример:

```
@ApTaskEntry(name = "Store OCR Result")
@Slf4j
@InputToOutput
public class StoreOcrResult extends ApTask {

        @Input(OcrTask.OCR_TASK_DATA_KEY)
        private OcrTaskData ocrTaskData;

        @Override
        public void execute() {
                log.info("Received response OCR Task {} ", ocrTaskData.getTaskUuid());
                List<OcrResult> ocrResults = this.ocrTaskData.getOcrResults();
                //...
        }

}
```

OcrTask.OCR_TASK_DATA_KEY - ключ ввода/вывода данных. OcrTask использует этот ключ для получения входных данных для OCR.

Полный пример

PrepareOcrTask.java

```
import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrFormats;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.sample.ml.ie.repository.DocumentRepository;
import lombok.extern.slf4j.Slf4j;

import javax.inject.Inject;
import java.util.Arrays;
import java.util.Map;

@ApTaskEntry(name = "Prepare OCR Task")
@Slf4j
@InputToOutput
public class PrepareOcrTask extends ApTask {

    @Inject
    private DocumentRepository documentRepository;

    @Output(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskData;

    @Output("configuration")
    private Map<String, Object> configuration;

    @AfterInit
    public void init() {
```

```
        configuration = MlTaskUtils.readObjectFromString(getConfigurationService().get("CONFIGURATION", "{}"),
Map.class);
    }

    @Override
    public void execute() {
        this.ocrTaskData = new OcrTaskData();
        ocrTaskData.setDocumentLocation("ie_demo_invoice/invoice101.pdf");
        ocrTaskData.getFormats().addAll(Arrays.asList(OcrFormats.TEXT, OcrFormats.HOCR, OcrFormats.JSON,
OcrFormats.IMAGE));
        ocrTaskData.setConfiguration(configuration);
    }
}
```

StoreOcrResult.java

```
package com.iba.sample.ml.ie.tasks;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrResult;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import lombok.extern.slf4j.Slf4j;

import java.util.List;

@ApTaskEntry(name = "Store OCR Result")
@Slf4j
@InputToOutput
public class StoreOcrResult extends ApTask {

    @Input(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskData;

    @Override
    public void execute() {
        log.info("Received response OCR Task {} ", ocrTaskData.getTaskUuid());
        List<OcrResult> ocrResults = this.ocrTaskData.getOcrResults();
        //...
    }

}
```

OcrProcess.java

```
import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.boot.ApModuleRunner;
import com.iba.kanclerrpa.engine.boot.configuration.DevelopmentConfigurationModule;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.sample.ml.ie.tasks.StoreOcrResult;
import com.iba.sample.ml.ie.tasks.PrepareOcrTask;
import lombok.extern.slf4j.Slf4j;

@ApModuleEntry(name = "OCR AP Sample", description = "OCR AP Sample")
@Slf4j
public class OcrProcess extends ApModule {

    public TaskOutput run() throws Exception {
```

```
                return execute(getInput(), PrepareOcrTask.class)
                        .thenCompose(execute(OcrTask.class))
                        .thenCompose(execute(StoreOcrResult.class)).get();
    }

    public static void main(String[] args) {
        ApModuleRunner.localLaunch(OcrProcess.class, new DevelopmentConfigurationModule("cs.json"));
    }

}
```

Модуль интеграции ABBYY FlexiCapture

Что такое FlexiCapture?

ABBYY FlexiCapture — это платформа интеллектуальной обработки документов, созданная для сложной обработки цифровых документов. FlexiCapture объединяет лучшие возможности обработки естественного языка, машинного обучения и расширенных возможностей оптического распознавания символов в единую платформу корпоративного масштаба для обработки документов любого типа, от простых форм до сложных документов произвольной формы. Эта библиотека доступна в nexus Канцлер RPA:

```
<dependency>
    <groupId>com.iba</groupId>
    <artifactId>kanclerrpa-flexicapture-client</artifactId>
    <version>${version}</version>
</dependency>
```

Предварительные условия

- Установленный и настроенный сервер FlexiCapture v.12 или выше.
- Действующая лицензию, включая функции Verification и Scanning.
- Проект FlexiLayout должен быть создан и развернут на сервере. Материалы для пользователей и разработчиков FlexiCapture можно найти на официальном учебном ресурсе ABBYY FlexiCapture https://help.abbyy.com/en-us/flexicapture/12/flexilayout_studio/tutorial_all.

Использование модуля

Maven

Чтобы включить модуль, используйте maven зависимость :

```
        <dependency>
                <groupId>com.iba</groupId>
                <artifactId>kancler-rpa-flexicapture-client</artifactId>
                <version>1.0</version>
        </dependency>
```

Этапы разработки

Ниже представлен список и последовательность методов, которые необходимы для реализации сценария извлечения данных:

Инициализация сервиса

```
FlexiCaptureService client = new FlexiCaptureService(String flexiCaptureBaseUrl, String username, String
password);
```

Создание задачи распознавания

```
SessionContainer createRecognitionTask(String fileName, InputStream stream, String projectName);
```

Получение процента готовности задачи

```
int percentCompleted = getTaskStatus(@NotNull SessionContainer recognitionTaskData)
```

Получение списка документов, требующих проверки

```
List<String> getVerificationQueue(SessionContainer recognitionTaskData)
```

Ссылка для проверки задачи сборки

```
{FlexiCaptureServerUrl}/FlexiCapture12/Verification/Verify?taskId={taskId}
```

Получение результатов распознавания

```
ExtractResultsContainer getBatchExtractResult(SessionContainer recognitionTaskData)
```

Примеры использования

Синхронный вызов

```
    public void runAttendedScenario() throws  Exception{
        String fileNameWithExtension = "ES_Manzana.tif";
        String invoiceDemoProject = "MultipageInvoiceProject";
                String username = "Administrator";
                String password = "password";

        FlexiCaptureService client = new FlexiCaptureService("http://10.224.32.19", username, password);
        InputStream fileStream = this.getClass().getResourceAsStream(fileNameWithExtension);

        String fileName = FilenameUtils.getName(fileNameWithExtension);

        SessionContainer task = client.createRecognitionTask(fileName, fileStream, invoiceDemoProject);
        int percentCompleted = 0;
        while (percentCompleted < 100){
            percentCompleted = client.getTaskStatus(task);
            logger.info("processed " + percentCompleted + "%");
            if(percentCompleted < 100){
                if(client.getVerificationQueue(task).size() > 0){
                    logger.warn("Manual verification required: " + client.getVerificationUrl() + task.
getTaskId());
                }
                Thread.sleep(5000);
```

```
            }
        }
        ExtractResultsContainer extractResults = client.getBatchExtractResult(task);
        List<File> savedFiles = CommonUtils.storeToDir(extractResults);
        logger.info("Recognition results:");
        for(File savedFile : savedFiles){
            logger.info(savedFile);
        }
    }
```

Асинхронный вызов

```
public class FlexiCaptureEventListenerImpl implements FlexiCaptureEventListner {
    private static final Logger logger = Logger.getLogger(FlexiCaptureEventListenerImpl.class);
    @Override
    public void onSuccess(ExtractResultsContainer extractResults, SessionContainer task) throws Exception {
        List<File> savedFiles = CommonUtils.storeToDir(extractResults);
        savedFiles.forEach(file -> logger.info(file));
    }

    @Override
    public void onError(Exception e) {
        logger.error("Document recognition error: ", e);
    }

    @Override
    public void onValidationRequired(String verificationUrl, SessionContainer task) {
        System.out.println("Manual verification required: " + verificationUrl + task.getTaskId());
    }
}

public class FlexiCaptureAsyncRunner {
    public static void main(String[] args) throws UserNotFoundException, ServerFault, MalformedURLException {
        String fileNameWithExtension = "/ES_Manzana.tif";
        String invoiceDemoProject = "InvoiceDemoProject";
        String username = "Administrator";
        String password = "Start123";

        InputStream fileStream = FlexiCaptureAsyncRunner.class.getResourceAsStream(fileNameWithExtension);
        FlexiCaptureEventListner listener = new FlexiCaptureEventListenerImpl();
        FlexiCaptureService client = new FlexiCaptureService("http://10.224.32.19", username, password);
        String fileName = FilenameUtils.getName(fileNameWithExtension);

        client.createAsyncRecognitionTask(fileName, fileStream, invoiceDemoProject, listener);

    }
}
```

Рекомендации по OCR

OCR (оптическое распознавание символов) — это технология, используемая для распознавания текста внутри изображений, таких как отсканированные документы или фотографии, и преобразования его в машиночитаемые текстовые данные.

Встроенный процесс OCR платформы Канцлер RPA включает 3 основных этапа:

- Препроцессор ImageMagick
- Механизм распознавания текста Tesseract
- HOCR Постпроцессор

Препроцессор ImageMagick

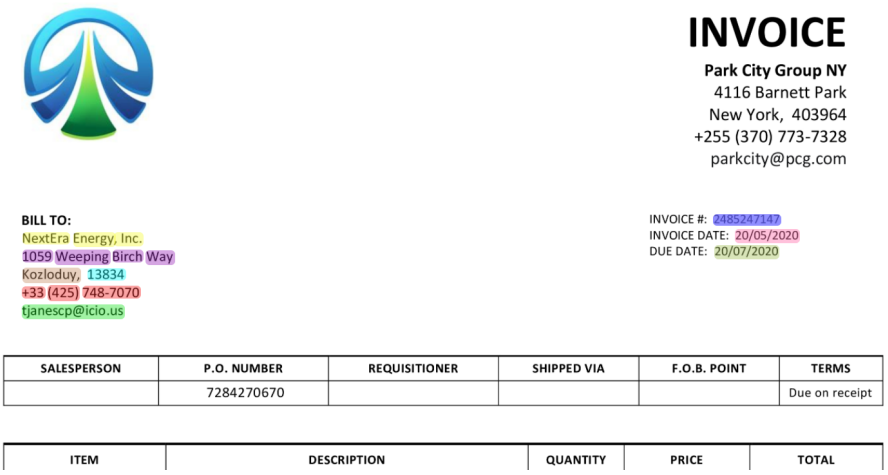
Препроцессор ImageMagick используется для отображения, создания, преобразования, изменения и редактирования цифровых изображений. ImageMagick в основном состоит из ряда опций для управления изображениями.

Список рекомендуемых к использованию опций ImageMagick:

- **-resample**. Этот параметр изменяет размер изображения таким образом, чтобы его визуализированный размер оставался таким же, как у оригинала при указанном целевом разрешении. Рекомендуемое значение: не менее 300 DPI. Дополнительные сведения о DPI см. в документации Tesseract — Масштабирование.
- **-density**. Этот параметр указывает разрешение изображения, которое необходимо сохранить при кодировании растрового изображения. Единицей измерения по умолчанию является количество точек на дюйм (DPI). Рекомендуемое значение должно быть равно параметру *-resample*.
- **-units**. Этот параметр определяет единицы разрешения изображения и обычно используется вместе с параметром *-density* . Рекомендуемое значение: "*PixelsPerInch*."
- **-quality**. Этот параметр определяет уровень сжатия JPEG/MIFF/PNG. Рекомендуемое значение: «100»..
- **-deskew**. Этот параметр выпрямляет изображение. Порог 40% подходит для большинства изображений. Рекомендуемое значение: «40%».
- **-contrast**. Используйте этот параметр для повышения контрастности изображения.
- **+contrast**. Используйте этот параметр для понижения контрастности изображения.
- **-alpha.** Этот параметр помогает исправить слои в читаемых PDF-файлах. Рекомендуемое значение: "*flatten"*.

Ниже вы можете увидеть результат распознавания читабельного PDF с и без "-alpha", "flatten" соответственно:

1)



INVOICE

Park City Group NY
4116 Barnett Park
New York, 403964
+255 (370) 773-7328
parkcity@pcg.com

BILL TO:
NextEra Energy, Inc.
1059 Weeping Birch Way
Kozloduy, 13834
+33 (425) 748-7070
tjanescp@icio.us

INVOICE #: 7485247147
INVOICE DATE: 20/05/2020
DUE DATE: 20/07/2020

SALESPERSON	P.O. NUMBER	REQUISITIONER	SHIPPED VIA	F.O.B. POINT	TERMS
	7284270670				Due on receipt

ITEM	DESCRIPTION	QUANTITY	PRICE	TOTAL

2) 

Расширенный список параметров ImageMagick см. в разделе Полный список параметров ImageMagick.

Механизм распознавания текста Tesseract

Tesseract — это OCR-движок с поддержкой юникода и способностью распознавать более 100 языков.

Список рекомендуемых к использованию опций Tesseract:

- **-l**. Этот параметр указывает используемый язык или сценарий. Если ничего не указано, предполагается 'eng' (английский). С помощью + можно указать несколько языков или скриптов. Пример: «eng+deu+fra». Для получения дополнительной информации вы можете обратиться к полному списку языков Tesseract.
- **--psm**. Метод сегментации страницы позволяет Tesseract запускать анализ макета и принимать определенную форму изображения. Рекомендуемое значение: 12.
- **--oem**. Этот параметр указывает режим модуля OCR. Рекомендуемое значение: 3 (по умолчанию).
- **--dpi**. Этот параметр указывает разрешение *N* в DPI для входного изображения. Рекомендуемое значение должно быть равно параметру *-resample* ImageMagick и иметь разрешение не менее 300 DPI.

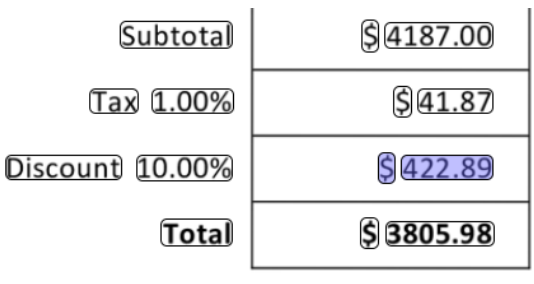
⚠️ Без этой опции разрешение считывается из метаданных, включенных в изображение. Если изображение не содержит этой информации, Tesseract пытается ее угадать, что приводит к значительному увеличению времени обработки документа.

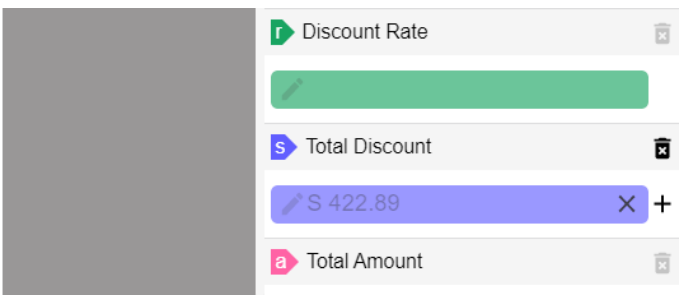
Для получения дополнительной информации о параметрах Tesseract посетите страницу руководства Tesseract.

HOCR Постпроцессор

Постпроцессор HOCR помогает исправить распространенные ошибки OCR, предоставляя пары распознанных и исправленных слов. Правила постобработки можно задавать как парами слов, так и регулярным выражением.

В приведенном примере видно, что символ «$» был распознан как «S».





Исправить ошибку OCR можно с помощью:

- пары слов: "^S$": "\\$" - в случае если '$' является отдельным bbox.

- регулярного выражения : "S(?=[0-9]+([.,][0-9]+)?)": "\\$"- в случае если '$' представляет часть bbox и следует сумма, т. е. '$422.89'.

Машинное обучение

Понятия и сущности

Машинное обучение - это метод анализа данных, который автоматизирует построение аналитической модели. Алгоритмы машинного обучения строят модель на основе выборочных данных, известных **как тренировочные данные**, чтобы делать прогнозы или принимать решения без необходимости программирования. Алгоритмы машинного обучения используются в самых разных приложениях, таких как фильтрация электронной почты и машинное зрение, где трудно или невозможно разработать обычные алгоритмы для выполнения необходимых задач.

Платформа Канцлер RPA предоставляет разработчикам инфраструктуру для тренировки моделей машинного обучения и использования их для обработки данных.

Давайте посмотрим на артефакты Канцлер RPA, связанные с областью машинного обучения.

Пакеты документов

Работа над моделями машинного обучения начинается с данных - желательно, большого количества данных (документов), для которых целевой ответ известен. Когда целевой ответ определён, документы называются размеченными. Например, для задачи классификации электронной почты целью является метка, указывающая, является ли электронное письмо спамом или нет. Алгоритм машинного обучения обучается на размеченных примерах, которые мы предоставляем.

Зачастую данные недоступны (или нечитабельны) в размеченной форме. Сбор и маркировка данных часто является наиболее важным шагом в решении проблемы машинного обучения. Пример данных должен быть репрезентативным для данных, которые будут обрабатываться моделью при прогнозировании. Например, если вы хотите предсказать, является ли электронное письмо спамом или нет, вы должны собирать как положительные (спам-письма), так и отрицательные (неспам-письма), чтобы алгоритм машинного обучения мог находить шаблоны, которые будут различать эти два типа электронной почты.

После того, как у вас есть размеченные данные, вам может потребоваться преобразовать их в формат, приемлемый для вашего алгоритма или программного обеспечения - файл, называемый тренировочным набором. Следующим шагом является тренировка модели. Этот процесс содержит алгоритм машинного обучения с тренировочными данными и всеми необходимыми параметрами и конфигурациями, в результате создается модель машинного обучения и тренировочный лог сохраняется в системе.

Обработчик документов

Обработчик докумнтов - это специальный автоматизированный процесс, который контролирует жизненный цикл пакетов документов и обрабатывает все преобразования, необходимые для сбора данных и применения OCR, разметки документов в **Рабо чем пространстве**, подготовки тренировочных пакетов данных и тренировки моделей.

Встроенные обработчики документов

Имя автоматизированного процесса	Репозиторий Nexus	Идентификатор группы	Идентификатор предмета	Файл	Описание
CL Document Processor	kanclerrpa	com.iba	kancler-rpa-cldp-ap	kancler-rpa-cldp-ap-X.X.X-bin.zip	Осуществляет OCR-обработку изображений/pdf-файлов в формате xocr, разметку для извлечения информации
IE Document Processor	kanclerrpa	com.iba	kancler-rpa-iedp-		Осуществляет OCR-обработку

			ap	kancler-rpa-cldp-ap-X.X.X-bin.zip	изображений/pdf-файлов в формате xocr, классификацию
HTML CL Document Processor	kanclerrpa	com.iba	kancler-rpa-clhtmldp-ap	kancler-rpa-clhtmldp-ap-X.X.X-bin.zip	Осуществляет разметку для извлечения информации из HTML-документов
HTML IE Document Processor	kanclerrpa	com.iba	kancler-rpa-iehtmldp-ap	kancler-rpa-iehtmldp-ap-X.X.X-bin.zip	Осуществляет классификацию документов в HTML-формате

Тип пользовательской задачи

Тип пользовательской задачи - это специальное приложение java-скрипт (предоставляется в виде zip-файла), которое управляет жизненным циклом обработки документов человеком. Оно обрабатывает все действия пользовательского интерфейса над документом для подготовки outputJson из inputJson в **Рабочем пространстве**.

Встроенные типы пользовательских задач

Имя типа пользовательской задачи	Репозиторий Nexus	Идентификатор группы	Идентификатор предмета	Файл	Описание
Classification Task	kanclerrpa	com.iba	kancler-rpa-cldp-ap	kancler-rpa-cldp-ap-X.X.X-bin.zip	Осуществляет разметку для извлечения информации в формате xocr
Information Extraction Task	kanclerrpa	com.iba	kancler-rpa-iedp-ap	kancler-rpa-cldp-ap-X.X.X-bin.zip	Осуществляет классификацию в формате xocr
HTML Classification Task	kanclerrpa	com.iba	kancler-rpa-clhtmldp-ap	kancler-rpa-clhtmldp-ap-X.X.X-bin.zip	Осуществляет разметку для извлечения информации из HTML-документов
HTML Information Extraction Task	kanclerrpa	com.iba	kancler-rpa-iehtmldp-ap	kancler-rpa-iehtmldp-ap-X.X.X-bin.zip	Осуществляет классификацию документов в HTML-формате

Модели машинного обучения

Термин **модель машинного обучения** относится к артефакту, который создается процессом тренировки. Модель машинного обучения Канцлер RPA - это архивный файл, который включает в себя все сценарии, сериализованные классы, параметры и файлы конфигурации, необходимые для запуска обработки данных.

Платформа Канцлер RPA поддерживает два типа моделей машинного обучения:

- Модели классификации
- Модели извлечения информации

Встроенные модели машинного обучения

Все модели размещены в репозитории **pypi-ci** nexus.

Извлечение информации

Тренируемая модель	Версия	Дата выхода	Описание улучшений
ml_ie_spacy2_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка Spacy2.3.4
ml_ie_spacy3_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка

			Spacy3.1.1

Извлечение информации из HTML

Тренируемая модель	Версия	Дата выхода	Описание улучшений
ml_iehtml_spacy2_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка Spacy2.3.4
ml_iehtml_spacy3_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка Spacy3.1.1

Классификация

Тренируемая модель	Версия	Дата выхода	Описание улучшений
ml_cl_spacy3_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка Добавлена лемматизация с использованием предварительно обученных spacy-моделей Spacy3.1.1

Классификация HTML-документов

Тренируемая модель	Версия	Дата выхода	Описание улучшений
ml_clhtml_spacy3_model	2.4.1	2022-05	Поддержка универсальных контейнеров машинного обучения Динамическая поддержка загрузки языка Добавлена лемматизация с использованием предварительно обученных spacy-моделей Spacy3.1.1

Финансовая информационная модель

Тренируемая модель	Версия	Дата выхода	Описание улучшений
ml_ie_finext_model	2.4.1	2022-05	Поддержка универсальных контейнеров Машинного обучения

Репозиторий моделей

Репозиторий моделей содержит все модели машинного обучения, доступные в системе: всякий раз, когда новая модель обучается, она помещается в репозиторий, и всякий раз, когда модель требуется для обработки данных, она извлекается из репозитория. Каждая модель идентифицируется по **названию** и **версии**.

Сервер управления Канцлер RPA также предоставляет способ экспорта существующих моделей в виде файла и импорта предварительно обученных моделей в систему.

Контейнер машинного обучения

Контейнер машинного обучения является компонентом платформы Канцлер RPA, который отвечает за две основные задачи:

- При выполнении запроса **на тренировку** контейнер модели предоставляется с тренировочным набором, файлами конфигурации модели, в которых указывается тип модели и параметры тренировки. Контейнер машинного обучения выбирает алгоритм тренировки, подходящий для заданного типа модели, выполняет процесс тренировки модели для заданного количества итераций, выбирает лучшую модель и упаковывает ее вместе с необходимыми конфигурационными файлами. На последнем шаге контейнер машинного обучения загружает полученный пакет модели в репозиторий моделей.

- Если модель используется для обработки данных, запрос указывает модель, которая должна быть выполнена, и входной JSON. Контейнер машинного обучения извлекает указанную модель из репозитория моделей, если это необходимо, и выполняет ее. Результат выполнения помещается в выходной JSON.

Сервер управления Канцлер RPA взаимодействует с контейнером машинного обучения посредством потока сообщений.

Контейнер машинного обучения является одним из масштабируемых компонентов платформы: при необходимости существует возможность масштабирования количества контейнеров машинного обучения.

Модели классификации

Обзор

Классификация документов или категоризация документов является проблемой в библиотечном деле, информатике и компьютерных науках. Задача состоит в том, чтобы назначить документу один или несколько классов или категорий. Это может быть сделано «вручную» или алгоритмически.

Канцлер RPA включает в себя реализацию как **классификации с одной меткой**, так и **класификации с несколькими метками**.

Рассмотрим разницу этих двух методов на очень простом примере и предположим, что у нас есть 3 категории документов. Независимо от того, какой тип классификации используется, ответ модели для одного документа всегда представляет собой вектор из 3 значений:

- Классификация с одной меткой - каждой категории присваивается значение от 0 до 1; сумма всех вероятностей равна 1
- Классификация с несколькими метками - каждой категории присваивается значение от 0 до 1 независимо от остальных, поэтому сумма всех вероятностей находится в диапазоне 0..3.

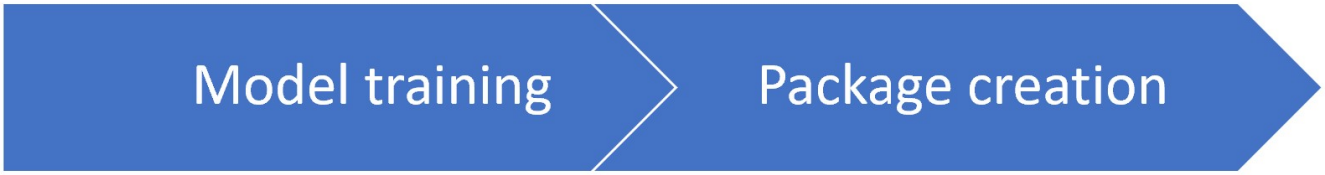
Например, мы классифицируем статьи по трем категориям: спорт, бизнес, пресса. Статья в спортивном журнале потенциально может иметь следующий балл:

- Классификация с одной меткой - [спорт = 0.3, бизнес = 0.0, пресса = 0.7]
- Классификация с несколькими метками - [спорт = 0.8, бизнес = 0.0, пресса = 0.9]

Классификация документов как поток задач

Чтобы изучить оба процесса более подробно, рассмотрим, какие этапы являются частью обучения и работы модели.

Процесс тренировки модели



Тренировка модели

В Канцлер RPA данный шаг включает в себя тренировку модели машинного обучения с использованием предоставленного тренировочного набора данных. Система запускает тренировку для заданного количества итераций и выбирает лучшую модель.

Разработчик процесса может указать тип модели (с одной или несколькими метками), количество итераций и т.д., используя конфигурационный JSON-файл.

Создание пакета

Модель Spacy поставляется в комплекте с конфигурационными файлами и загружается в репозиторий Nexus.

Процесс классификации

Запуск модели

Результатом запуска модели является JSON с двумя секциями:

- раздел **scores** содержит пары "ключ-значение" с названиями категорий и их вероятностями
- раздел **categories** содержит одну категорию, в случае классификации с одной меткой список категорий, вероятность которых выше, чем установлена в настройках типа документов "*scoreTreshold*".

```
{
  "scores": {
    "": 0.7,
    "": 0.3
  },
  "categories": [
    ""
  ]
}
```

```
{
  "scores": {
    "Science": 0.1,
    "Sport": 0.8,
    "Business": 0.2,
    "Media": 0.6
  },
  "categories": [
    "Sport",
    "Media"
  ]
}
```

Конфигурационный файл тренировки модели

Для обучения модели классификации необходимо предоставить JSON-файл, определяющий параметры конфигурации для процесса тренировки.

Рассмотрим подробнее данные параметры конфигурации.

Разработчик может указать параметры модели в файле конфигурации. Этот файл является **обязательным** и содержит следующие настройки:

- **ocr_fixes**(list of objects)(необязательно) - здесь определяются значения, которые должны быть заменены другими значениями. В приведенном ниже примере значение "G4LD" будет заменено на значение "64LD".
- **trainer_name**(string)(обязательно) - артефакт python, создающий пакеты моделей для обработки с определенным типом модели. В нем два модуля: модуль для обучения на размеченных данных и генерации пакета обученной модели, и модуль для загрузки обученной модели из Nexus или из кэша и запуска ее на входных данных. См. раздел Встроенные модели классификациии и Встроенные модели классификации HTML для получения более подробной информации.
- **trainer_version**(string)(обязательно) - версия тренера. Пожалуйста, обратитесь к разделу Встроенные модели классификациии и Встроенные модели классификации HTML для получения более подробной информации.
- **trainer_description**(string)(обязательно) - описание тренера.
- **single_label**(boolean)(необязательно) - тип модели классификации (с одной или несколькими метками). Значение по умолчанию 'true'.
- **lang**(string)(optional) - язык входных данных. Значение по умолчанию 'en'.
- **lemmatization**(boolean)(необязательно) - техника, которая используется для сведения слов к нормализованной форме. В лемматизации преобразование использует предварительно обученные компоненты spacy моделей (часть речи, контекст, таблица нормализации) для сопоставления различных вариантов слова с его корневым форматом. Таким образом, с помощью этого подхода мы можем уменьшить нетривиальные перегибы, такие как «есть», «был», «были», обратно к корню «быть». Лемматизация потенциально может повысить качество классификационной модели. Значение по умолчанию: 'false'.

- **iterations**(number)(необязательно) - количество итераций обучения модели на заданном тренировочном пакете. Значение по умолчанию: '30'.
- **categories**(list of strings)(необязательно) - категории, к которым может принадлежать документ.

```
{
  "ocr_fixes": {"G4LD": "64LD"},
  "model": {
    "trainer_name": "ml_cl_spacy3_model",
    "trainer_version": "2.4.1",
    "trainer_description": "Text classification",
    "train_config": {
      "single_label": true,
      "lemmatization": true
      "iterations": 5,
      "categories": [
        "",
        ""
      ]
    }
  }
}
```

Модели извлечения информации

Обзор

Извлечение информации — это автоматический поиск конкретной информации, относящейся к выбранной теме, из входных данных. Инструменты извлечения информации позволяют извлекать информацию из текстовых документов, баз данных, веб-сайтов или нескольких источников сразу.

> Канцлер RPA предоставляет инфраструктуру для создания и запуска моделей машинного обучения, которые извлекают информацию из **документов PDF и HTML.**

Извлечение информации как поток задач

Процесс извлечения информации реализован в Канцлер RPA в виде потока задач. В этом потоке есть нечто большее, чем модели машинного обучения: платформа включает в себя также несколько вариантов расширения машинного обучения в виде правил и словарей.

Рассмотрим более подробно, какие этапы являются частью каждого обучения и работы модели.

Процесс тренировки модели

Тренировка модели

В Канцлер RPA данный шаг включает в себя обучение модели машинного обучения с использованием предоставленного тренировочного набора. Система запускает тренировку для заданного количества итераций и выбирает лучшую модель.

Разработчик процесса может указать тип модели, количество итераций и т.д., используя конфигурационный JSON-файл.

Создание пакета

Модель Spacy поставляется в комплекте с конфигурационными файлами и загружается в репозиторий Nexus.

Процесс извлечения информации

Запуск модели

Модель запускается один раз для каждого документа.

Выходные данные модели извлечения информации для PDF-документов:

Список сущностей является результатом выполнения модели. Сущность состоит из имени метки, индекса подсчета, содержимого метки и слов OCR, соответствующих региону сущности.


```
{
  "entities": [
    {
      "name": "DebitNoteId",
      "words": [
        {
          "bbox": [
            0.8227450980392157,
            0.1696969696969697,
            0.9168627450980392,
            0.18262626262626264
          ],
          "id": "page0_area2_paragraph2_line3_word15",
          "page": 0,
          "content": "DNT6268231"
        }
      ],
      "index": 0,
          "score": 1.0,
      "content": "DNT6268231"
    },
    {
      "name": "DebitNoteDate",
      "words": [
        {
          "bbox": [
            0.8227450980392157,
            0.19818181818181818,
            0.92,
            0.21151515151515152
          ],
          "id": "page0_area2_paragraph2_line4_word23",
          "page": 0,
          "content": "2018-06-30"
        }
      ],
      "index": 0,
          "score": 0.97,
      "content": "2018-06-30"
    }
  ]
}
```


Выходные данные модели извлечения информации для HTML-документов:

Html с тегами является результатом выполнения модели. Размеченный HTML состоит из тегов rpa-выбора с информацией о метке и заказе.


```
<html>
    <head>
    </head>
    <body>
        <div class="grid-body">
            <div class="invoice-title">
                <div class="row">
                    <div class="col-xs-12"></div>
                </div><br />
                <div class="row">
                    <div class="col-xs-12">
                        <h2>invoice<br /><span class="small">order #<rpa-selection data-content="1097"
                                    data-order="0" data-type="Order Number">1097</rpa-selection></span></h2>
                    </div>
                </div>
            </div>
            <hr />
            <div class="row">
                <div class="col-xs-6">
                    <address><strong>Billed To:</strong><br />
                        <rpa-selection data-content="Costco Wholesale" data-order="0" data-type="Name">
                            Costco Wholesale</rpa-selection><br />
                        <rpa-selection data-content="999 Lake Drive" data-order="0" data-type="Addresses">
                            999 Lake Drive</rpa-selection><br />
                        <rpa-selection data-content="Issaquah, WA 98027" data-order="0"
```

```
                        data-type="Addresses">Issaquah, WA 98027</rpa-selection><br /><abbr
                        title="Phone">P:</abbr>
                    <rpa-selection data-content="(222) 417-0141" data-order="0"
                        data-type="Phone Numbers">(222) 417-0141</rpa-selection>
                </address>
            </div>
        </div>
    </div>
</body>
</html>
```

Конфигурационный файл тренировки модели

Для обучения модели извлечения информации необходимо предоставить JSON, определяющий параметры конфигурации для процесса тренировки.

Давайте подробнее рассмотрим эти параметры конфигурации.

- **ocr_fixes**(list of objects)(необязательно) - здесь определяются значения, которые должны быть заменены другими значениями. В приведенном ниже примере значение "G4LD" будет заменено на значение "64LD".
- **trainer_name**(string)(обязательно) - артефакт python, создающий пакеты моделей для обработки с определенным типом модели. В нем два модуля: модуль для обучения на размеченных данных и генерации пакета обученной модели, и модуль для загрузки обученной модели из Nexus или из кэша и запуска ее на входных данных. См. раздел Встроенные модели извлечения информации и Встроенные модели извлечения информации из HTML для получения более подробной информации.
- **trainer_version**(string)(обязательно) - версия тренера. См. раздел Встроенные модели извлечения информации и Встроенные модели извлечения информации из HTML для получения более подробной информации.
- **trainer_description**(string)(обязательно) - описание тренера
- **lang**(string)(необязательно) - язык входных данных. Значение по умолчанию 'en'.
- **iterations**(number)(необязательно) - количество итераций обучения модели на заданном тренировочном пакете. Значение по умолчанию: '30'.
- **concat_single_entities**(boolean)(необязательно) - Значение по умолчанию: 'true'.
- **post_processing_rules**(list of objects)(необязательно) - после NER-извлечения модель использует **EntityMatcher** с правилами, определенными в post_processing_rules.json. Конфигурационный JSON должен содержать список имен меток с регулярными выражениями для поиска сущностей.
- **base_model_patterns**(list of objects)(необязательно) - используется для настройки **EntityRuler** для маркировки элементов данных. Он запускается перед извлечением данных и предоставляет модели дополнительную информацию о структуре документа, повышающую точность извлечения данных.
- **labels**(list of objects)(необязательно) - метки добавляются NER pipe на этапе обучения. В случае пустой конфигурации все метки, найденные в тренировочном наборе данных, будут автоматически добавлены в модель, а выходное измерение будет выведено автоматически (дорогостоящая операция). Флаг множественности влияет на способ вычисления индекса сущности на этапе обработки. Индекс меток с кратностью равен приращениям True по всему документу, в то время как для меток с False индекс кратности всегда равен нулю.

```
{
    "ocr_fixes": {
        "G4LD": "64LD"
    },
    "model": {
        "trainer_name": "ml_ie_spacy3_model",
        "trainer_version": "2.4.1",
        "trainer_description": "Information Extraction",
        "train_config": {
            "lang": "en",
            "iterations": 5
        },
        "process_config": {
            "concat_single_entities": true
        }
    },
    "post_processing_rules": [
        {
            "label": "kwDebitNoteID",
            "regex": [
                "Debit",
                "^Note.*$",
                "#",
                ":"
            ]
        }
    ],
    "base_model_patterns": [
        {
            "label": "kwDebitNoteID",
            "id": "kwDebitNoteID",
```

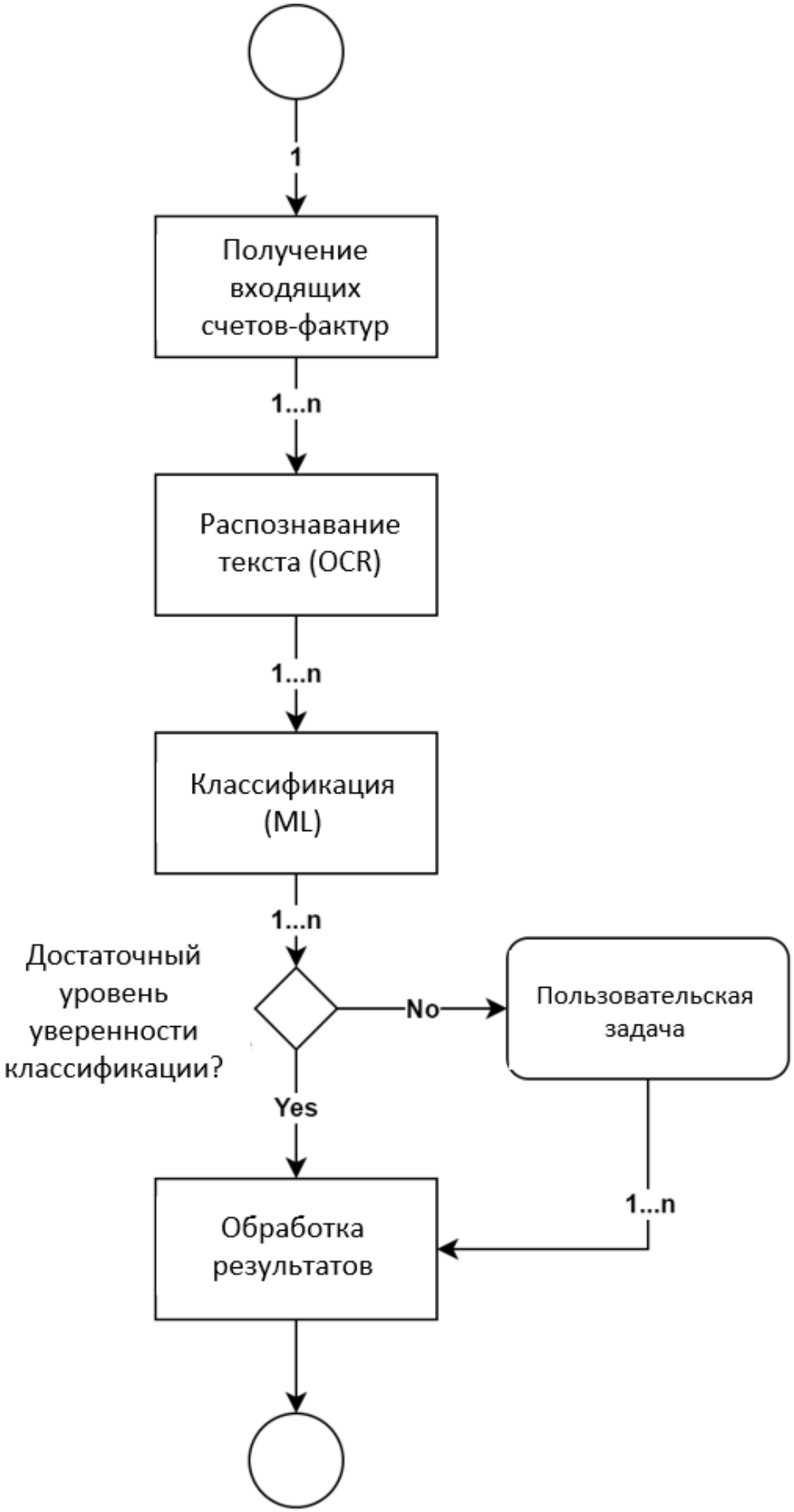
```
            "pattern": [
                {
                    "TEXT": "Debit"
                },
                {
                    "TEXT": {
                        "REGEX": "^Note.*$"
                    }
                },
                {
                    "TEXT": "#"
                },
                {
                    "TEXT": ":"
                }
            ]
        },
        {
            "label": "kwDebitNoteID",
            "id": "kwDebitNoteID",
            "pattern": [
                {
                    "TEXT": "Debit"
                },
                {
                    "TEXT": {
                        "REGEX": "^Note.*$"
                    }
                },
                {
                    "TEXT": ":"
                }
            ]
        },
        {
            "label": "kwDebitNoteDate",
            "id": "kwDebitNoteDate",
            "pattern": [
                {
                    "TEXT": "Debit"
                },
                {
                    "TEXT": "Note"
                },
                {
                    "TEXT": "Date"
                },
                {
                    "TEXT": {
                        "REGEX": "^[:|;]$"
                    }
                }
            ]
        }
    ],
    "labels": {
        "DebitNoteDate": false,
        "DebitNoteId": false,
        "VendorSupportProgram": false,
        "Percent": false,
        "TotalAmount": false,
        "SKU": true,
        "SKUDescription": true,
        "Units": true,
        "Quota": true,
        "kwDebitNoteID": false,
        "kwDebitNoteDate": false,
        "BillingFrequency": false
    }
}
```

Процесс классификации документов

Данный раздел содержит описание полного процесса разработки автоматизации классификации. Он содержит 2 статьи:

1. Настройка пользовательской задачи и тренировка модели машинного обучения - последовательность из **трёх шагов**
2. Разработка автоматизированного процесса, которая использует модель классификации - последовательность из **пяти шагов**

Автоматизированный процесс классификации представляет собой последовательнось, отображённую на схеме ниже, он должен обрабатывать все записи асинхронно параллельно:


1
Получение входящих счетов-фактур
1...n
Распознавание текста (OCR)
1...n
Классификация (ML)
1...n
Достаточный уровень уверенности классификации?
No
Пользовательская задача
Yes
1...n
Обработка результатов

Настройка пользовательской задачи и тренировка модели машинного обучения (Классификация)

Шаг 1. Создание нового типа документов (Классификация)

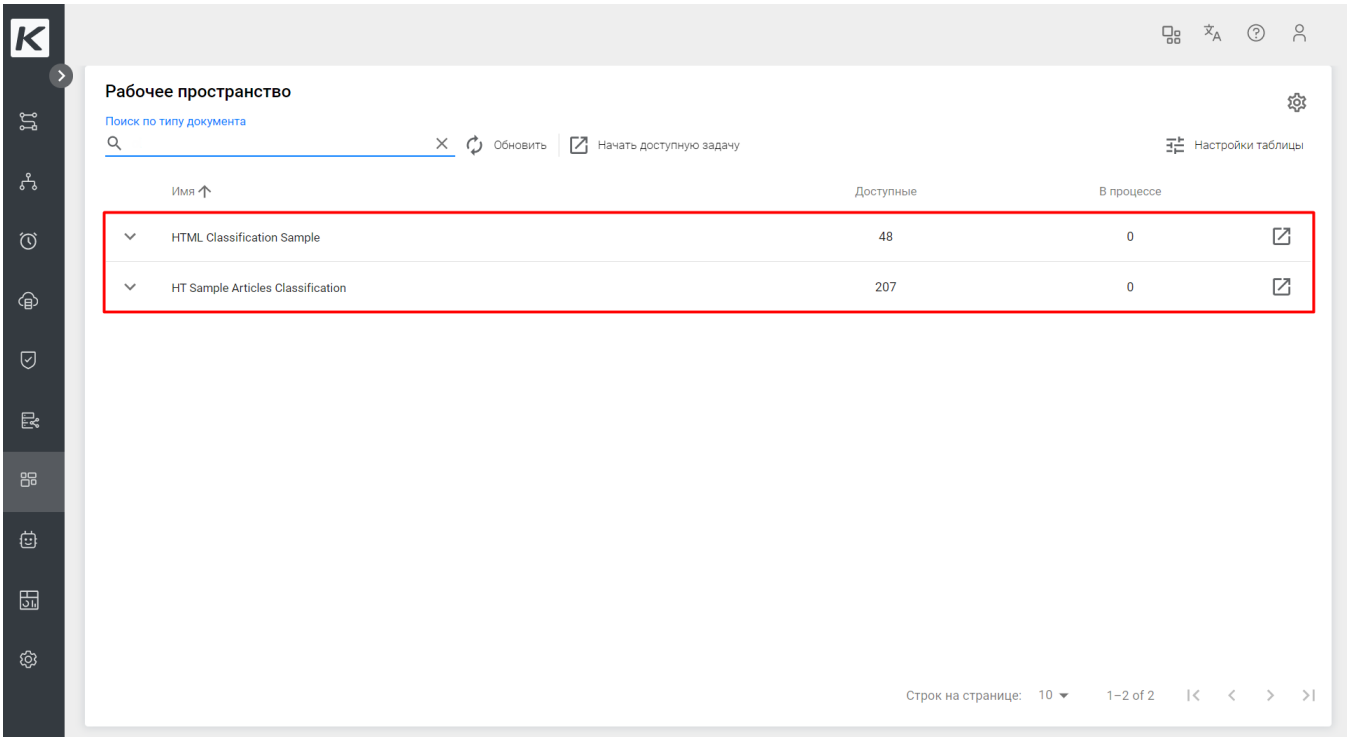
Для описания категорий, с которыми может быть связан документ, необходимо создать новый **тип документа. Тип документа** — это конфигурация, которая позволяет пользовательской задаче и модели машинного обучения знать категории ваших документов. Каждый тип документов относится к определённому **типу пользовательской задачи**.

Тип пользовательской задачи — это специальное веб-приложение, которое может считывать и анализировать конфигурацию типа документов и соответствующим образом отображать входные документы.

Канцлер RPA предоставляет 5 предопределенных типов пользовательских задач:

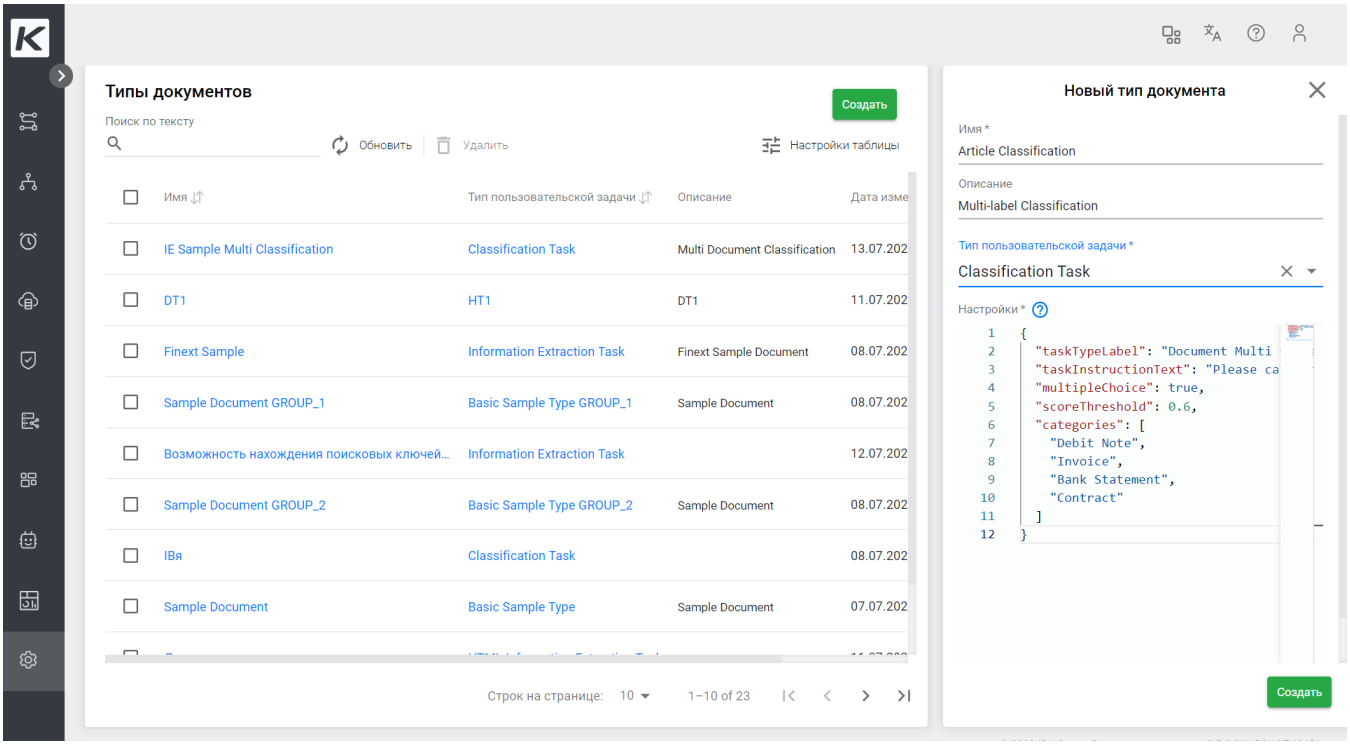
- Тип пользовательской задачи извлечения информации
- Тип пользовательской задачи классификации документов
- Тип пользовательской задачи заполнения формы
- Тип пользовательской задачи извлечения информации из HTML-документов
- Тип пользовательской задачи классификации HTML-документов

Для процесса **классификации** нам нужно создать тип документов, который относится к предопределенным типу пользовательской задачи классификации или типу пользовательской задачи классификации HTML-документов (в зависимости от типа обрабатываемых документов):



> **Примечание**
>
> Вы можете создавать свои собственные типы пользовательских задач для конкретных случаев. Обратитесь к статье Создание типа пользовательской задачи.

Для создания нового типа документов перейдите в **Администрирование - Типы документов** и нажмите кнопку **Создать**. Заполните поля **Имя, Описание, Тип пользовательской задачи** и **Настройки** для нового типа документов. См. Типы документов.

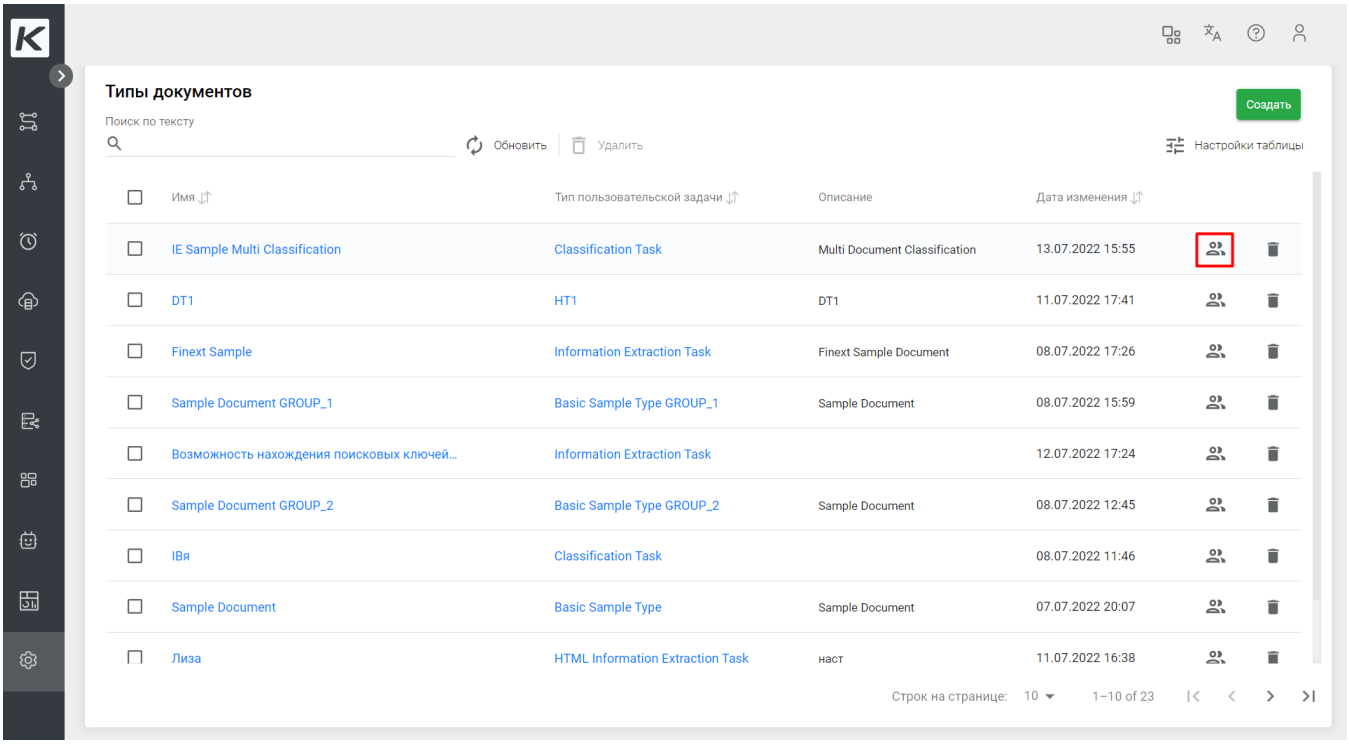


Параметры типа документов представляют собой конфигурацию в формате JSON, подробнее о настройках см. следующую статью: Настройки JSON-структуры типа документов.

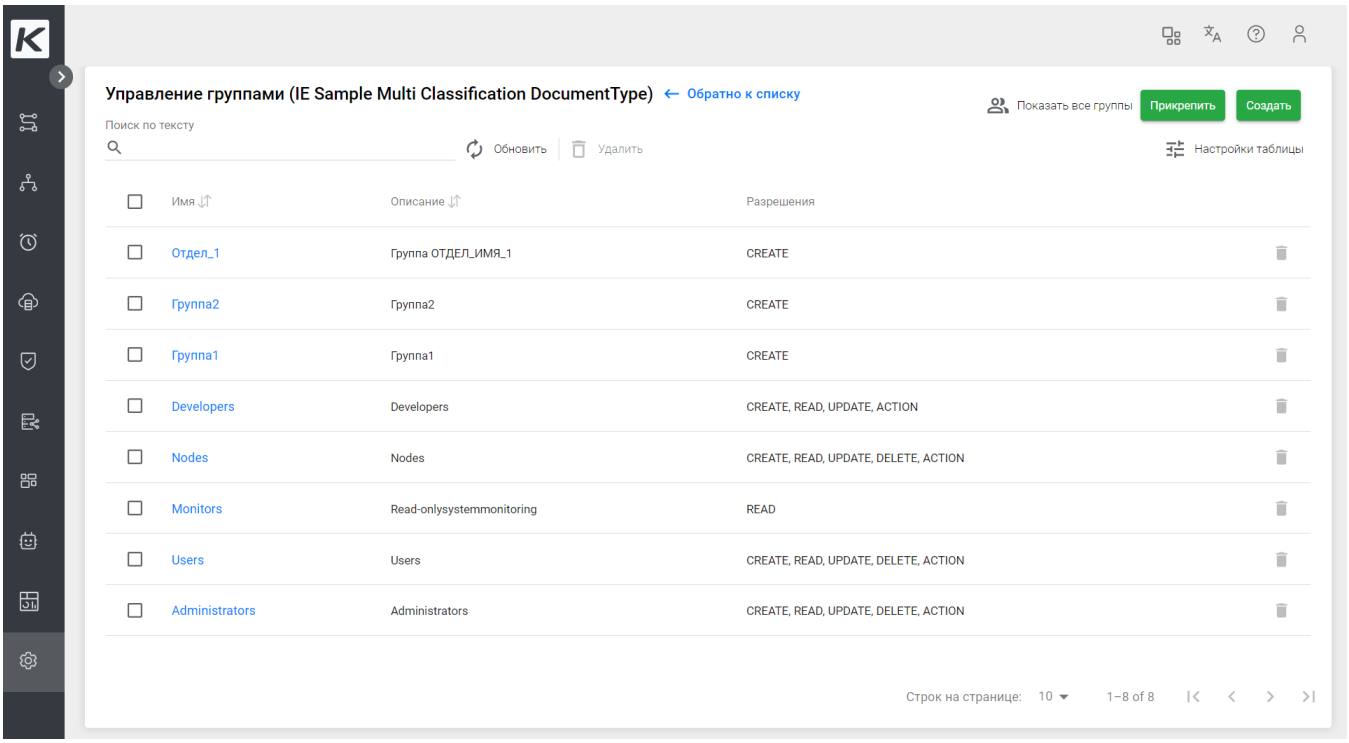
Ограничение доступа к типу документов

Каждый тип документа может быть быть видимым только для определенной группы пользователей. Это полезно, когда вам нужно позволить разным командам работать только с теми типами документов, за которые они отвечают. Например, отдел страхования должен иметь доступ к типу документов "страховые случаи", в то время как отдел маркетинга должен иметь возможность видеть только тип документов "аннулирование счетов-фактур".

Чтобы перейти к настройкам списка групп, имеющих доступ к определённому типу документов используйте кнопку Настройки доступа в строке с типом документов :

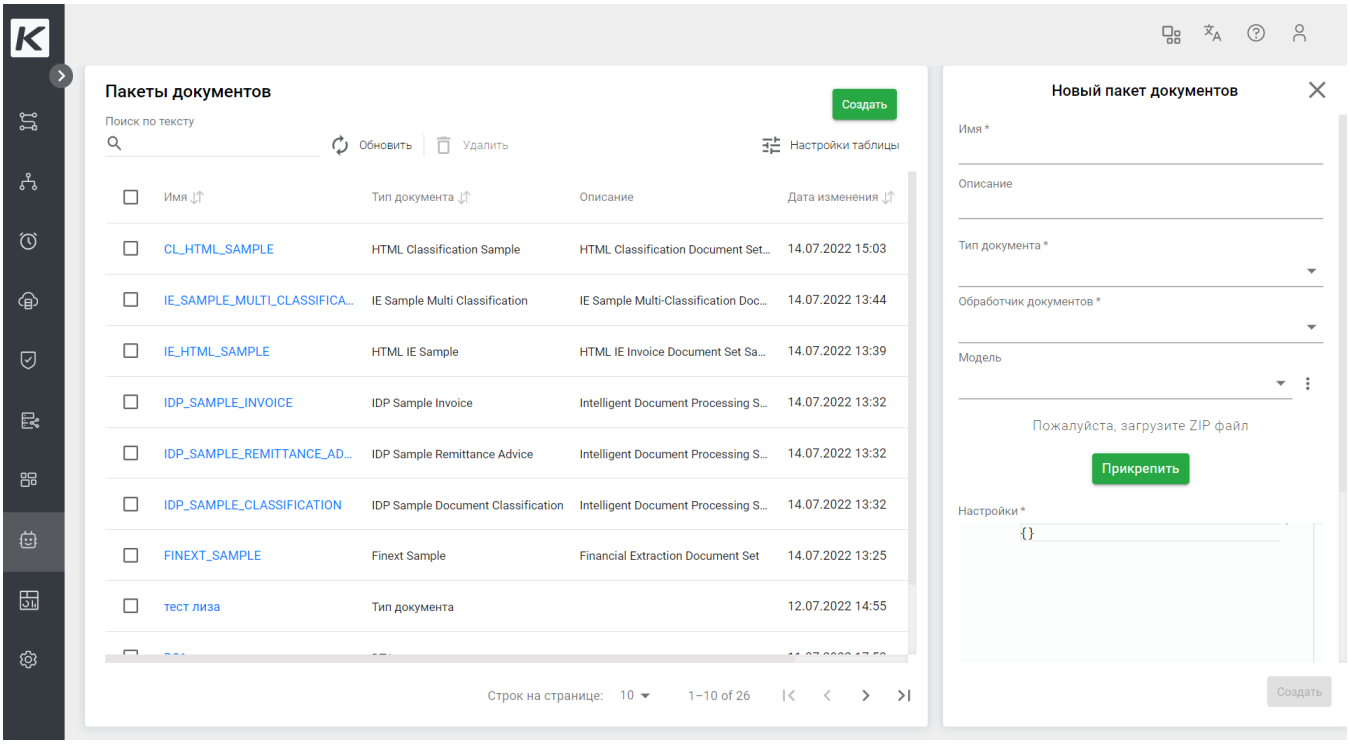


Эта кнопка перенаправляет на страницу **Управление группами** для конкретного типа документов, где можно настроить доступ:



Шаг 2. Подготовка набора данных для тренировки (Классификация)

Чтобы подготовить тренировочный набор, который используется для тренировки модели машинного обучения, необходимо создать новый пакет документов. Для этого перейдите в **Машинное обучение - Пакеты документов** и нажмите кнопку **Создать**.



Подготовка тренировочного набора состоит из нескольких шагов:

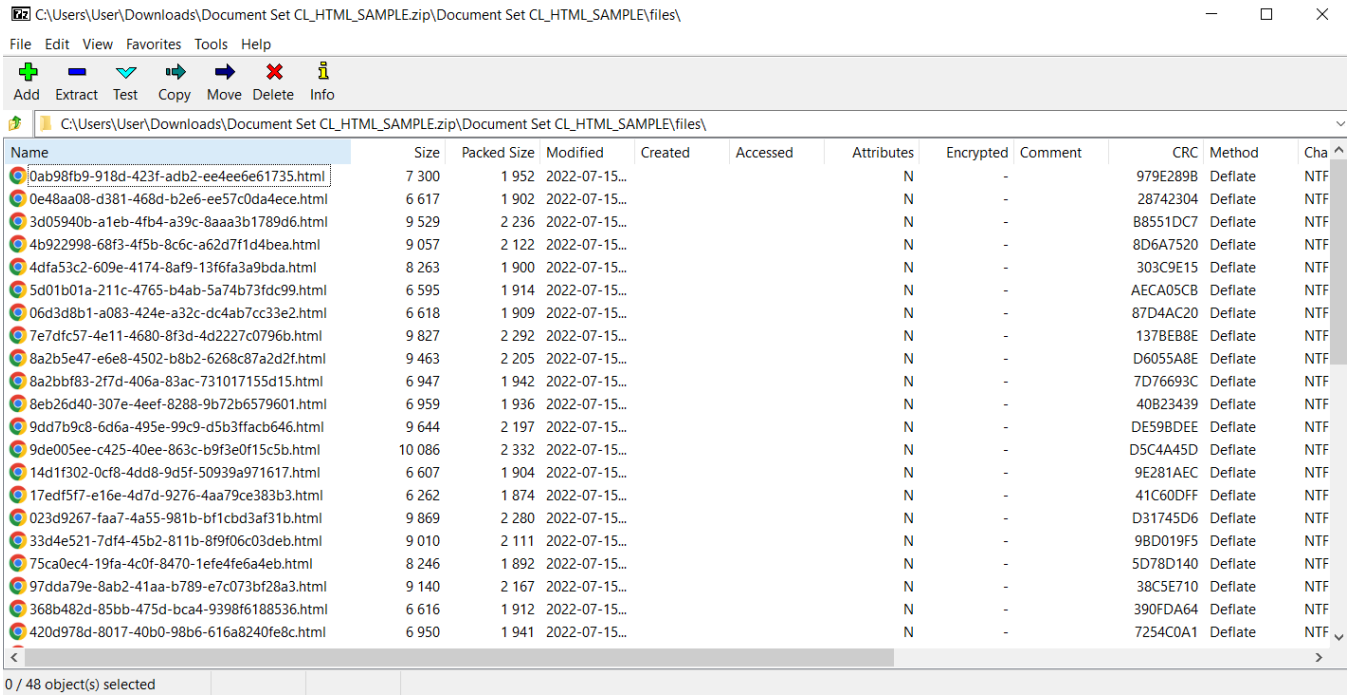
- Загрузка ZIP-файла
- Выбор типа документов
- Подготовка входных данных документов
- Отправка документов на рабочее пространство и их разметка

Загрузка ZIP-файла

Главная информация тренировочного набора это документы. Вам нужно поместить в ZIP-архив пакет документов, которые вы хотите использовать для тренировки модели машинного обучения.

> По умолчанию Канцлер RPA поддерживает обработку изображений, документов формата PDF и HTML для тренировки модели классификации.

Ваш архив должен выглядеть следующим образом:



Выбор типа документов

Выберите тип документов, созданный в предыдущем шаге. Тип документов определяет, как документы должны отображаться в пользовательской задаче для присвоения категорий и тренировки модели машинного обучения.

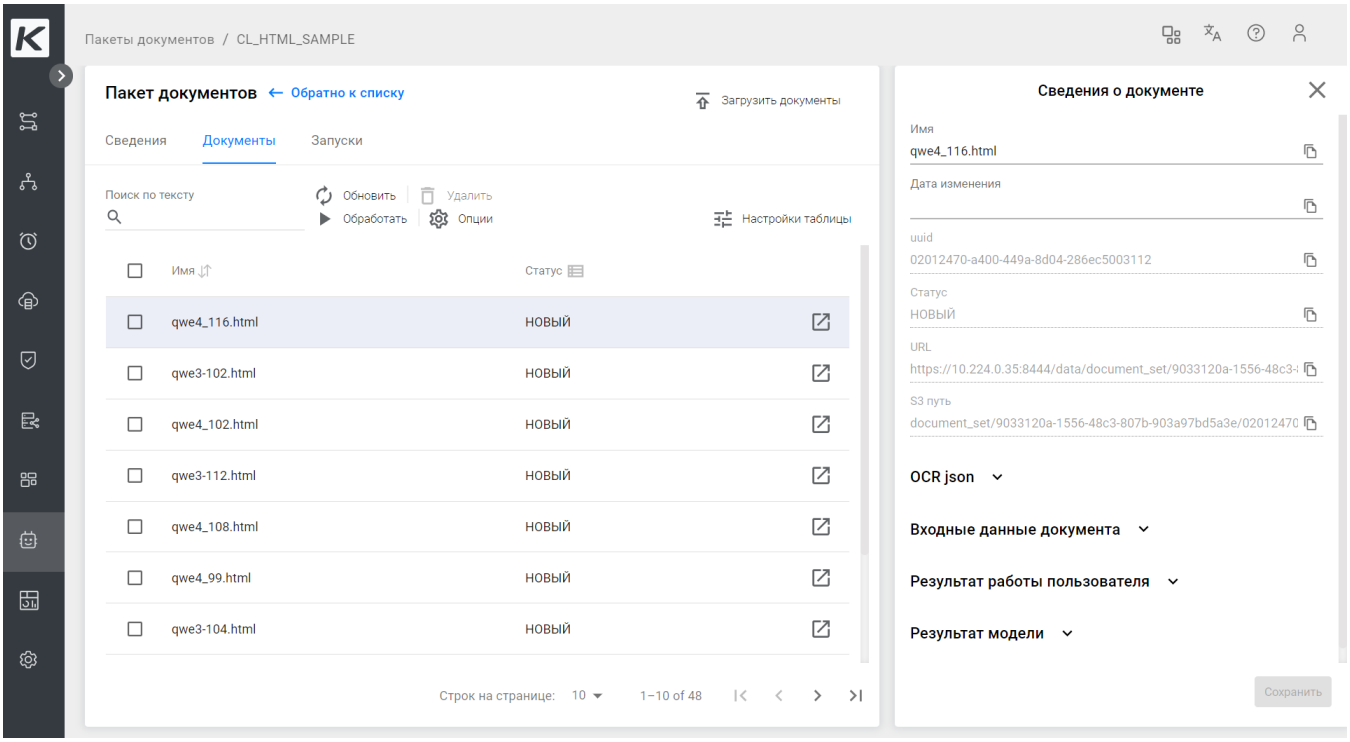
Обработчик документов

Обработчик документов - это специальный процесс, который включает в себя этапы подготовки к отображению ваших документов в пользовательской задаче.

Для обработки документов в HTML-формате должен быть выбран обработчик "**CL HTML Document Processor**". Для обработки текстовых, PDF документов или изображений Канцлер RPA предлагает "**CL Document Processor**". Этот обработчик включает в себя:

- преобразование PDF-файлов в формат изображений;
- улучшение изображений с помощью скриптов ImageMagick;
- отправку изображений на OCR;
- отправку документов с результатом OCR в пользовательские задачи;
- обработку результатов выполнения пользовательской задачи и их преобразование в формат входных данных модели машинного обучения.

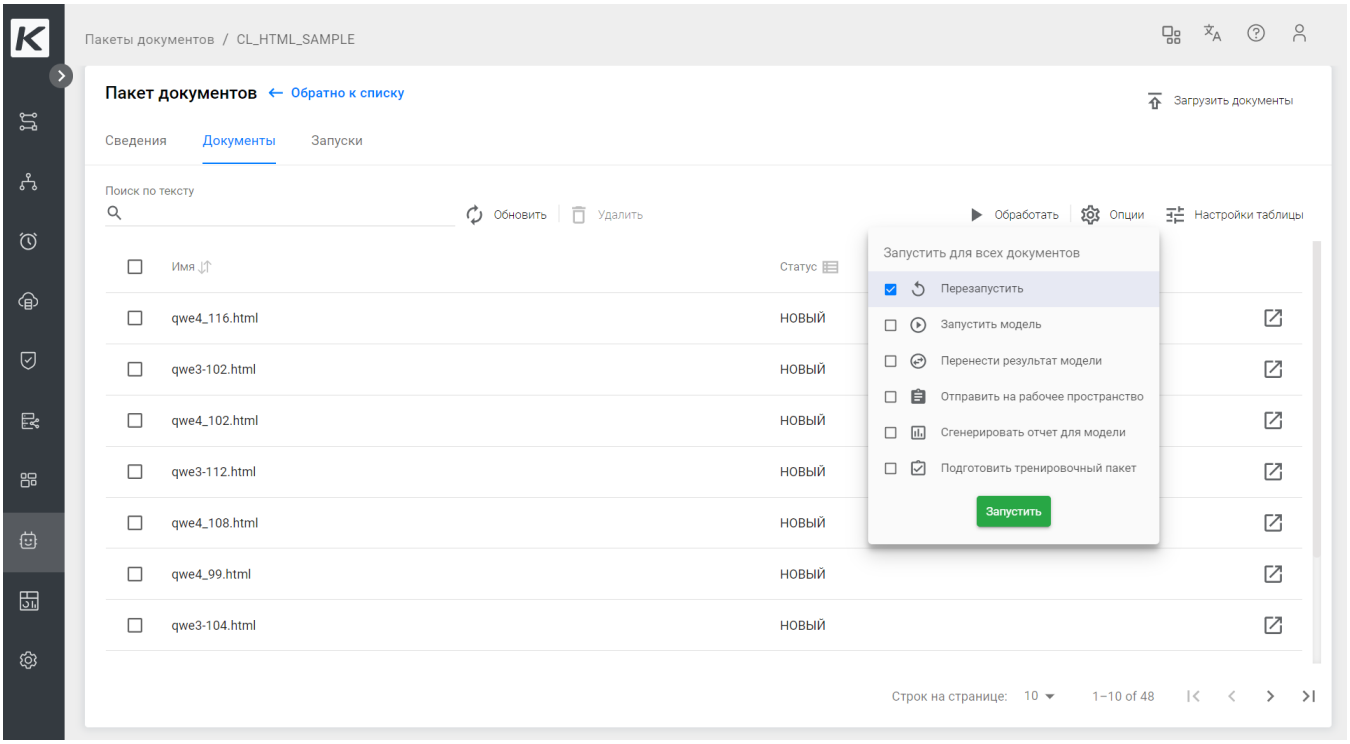
Пакет документов можно создать после загрузки ZIP-архива, выбрав тип документов и обработчик документов. Документы из вашего пакета документов будут загружены в хранилище файлов. Сведения о каждом документе в пакете документов можно найти, открыв его:



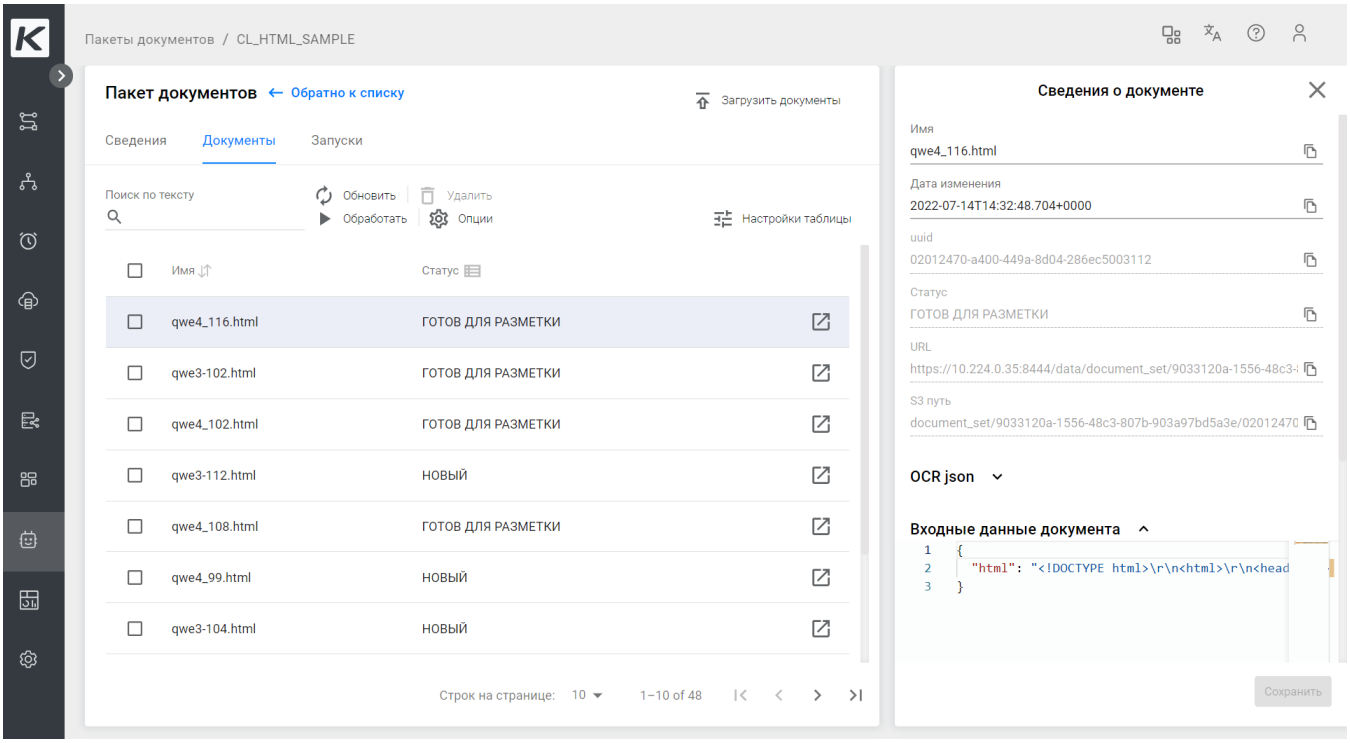
Поскольку этот Пакет документов является новым, **Входные данные документа**, **Результат работы пользователя** и **Результат модели** пусты.

Подготовка входных данных документов

Входные данные документов — это данные, которые поступают в качестве входных данных для пользовательских задач. Обработчик документов отвечает за подготовку входных данных. Для запуска обработчика документов и подготовки входных данных документов необходимо выбрать **Перезапустить** в меню действий **Обработать**, а затем нажать кнопку **Запустить**:

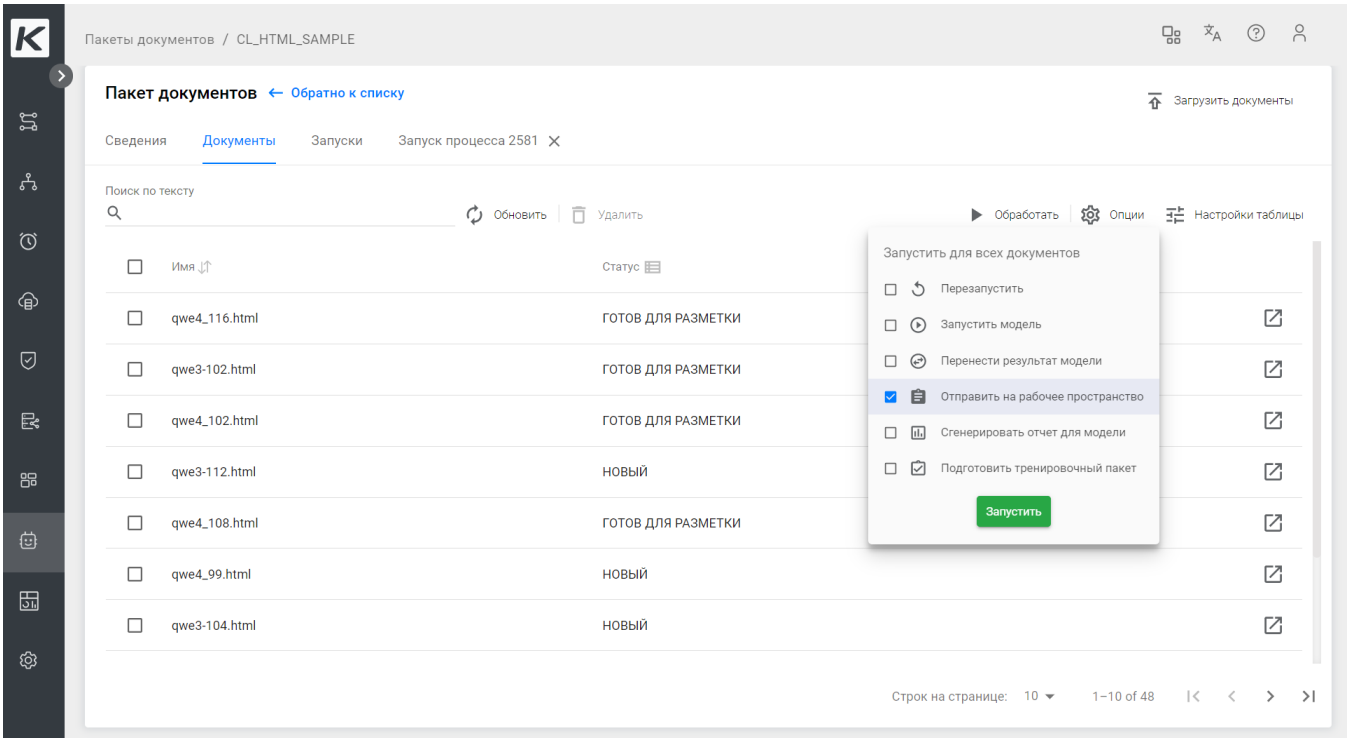


Обработчик документов начнет работать. Вы можете отслеживать ход его работы в разделе **Управление запусками**. Подождите, пока все документы получат статус READY_FOR_TAGGING. В результате **Входные данные документа** будут сгенерированы, и вы сможете увидеть входные данные в пользовательской задаче.

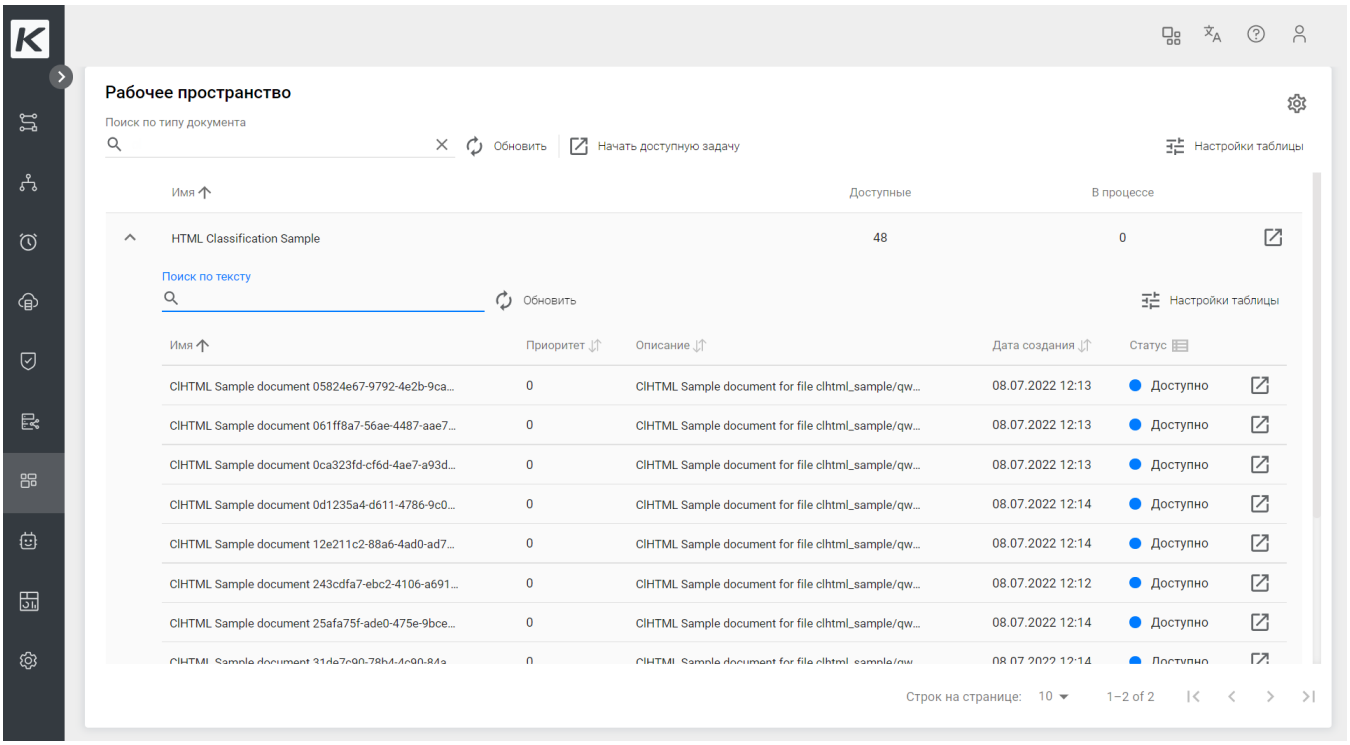


Отправка документов на рабочее пространство и их разметка

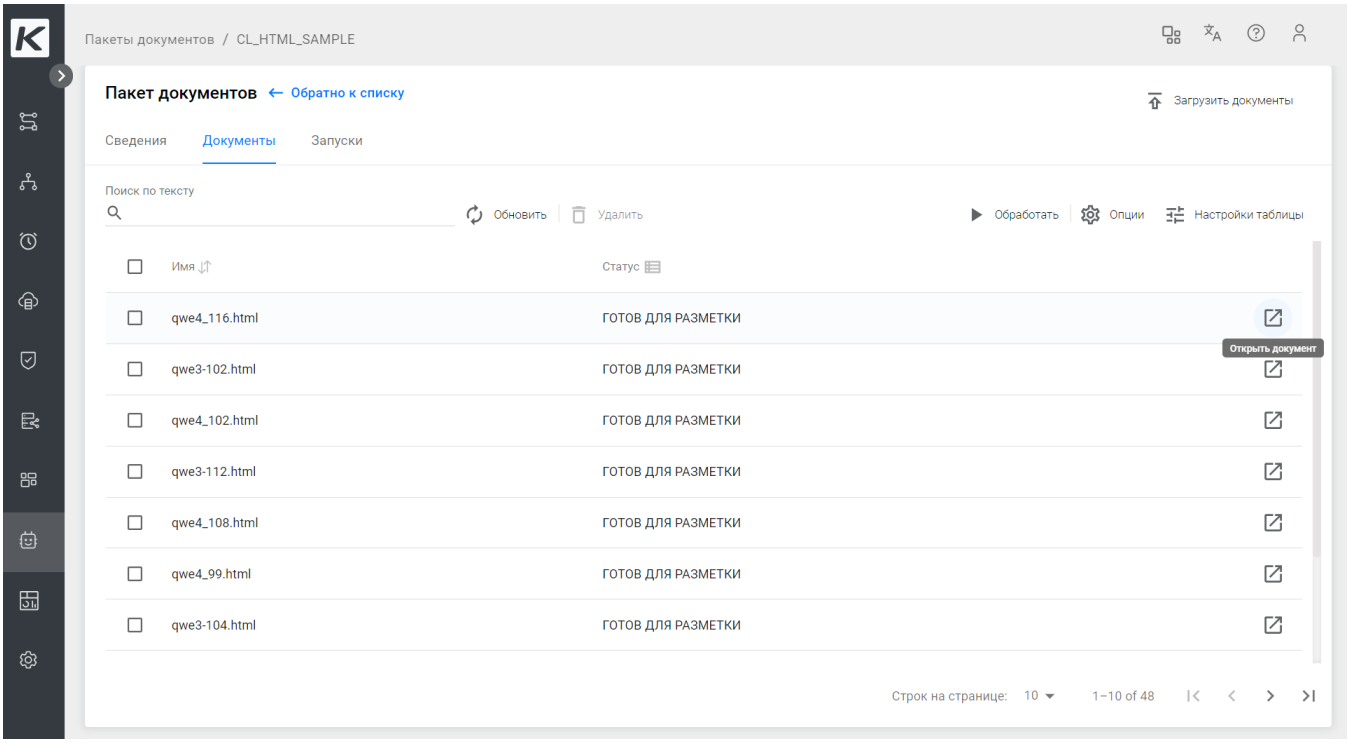
Следующим шагом является непосредственно классификация документов. Этот результат классификации будет использоваться алгоритмами машинного обучения для обучения модели. Чтобы пакет документов можно было переместить на Рабочее пространство и позволить пользователям классифицировать их с помощью пользовательской задачи, необходимо выбрать действие **Отправить на рабочее пространство**:



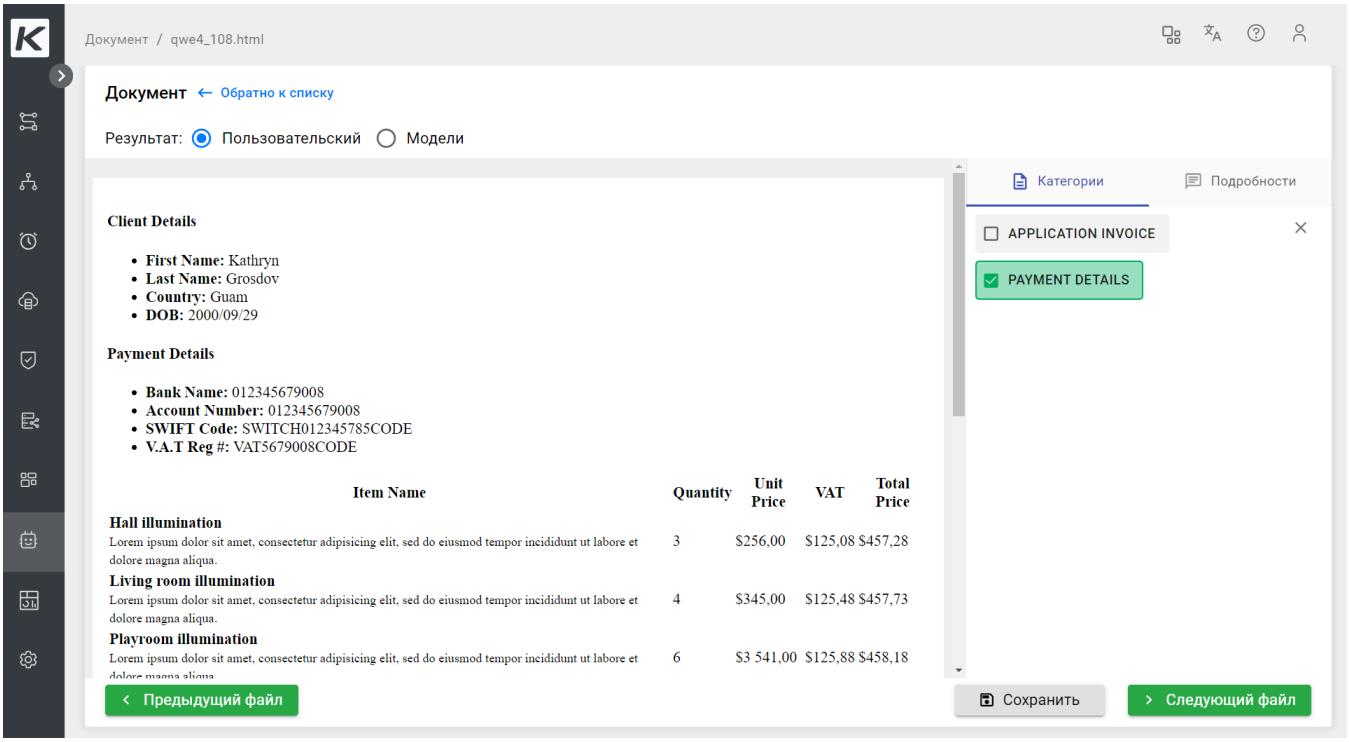
Документы должны отображаться на Рабочем пространстве в соответствующих типах документов. Чтобы классифицировать их, нажмите кнопку **Начать работу**:



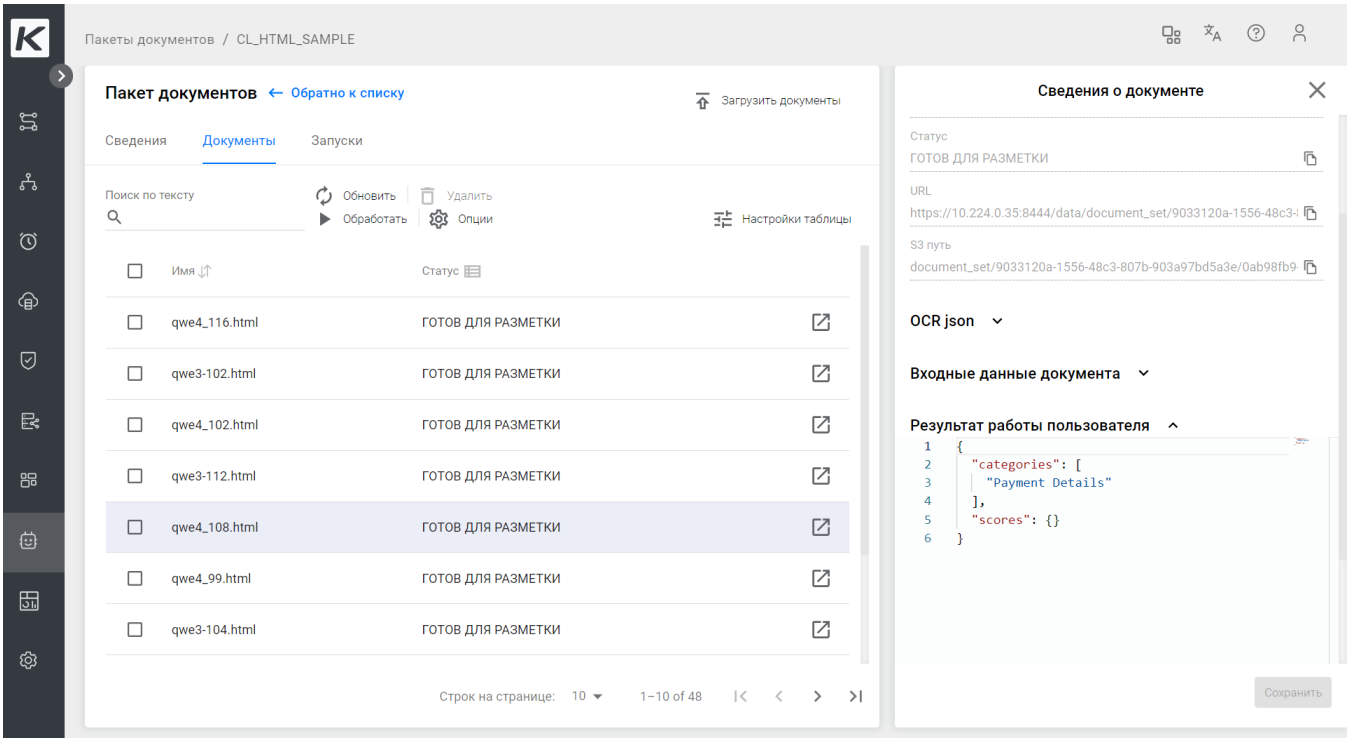
Документ также можно классифицировать, не отправляя на **Рабочее пространство**. Нажмите на кнопку **Открыть документ** в строке с документом на вкладке **Документы** страницы **Пакет документов:**



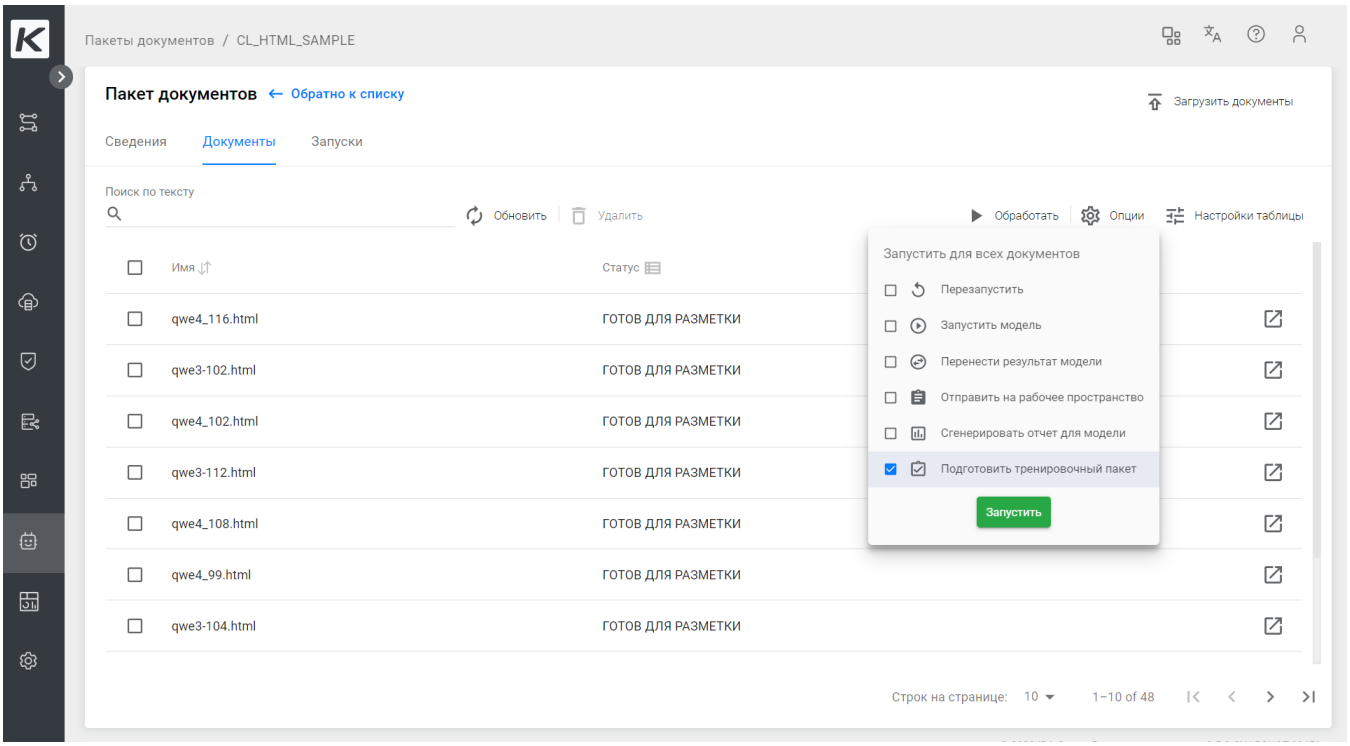
Вы можете начинать классифицировать документы:



После того, как документ классифицирован, для него генерируется **Результат работы пользователя**:



Чтобы завершить подготовку тренировочного набора данных, нажмите **Обработать - Подготовить тренировочный пакет** и подтвердите свое действие с помощью кнопки **Запустить**:

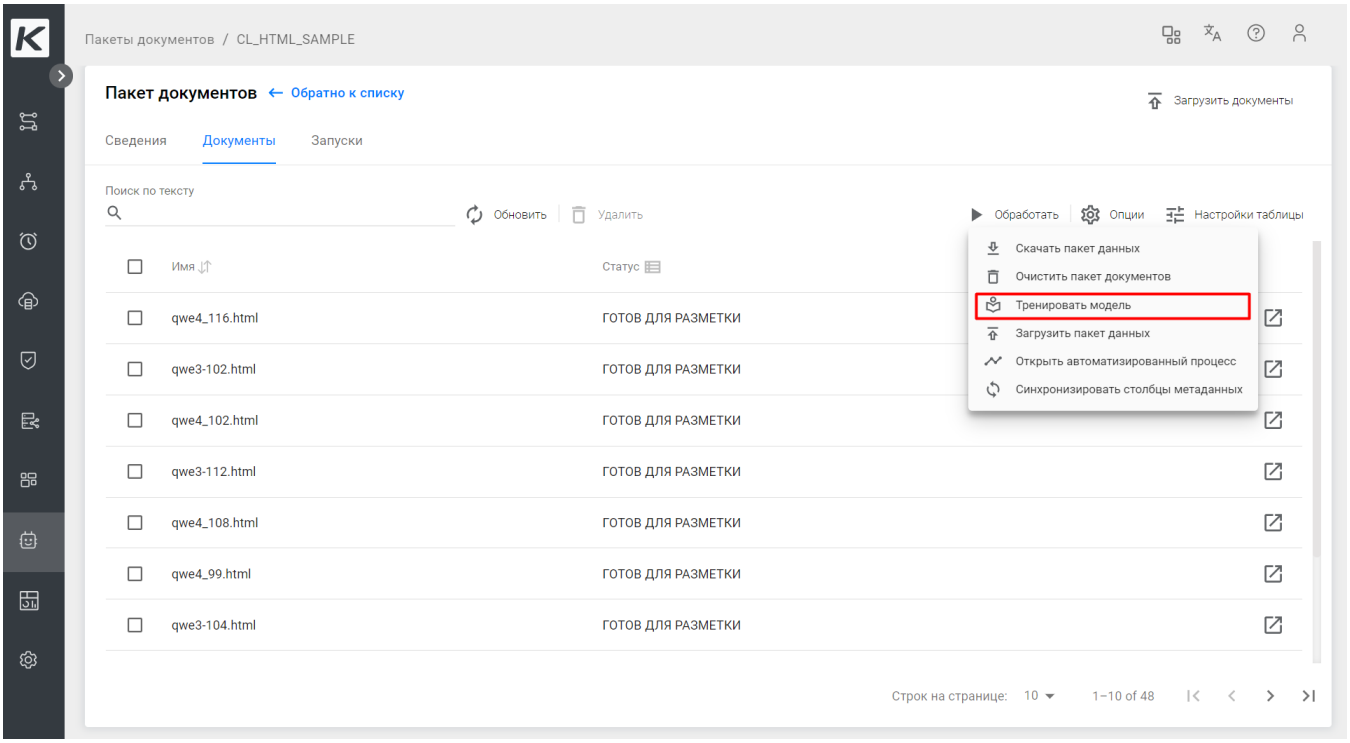


Когда вы выполнили все эти действия, можете начинать тренировать вашу модель.

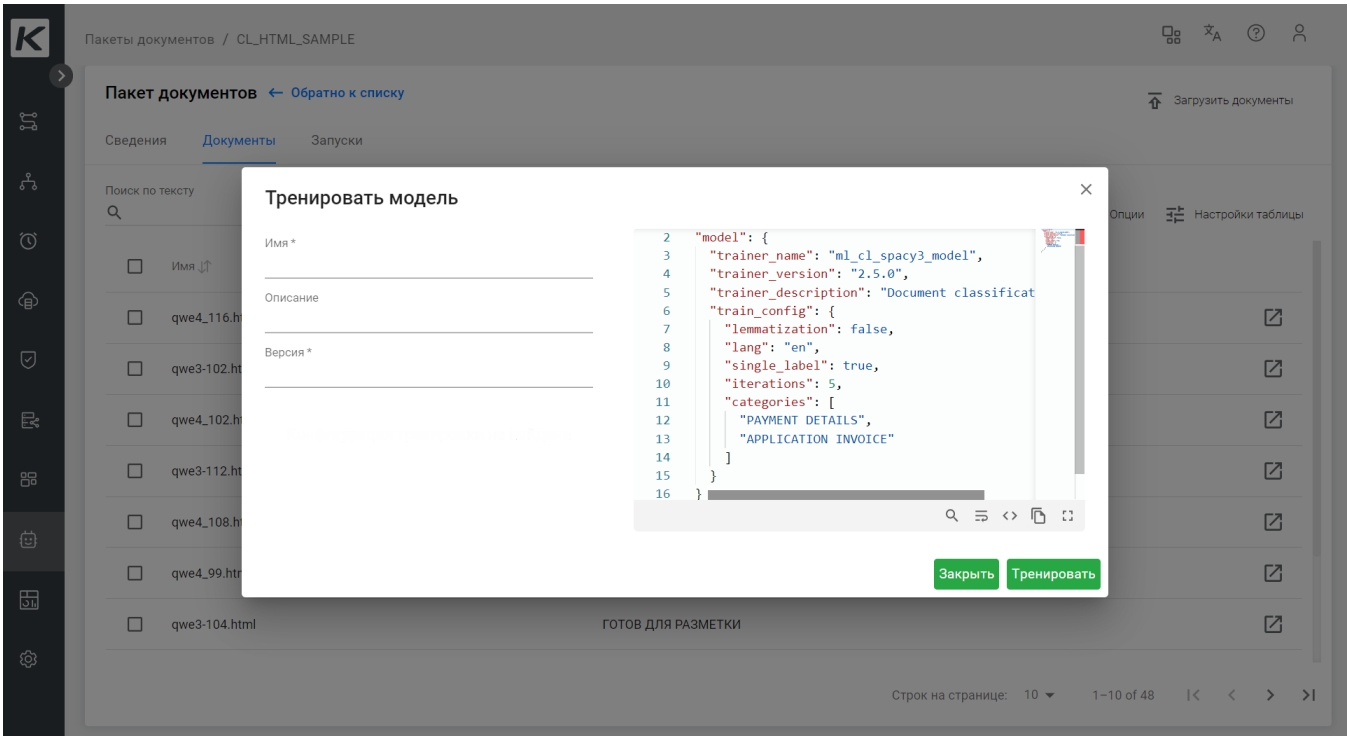
Шаг 3. Тренировка модели машинного обучения (Классификация)

После того как тренировочный набор подготовлен и все документы классифицированы, можно тренировать модель машинного обучения.

Перейдите на вкладку **Документы** страницы **Пакет документов**, и нажмите **Опции - Тренировать модель**.



Появится следующее диалоговое окно:

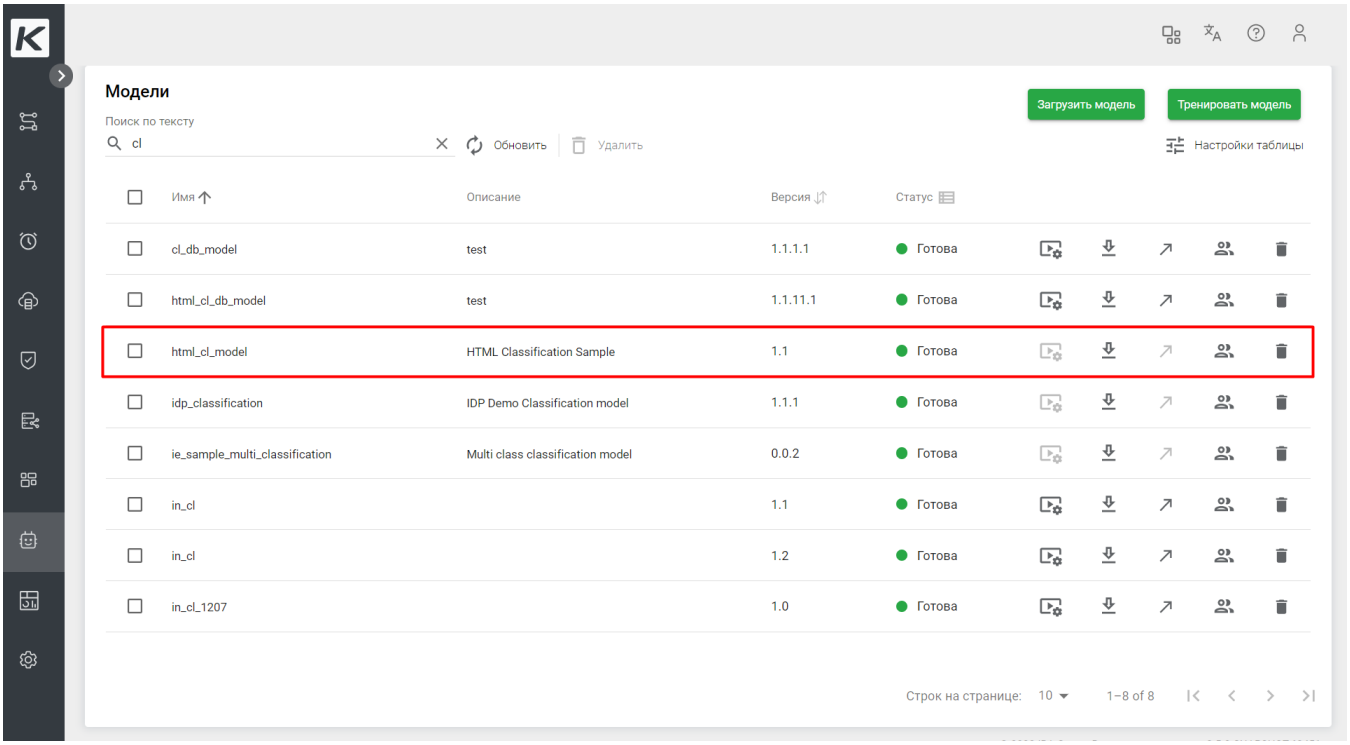


Заполните поля:

- **Имя** модели,
- **Описание** модели,
- **Версия** модели.

Конфигурация тренировки создается автоматически в формате JSON после выбора пакета тренировочных документов. Для получения подробной информации см. статью: Модели классификации.

После того, как вы нажмете кнопку **Тренировать** в диалоговом окне, подождите, пока тренировка модели не будет завершена, и ваша модель не получит статус **Готова**. Статус тренировки модели отображается на странице **Машинное обучение - Модели**:



Разработка автоматизированного процесса (Классификкация)

Шаг 1. Подготовка входных документов (Классификация)

Данный шаг полностью соответствует первому шагу разработки автоматизированного процесса извлечения информации: Шаг 1. Подготовка входных документов (Извлечение информации).

Пример кода

Как и на первом шаге автоматизированного процесса извлечения информации, мы собираемся реализовать сканирование хранилища Канцлер RPA на наличие новых входящих счетов-фактур.

Автоматизированный процесс выглядит идентично процессу извлечения информации за исключением названий пакетов, названия автоматизированного процесса и конфигураций:

InvoiceClassificationSample.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.samples.classification.task.GetIncomingInvoices;
import lombok.extern.slf4j.Slf4j;

@ApModuleEntry(name = "InvoiceClassificationSample")
@Slf4j
public class InvoiceClassificationSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        return doNotCareOfResult();
    }
}
```

Идентична и задача, которая сканирует все PDF-файлы из определенной папки в файловом хранилище, а затем перемещает его в другую рабочую папку:

GetIncomingInvoices.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.utils.storage.StorageManager;
import com.iba.samples.classification.ExportConstants;
import com.iba.samples.classification.PropertyConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;

import javax.inject.Inject;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

@ApTaskEntry(name = "Get Documents from Storage")
@Slf4j
@InputToOutput
```

```
public class GetIncomingInvoices extends ApTask {

    @Inject
    private StorageManager storageManager;

    @Output(ExportConstants.INVOICE_DOCUMENT_S3_PATHS_LIST)
    private List<String> invoicesS3PathsList = new ArrayList<>();

    private String s3Bucket;
    private String s3FolderToScan;

    private String ocrS3WorkBucket;
    private String ocrS3WorkFolder;

    @AfterInit
    public void init() {
        s3Bucket = getConfigurationService().getConfiguration().get(PropertyConstants.S3_BUCKET);
        s3FolderToScan = getConfigurationService().getConfiguration().get(PropertyConstants.S3_FOLDER_TO_SCAN);

        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        ocrS3WorkFolder = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_FOLDER);
    }

    @Override
    public void execute() throws Exception {
        List<String> filesPathList = storageManager.listFiles(s3Bucket, s3FolderToScan, ".*\\.pdf");
        log.info("There're {} document(s) found to process on S3 bucket '{}' in folder '{}'", filesPathList.
size(), s3Bucket, s3FolderToScan);
        for (String s3Path : filesPathList) {
            String newS3Path = moveFile(s3Bucket, s3Path, ocrS3WorkBucket, ocrS3WorkFolder);
            invoicesS3PathsList.add(newS3Path);
        }
    }

    private String moveFile(String s3SourceBucket, String s3SourceFilePath, String s3DestBucket, String
s3DestFolder) throws IOException {
        String fileName = FilenameUtils.getName(s3SourceFilePath);
        String resultS3Path = s3DestFolder + "/" + fileName;
        copyFile(s3SourceBucket, s3SourceFilePath, s3DestBucket, resultS3Path);
        storageManager.deleteFile(s3SourceBucket, s3SourceFilePath);
        return resultS3Path;
    }

    private void copyFile(String s3SourceBucket, String s3SourceFilePath, String s3DestBucket, String
s3DestFilePath) throws IOException {
        try(InputStream fileInputStream = storageManager.getFile(s3SourceBucket, s3SourceFilePath)) {
            storageManager.uploadFile(s3DestBucket, s3DestFilePath, fileInputStream);
        }
    }
}
```

PropertyConstants.java

```
package com.iba.samples.classification;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
}
```

ExportConstants.java

```
package com.iba.samples.classification;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=classification-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
```

Шаг 2. Оцифровка документов с помощью OCR (Классификация)

Данный шаг полностью соответствует второму шагу разработки автоматизированного процесса извлечения информации: Шаг 2. Оцифровка документов с помощью OCR (Извлечение информации)

Пример кода

Автоматизированный процесс выглядит идентично процессу извлечения информации за исключением названий пакетов, названия автоматизированного процесса и конфигураций.

Класс AP (автоматизированный процесс):

InvoiceClassificationSample.java

```
package com.iba.samples.classification;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.classification.task.GetIncomingInvoices;
import com.iba.samples.classification.task.PrepareInputForOcrTask;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceClassificationSample")
@Slf4j
public class InvoiceClassificationSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }
}
```

Новый класс, который готовит входные данные для задачи OCR:

PrepareInputForOcrTask.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrFormats;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.classification.ExportConstants;
import com.iba.samples.classification.PropertyConstants;
import lombok.extern.slf4j.Slf4j;

import java.util.*;

@ApTaskEntry(name = "Prepare Input For OCR Task")
@Slf4j
@InputToOutput
public class PrepareInputForOcrTask extends ApTask {
    private static final String OCR_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY = "documents_bucket";
    private static final String OCR_TASK_CONFIGURATION_TESSERACT_OPTIONS_KEY = "tesseractOptions";
    private static final String OCR_TASK_CONFIGURATION_IMAGE_MAGICK_OPTIONS_KEY = "imageMagickOptions";
    private static final List<String> OCR_OUTPUT_FORMATS = Arrays.asList(OcrFormats.TEXT, OcrFormats.HOCR,
OcrFormats.JSON, OcrFormats.IMAGE);

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskData;

    private String ocrS3WorkBucket;
    private String ocrTesseractOptions;
    private String ocrImageMagickOptions;

    private Map<String, Object> ocrConfiguration = new HashMap<>();

    @AfterInit
    public void init() {
        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        ocrTesseractOptions = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_TESSERACT_OPTIONS);
        ocrImageMagickOptions = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_IMAGE_MAGICK_OPTIONS);

        ocrConfiguration.put(OCR_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY, ocrS3WorkBucket);
        ocrConfiguration.put(OCR_TASK_CONFIGURATION_TESSERACT_OPTIONS_KEY, MlTaskUtils.readObjectFromString
(ocrTesseractOptions, ArrayList.class));
        ocrConfiguration.put(OCR_TASK_CONFIGURATION_IMAGE_MAGICK_OPTIONS_KEY, MlTaskUtils.readObjectFromString
(ocrImageMagickOptions, ArrayList.class));
    }

    @Override
    public void execute() throws Exception {
        log.info("Prepare input for document in '{}' with ocr task with configuration: {}", documentS3Path,
ocrConfiguration);

        ocrTaskData = new OcrTaskData();
        ocrTaskData.setDocumentLocation(documentS3Path);
        ocrTaskData.getFormats().addAll(OCR_OUTPUT_FORMATS);
        ocrTaskData.setConfiguration(ocrConfiguration);
    }
}
```

PropertyConstants.java

```
package com.iba.samples.classification;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";
}
```

ExportConstants.java

```
package com.iba.samples.classification;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=classification-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]
```

Шаг 3. Применение и запуск модели машинного обучения (Классификация)

Данный шаг полностью соответствует третьему шагу разработки автоматизированного процесса извлечения информации: Шаг 3. Применение и запуск модели машинного обучения (Извлечение информации)

Пример кода

Автоматизированный процесс выглядит идентично процессу извлечения информации за исключением названий пакетов, названия автоматизированного процесса и конфигураций:

InvoiceProcessingSample.java

```
package com.iba.samples.classification;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.classification.task.GetIncomingInvoices;
import com.iba.samples.classification.task.PrepareInputForMlTask;
import com.iba.samples.classification.task.PrepareInputForOcrTask;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceClassificationSample")
@Slf4j
public class InvoiceClassificationSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }

}
```

Код **PrepareInputForMlTask:**

PrepareInputForMlTask.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.classification.ExportConstants;
import com.iba.samples.classification.PropertyConstants;
import lombok.extern.slf4j.Slf4j;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@ApTaskEntry(name = "Prepare input for ML Task")
@Slf4j
@InputToOutput
public class PrepareInputForMlTask extends ApTask {
    private static final String ML_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY = "documents_bucket";

    @Input(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskOutput;

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(MlTask.ML_TASK_DATA_KEY)
    private MlTaskData mlTaskData;

    private String modelName;
    private String modelVersion;

    private String documentId;

    private Map<String, Object> mlConfiguration = new HashMap<>();

    @AfterInit
    public void init() {
        modelName = getConfigurationService().get(PropertyConstants.CLASSIFICATION_MODEL_NAME);
        modelVersion = getConfigurationService().get(PropertyConstants.CLASSIFICATION_MODEL_VERSION);

        documentId = UUID.randomUUID().toString();

        String ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        mlConfiguration.put(ML_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY, ocrS3WorkBucket);
    }

    @Override
    public void execute() {
        log.info("Creating ML task input for document located at '{}' with assigned id '{}'", documentS3Path,
documentId);

        mlTaskData = new MlTaskData();
        mlTaskData.setModelName(modelName);
        mlTaskData.setModelVersion(modelVersion);
        mlTaskData.setConfiguration(mlConfiguration);
        MlTaskUtils.prepareClMlTask(documentId, mlTaskData, ocrTaskOutput);
    }

}
```

PropertyConstants.java

```
package com.iba.samples.classification;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";

          String CLASSIFICATION_MODEL_NAME = "classification_model.name";
        String CLASSIFICATION_MODEL_VERSION = "classification_model.version";
}
```

ExportConstants.java

```
package com.iba.samples.classification;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

classification_model.name=incoming_documents_classification
classification_model.version=1.0
```

Шаг 4. Чтение выходных данных машинного обучения и маршрутизация документов в пользовательскую задачу (Классификация)

Для считывания выходных данных модели машинного обучения необходимо получить значение по ключу "ml.task.data" из объекта TaskOutput. Оно содержит объект MlTaskData, который содержит данные результатов, поэтому вы можете получить окончательную категорию и ее оценку.

Существует специальный метод MlTaskUtils.createClHtOutputMap(mlTaskData), который преобразует выходные данные модели классификации в словарь соответствий (Map):

```
MlTaskData mlTaskDataOutput = previousTaskStepOutput.get(MlTask.ML_TASK_DATA_KEY, MlTaskData.class);
Map classificationResult = MlTaskUtils.createClHtOutputMap(mlTaskDataOutput);

String resultCategory = ((List)classificationResult.get("categories")).get(0).toString();
Double resultScore = (Double) ((Map)classificationResult.get("scores")).get(resultCategory);
```

В случае, если итоговые категории были классифицированы с недостаточной уверенностью, принимается решение о создании пользовательской задачи для обработки документа. Подробнее см. Создание пользовательской задачи.

Чтобы создать пользовательскую задачу для обработки документов, необходимо подготовить специальные входные данные для встроенного класса HumanTask (пользовательская задача). Входные данные должны содержать объект HumanTaskData под ключём "human.task.data" на входе.

Пример кода

В класс AP (автоматизированный процесс) мы добавляем дополнительную функцию, которая выполняется асинхронно и считывает выходные данные с шага работы модели машинного обучения. Если оценка категории результатов ниже порога достоверности (в нашем случае 30%), асинхронно запускается пользовательскуая задача. Если категория классифицирована с достаточной уверенностью, дополнительных действий не требуется (возвращаем уже завершенное Completable Future с выходными данными с предыдущего шага):

InvoiceClassificationSample.java

```
package com.iba.samples.classification;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.classification.task.*;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceClassificationSample")
@Slf4j
public class InvoiceClassificationSample extends ApModule {
        private static final Double CLASSIFICATION_CONFIDENCE_THRESHOLD = 0.3;

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);
```

```
        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class))
                                .thenCompose(previousTaskStepOutput -> {
                                    MlTaskData mlTaskDataOutput = previousTaskStepOutput.get(MlTask.
ML_TASK_DATA_KEY, MlTaskData.class);
                                                                        Map classificationResult = MlTaskUtils.
createClHtOutputMap(mlTaskDataOutput);

                                                                        String resultCategory = ((List)
classificationResult.get("categories")).get(0).toString();
                                                                        Double resultScore = (Double) ((Map)
classificationResult.get("scores")).get(resultCategory);

                                                                        if(resultScore <
CLASSIFICATION_CONFIDENCE_THRESHOLD) {
                                                                                return execute
(previousTaskStepOutput, PrepareInputForHumanTask.class)
                                                                                                .thenCompose
(execute(HumanTask.class));
                                                                        } else {
                                                                                return CompletableFuture.
completedFuture(previousTaskStepOutput);
                                                                        }
                                });

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }
}
```

Чтобы создать пользовательскую задачу для обработки документа, добавляется дополнительный шаг, который отвечает за подготовку входных данных для него. Кроме того, мы специально используем логическую переменную для вывода, которая помогает на следующих шагах определить, была ли запись обработана пользователем или нет.

PrepareInputForHumanTask.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTaskData;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.kanclerrpa.utils.storage.StorageManager;
import com.iba.samples.classification.ExportConstants;
import com.iba.samples.classification.PropertyConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;
```

```
import javax.inject.Inject;
import java.util.Map;

@ApTaskEntry(name = "Prepare Human Task")
@Slf4j
@InputToOutput
public class PrepareInputForHumanTask extends ApTask  {

    @Inject
    private StorageManager storageManager;

    @Input(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskOutput;

    @Input(MlTask.ML_TASK_DATA_KEY)
    private MlTaskData mlTaskOutput;

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(HumanTask.HUMAN_TASK_DATA_KEY)
    private HumanTaskData humanTaskData;

    @Output(ExportConstants.DOCUMENT_PROCESSED_BY_HUMAN)
    private boolean documentProcessedByHuman = true;

    private String ocrS3WorkBucket;
    private String documentType;

    @AfterInit
    public void init() {
        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        documentType = getConfigurationService().getConfiguration().get(PropertyConstants.DOCUMENT_TYPE);
    }

    @Override
    public void execute() throws Exception {
        String documentName = FilenameUtils.getName(documentS3Path);
        log.info("Creating human task for document {} with document type {}", documentName, documentType);

        humanTaskData = new HumanTaskData();
                 humanTaskData.setInputJson(MlTaskUtils.createClHtInputMap(ocrTaskOutput, storageManager,
ocrS3WorkBucket));
                humanTaskData.setOutputJson(MlTaskUtils.createClHtOutputMap(mlTaskOutput));

        humanTaskData.setName(documentName);
        humanTaskData.setDocumentType(documentType);
        humanTaskData.setDescription("Invoice Document: " + documentName);

        log.info("Sending task into workspace {} ", humanTaskData);
    }

}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";
```

```
    String IE_MODEL_NAME = "ie_model.name";
    String IE_MODEL_VERSION = "ie_model.version";

    String DOCUMENT_TYPE = "document_type";
}
```

ExportConstants.java

```
package com.iba.samples.classification;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
    String DOCUMENT_PROCESSED_BY_HUMAN = "DOCUMENT_PROCESSED_BY_HUMAN";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=classification-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

classification_model.name=incoming_documents_classification
classification_model.version=1.0

document_type=Incoming Documents Classification
```

Шаг 5. Результат процесса извлечения (Классификация)

После того, как документ классифицирован моделью машинного обучениея или пользователем, документы направляются в разные потоки в зависимости от категории.

Основная цель данного шага - продемонстрировать, как мы получаем извлеченную категорию в зависимости от источника результата классификации (модель машинного обучения или пользовательская задача).
Поскольку у нас есть специальная логическая выходная переменная, которую мы добавили на предыдущем шаге, мы можем легко определить, обрабатывается ли документ пользователем или моделью, поэтому в зависимости от этого есть разные способы извлечения результата:

```
if(documentProcessedByHuman) {
        resultCategory = ((List)humanTaskOutput.getOutputJson().get("categories")).get(0).toString();
        resultScore = 1d; //confidence is 100% as it was processed by human
} else {
        Map classificationResult = MlTaskUtils.createClHtOutputMap(mlTaskOutput);

        resultCategory = ((List)classificationResult.get("categories")).get(0).toString();
        resultScore = (Double) ((Map)classificationResult.get("scores")).get(resultCategory);
}
```

Пример кода

Оконательная версия кода AP (автоматизированного процесса):

InvoiceClassificationSample.java

```
package com.iba.samples.classification;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.classification.task.*;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.CompletableFuture;
import java.util.function.Function;

@ApModuleEntry(name = "InvoiceClassificationSample")
@Slf4j
public class InvoiceClassificationSample extends ApModule {
    private static final Double CLASSIFICATION_CONFIDENCE_THRESHOLD = 0.3;

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);
```

```
                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class))
                                .thenCompose(processAndRouteMlTaskResult())
                                .thenCompose(execute(PrepareResult.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }

    private Function<TaskOutput, CompletableFuture<TaskOutput>> processAndRouteMlTaskResult() {
        return previousTaskStepOutput -> {
            MlTaskData mlTaskDataOutput = previousTaskStepOutput.get(MlTask.ML_TASK_DATA_KEY, MlTaskData.class);
            Map classificationResult = MlTaskUtils.createClHtOutputMap(mlTaskDataOutput);

            String resultCategory = ((List)classificationResult.get("categories")).get(0).toString();
            Double resultScore = (Double) ((Map)classificationResult.get("scores")).get(resultCategory);

            if(resultScore < CLASSIFICATION_CONFIDENCE_THRESHOLD) {
                return execute(previousTaskStepOutput, PrepareInputForHumanTask.class)
                        .thenCompose(execute(HumanTask.class));
            } else {
                return CompletableFuture.completedFuture(previousTaskStepOutput);
            }
        };
    }
}
```

Заключительный шаг **PrepareResult** генерирует выходные данные с окончательным текстом:

PrepareResult.java

```
package com.iba.samples.classification.task;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTaskData;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.classification.ExportConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;

import java.util.List;
import java.util.Map;

@ApTaskEntry(name = "Validate invoice data")
@Slf4j
@InputToOutput
public class PrepareResult extends ApTask {

    @Input(value = ExportConstants.DOCUMENT_PROCESSED_BY_HUMAN, required = false)
    private boolean documentProcessedByHuman;
```

```
    @Input(value = HumanTask.HUMAN_TASK_DATA_KEY, required = false)
    private HumanTaskData humanTaskOutput;

    @Input(value = MlTask.ML_TASK_DATA_KEY, required = false)
    private MlTaskData mlTaskOutput;

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(ExportConstants.PROCESSING_RESULT)
    private String processingResult;

    @Override
    public void execute() throws Exception {
        String documentName = FilenameUtils.getName(documentS3Path);
        String resultCategory;
        Double resultScore;
        if(documentProcessedByHuman) {
            resultCategory = ((List)humanTaskOutput.getOutputJson().get("categories")).get(0).toString();
            resultScore = 1d; //confidence is 100% as it was processed by human
        } else {
            Map classificationResult = MlTaskUtils.createClHtOutputMap(mlTaskOutput);

            resultCategory = ((List)classificationResult.get("categories")).get(0).toString();
            resultScore = (Double) ((Map)classificationResult.get("scores")).get(resultCategory);
        }

        processingResult = "Classification result: document '" + documentName + "' relates to category '" +
resultCategory + "' with confidence " + resultScore;
        log.info(processingResult);
    }
}
```

PropertyConstants.java

```
package com.iba.samples.classification;

public interface PropertyConstants {
        String S3_BUCKET = "s3.bucket";
        String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

        String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
        String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
        String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
        String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";

        String CLASSIFICATION_MODEL_NAME = "classification_model.name";
        String CLASSIFICATION_MODEL_VERSION = "classification_model.version";

        String DOCUMENT_TYPE = "document_type";
}
```

ExportConstants.java

```
package com.iba.samples.classification;

public interface ExportConstants {
        String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
        String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
        String DOCUMENT_PROCESSED_BY_HUMAN = "DOCUMENT_PROCESSED_BY_HUMAN";
        String PROCESSING_RESULT = "PROCESSING_RESULT";
}
```

apm_run.properties

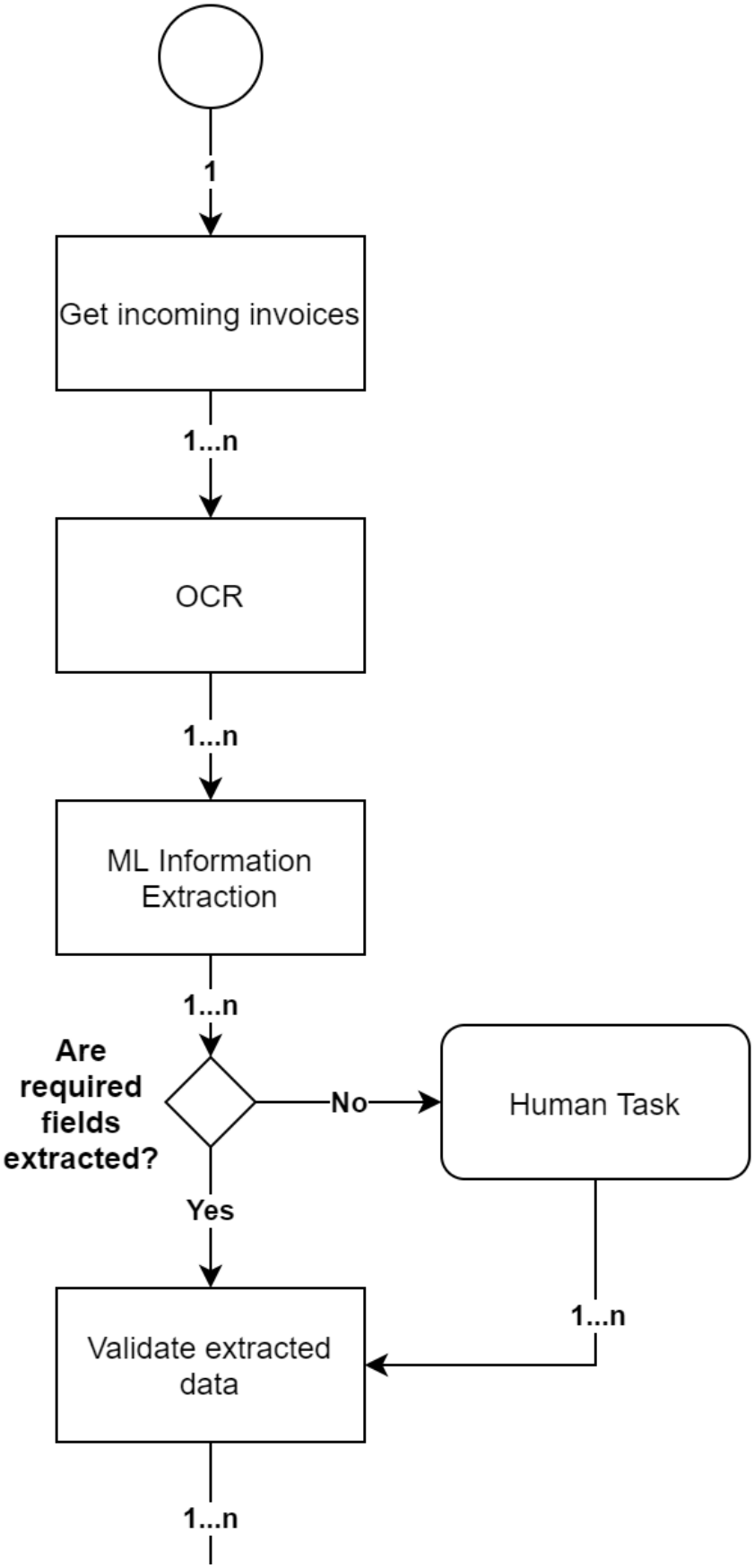
```
# KanclerRPA Client Configuration File
s3.bucket=classification-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

classification_model.name=incoming_documents_classification
classification_model.version=1.0

document_type=Incoming Documents Classification
```

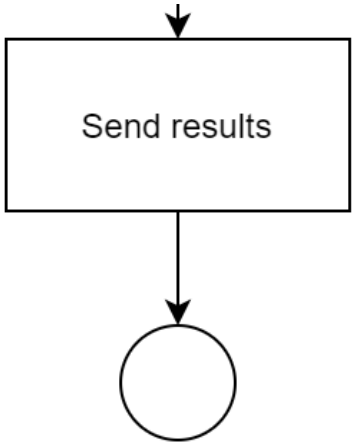
Процесс извлечения информации

Эта статья охватывает сквозной процесс разработки автоматизации извлечения информации и состоит из 2 разделов:

1. Настройка пользовательской задачи и тренировка модели
2. Разработка автоматизированного процесса (Извлечение информации)

В целом, процесс автоматизации извлечения информации выглядит так, как показано на диаграмме ниже, и он должен обрабатывать все записи асинхронно параллельно:


1
Get incoming invoices
1...n
OCR
1...n
ML Information Extraction
1...n
Are required fields extracted?
No
Human Task
Yes
1...n
Validate extracted data
1...n



Проанализируйте тип документа, чтобы определить следующие шаги.

В зависимости от формата файла входящих документов мы предлагаем различные способы работы с ним.

В таблице ниже вы можете увидеть 2 различных предлагаемых подхода:

- **Машинное обучение** - если вы решили использовать этот подход, продолжайте следовать этому руководству.
- **Основанный на правилах** - если вы решили использовать подход, основанный на правилах, вы можете пропустить следующие шаги этого руководства и использовать предложенный способ извлечения информации из документов.

Формат документа	Рекомендуемый подход	Комментарий
PDF	*Машинное обучение*	PDF-документы можно разделить на три разных типа, в зависимости от того, как файл был создан. То, как он был первоначально создан, также определяет, можно ли получить доступ к содержимому PDF-документа (текст, изображения, таблицы) или оно «заблокировано» в изображении страницы. ***1. "Истинные" или созданные в цифровом виде PDF-файлы*** Цифровые PDF-файлы, также известные как «истинные» PDF-файлы, создаются с помощью программного обеспечения, такого как Microsoft® Word, Excel®, или с помощью функции «печати» в программном приложении (виртуальный принтер). Они состоят из текста и изображений. И символы в тексте, и метаинформация имеют электронное символьное обозначение. С ABBYY FineReader 14 вы можете легко выполнять поиск в этих PDF-файлах, а также выбирать, редактировать или удалять текст так же, как вы делаете это в других редактируемых форматах, таких как Microsoft® Word. Изображения в документах, созданных в цифровом виде, можно изменять, перемещать или удалять. ***2. Отсканированные PDF-файлы*** При сканировании печатных документов на МФУ и офисных сканерах или при преобразовании изображения с камеры, jpg, tiff или снимка экрана в PDF содержимое «запирается» в изображении, похожем на снимок. Такие PDF-документы, содержащие только изображения, содержат только отсканированные/сфотографированные изображения страниц без текстового слоя. Следовательно, файлы PDF, содержащие только изображения, недоступны для поиска, а их текст обычно нельзя изменить или разметить. PDF-файл можно сделать доступным для поиска, применив OCR, с помощью которого добавляется текстовый слой, обычно под изображением страницы.

		3. PDF-файлы с возможностью поиска PDF-файлы с возможностью поиска обычно являются результатом применения OCR (оптического распознавания символов) к отсканированным PDF-файлам или другим документам на основе изображений. В процессе распознавания текста анализируются и «считываются» символы и структура документа. Текстовый слой добавляется к слою изображения, обычно расположенному под ним. Такие PDF-файлы почти неотличимы от исходных документов и полностью доступны для поиска. Текст в доступных для поиска документах PDF можно выбирать, копировать и размечать. Рабочий процесс обработки PDF зависит от типа PDF. В случае с доступными для поиска (**случай 3**) или «истинными» PDF-файлами (**случай 1**) мы можем получить содержимое файлов, используя библиотеку **apache pdfbox**. PDF-файлы (**случай 2**) должны сначала пройти этап оптического распознавания символов.
Image	*Машинное обучение*	Этот тип документов обычно поставляется в виде сканов. Типичный рабочий процесс для изображений выглядит следующим образом: 1. Убедитесь, что изображение имеет достаточное разрешение (не менее 300 dpi) и при необходимости конвертируйте его (используя библиотеку **ImageMagick**). 2. OCR распознавание
Excel	*Основанный на правилах*	Оба подхода (**машинное обучение** и **основанный на правилах**) могут быть применены к этому типу документа, но, как правило, гораздо проще реализовать подход, основанный на правилах, потому что Excel является структурированным документом. Машинное обучение можно использовать, если у заказчика много разных шаблонов документов excel. Примите во внимание, что вы должны **преобразовать excel в html** перед отправкой документа в ручную задачу.
HTML	*Основанный на правилах*	Для html-файлов мы также можем использовать оба подхода (**машинное обучение** и **основанный на правилах**). Примите во внимание, что если у клиента есть хорошо структурированный формат HTML, вероятно, лучшим решением будет **использование xpath для извлечения данных из документов.** Только в случае, если вы не знаете структуру HTML, вам следует использовать **машинное обучение**. Обычно это происходит, когда вам нужно извлечь данные из не автоматически сгенерированного тела письма, так как люди пишут письмо без какого-либо предопределенного шаблона.
Простой текст	*Машинное обучение*	**Машинное обучение** предпочтительно для этого формата, но также может быть применен **подход на основе правил** (например, мы на 100% уверены, что номер счета-фактуры является первым словом в документе). Обратите внимание, что простой текст является предпочтительным вариантом для машинного обучения, поскольку этот формат не содержит никакой дополнительной информации (например, тегов html).
Другие форматы		Вы можете столкнуться с другими типами документов. Просмотрите структуру документов, чтобы принять правильное решение об использовании подхода

Настройка пользовательской задачи и тренировка модели машинного обучения (Извлечение информации)

Шаг 1. Создайте новый тип документа (Извлечение информации)

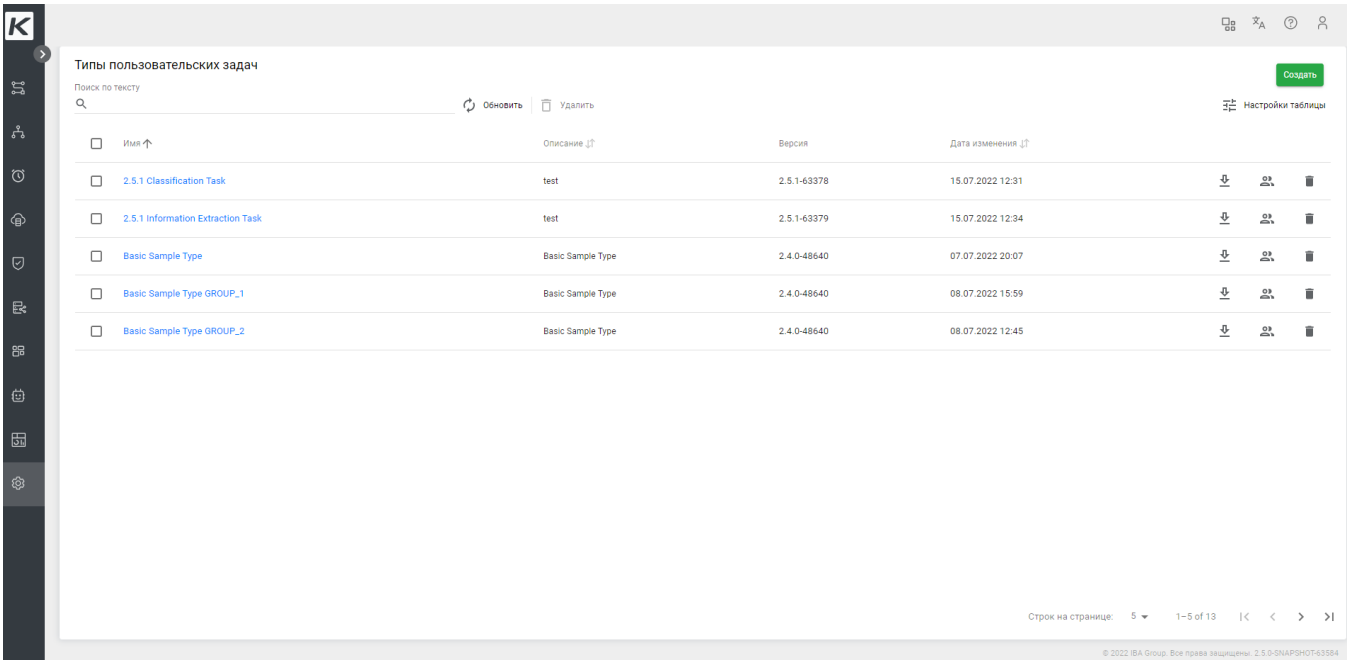
Чтобы описать, какие поля извлекать из документа, необходимо создать новый **тип документа. Тип документа** — это конфигурация, которая позволяет **пользовательской задаче** и **модели машинного обучения** знать данные, которые нужно получить из документа. Каждый тип документа относится к некоторым **типам пользовательских задач**.

Тип пользовательской задачи — это специальное небольшое веб-приложение, которое может считывать и анализировать конфигурацию типа документа и соответствующим образом отображать входные документы.

Канцлер RPA предоставляет 5 предопределенных типов пользовательских задач:

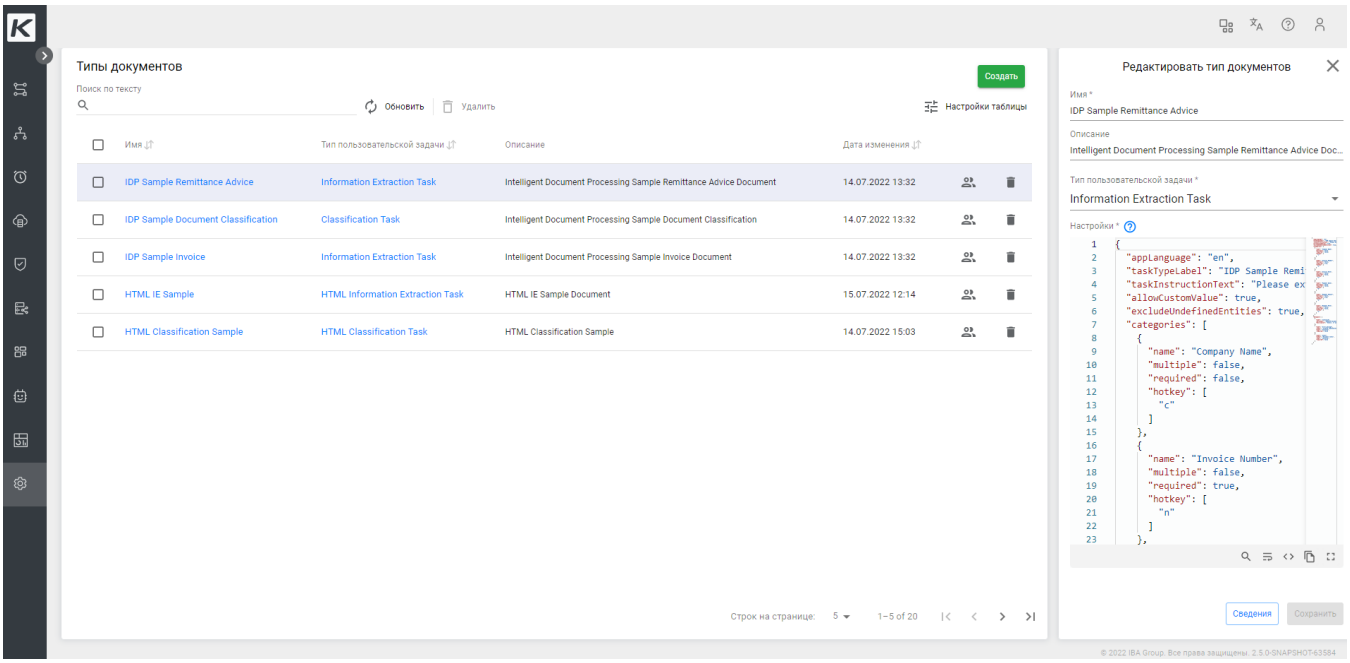
- Тип пользовательской задачи извлечения информации
- Тип пользовательской задачи классификации документов
- Тип пользовательской задачи заполнения формы
- Тип пользовательской задачи извлечения информации из HTML-документов
- Тип пользовательской задачи классификации HTML-документов

Для процесса извлечения информации необходимо создать тип документа, который относится к предварительно определенному типу пользовательской задачи для **извлечения информации** или предварительно определенному типу пользовательской задачи для извлечения информации в формате HTML(в зависимости от типа обрабатываемых документов):



Вы можете создавать свои собственные типы пользовательских задач для конкретных случаев . Обратитесь к статье Создание типа пользовательской задачи.

Для создания нового Типа документов нужно перейти в **Администрирование - Типы документов** и нажать на кнопку **Создать**. Необходимо предоставить Название, Описание, Тип пользовательской задачи и Настройки для нового типа документов: Типы документов.

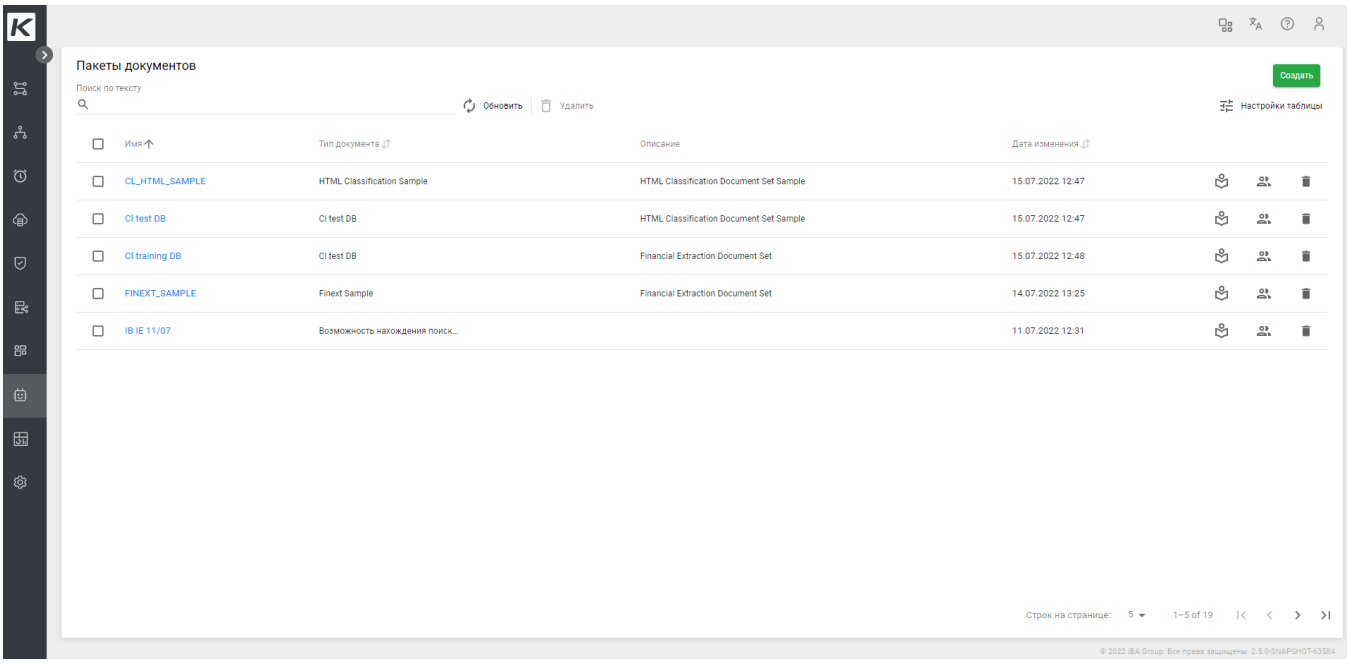


Параметры типа документов представляют собой конфигурацию в формате JSON и могут быть настроены в соответствии со следующей статьей: Настройки JSON-структуры типа документов.

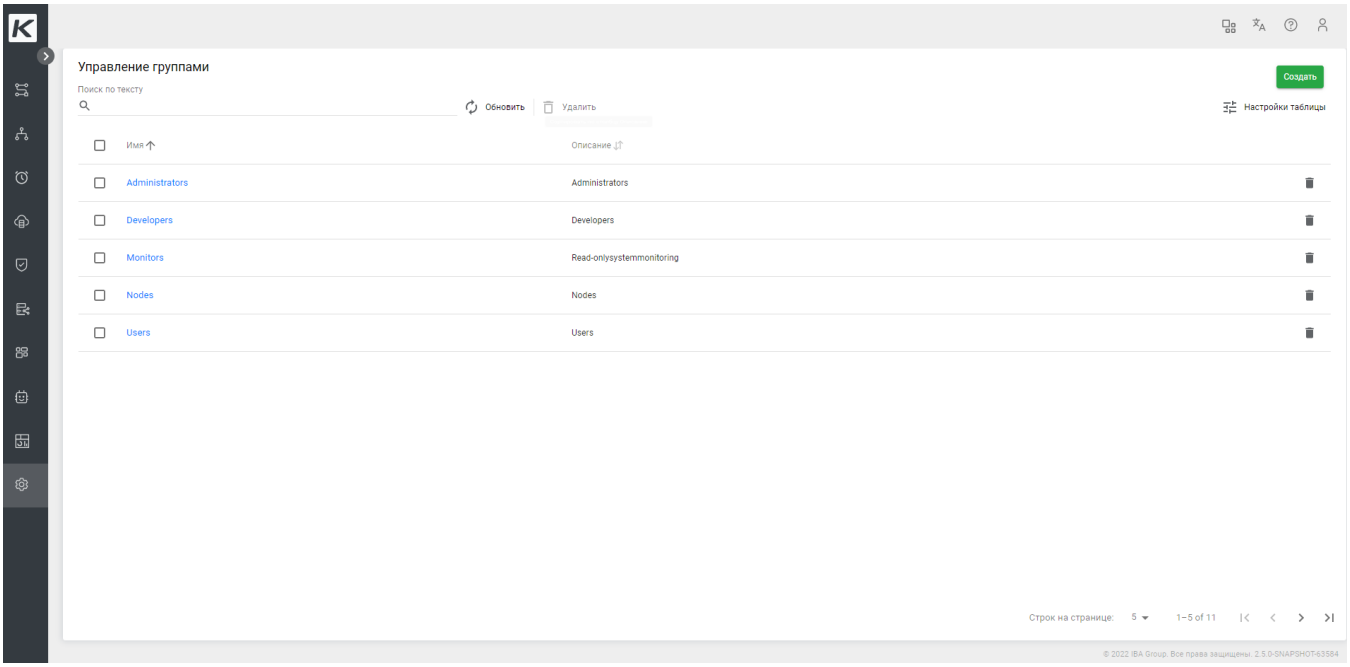
Ограничение доступа для типа документов

Каждый тип документа может быть ограничен, чтобы быть видимым только для определенной группы людей. Это полезно, когда нужно позволить разным командам работать только с типами документов, за которые они отвечают. Например, Отдел страхования должен иметь доступ к типу документов "страховые случаи", в то время как Отдел маркетинга должен иметь возможность видеть только тип документов "аннулирование счетов-фактур".

В типе документа есть специальная кнопка **Настройки доступа** для настройки списка групп, имеющих доступ к этому типу документа:

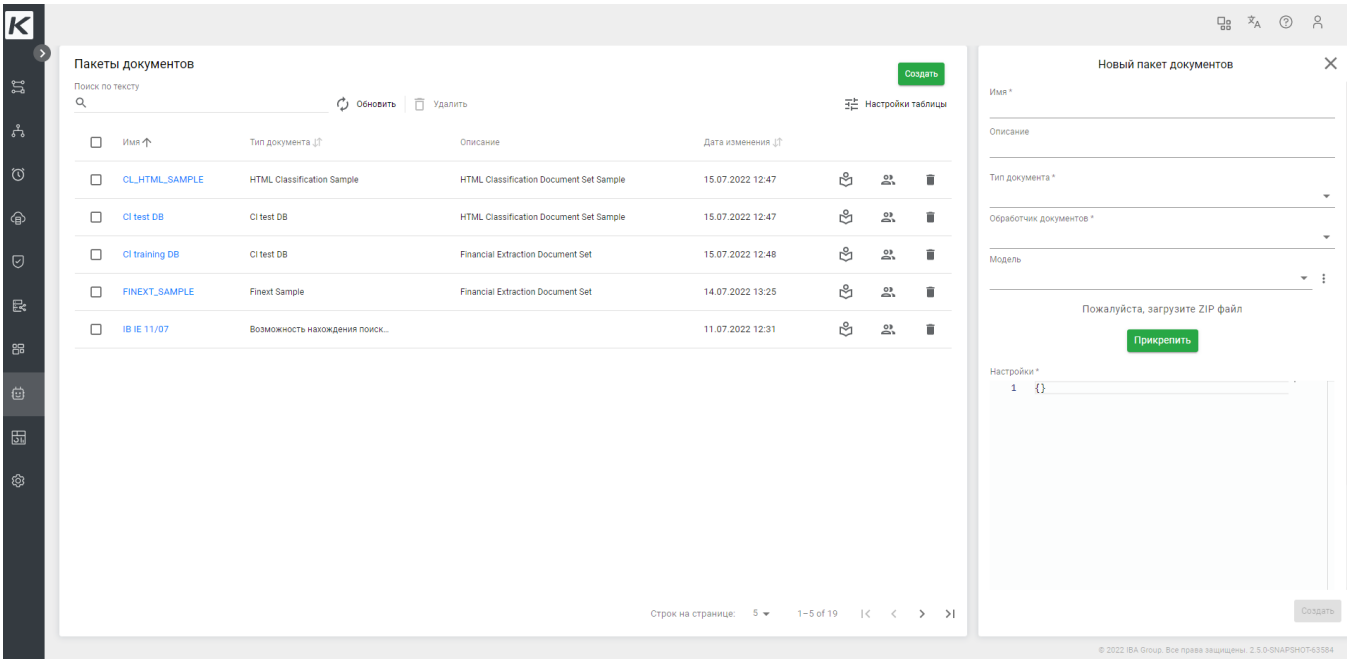


Эта кнопка открывает Управление группами для конкретного типа документов, где можно управлять доступами:


Управление группами
Создать
Поиск по тексту
Обновить
Удалить
Настройки таблицы
Имя
Описание
Administrators
Administrators
Developers
Developers
Monitors
Read-onlysystemmonitoring
Nodes
Nodes
Users
Users
Строк на странице: 5
1–5 of 11
© 2022 IBA Group. Все права защищены. 2.5.0-SNAPSHOT-63584

Шаг 2. Соберите и подготовьте тренировочный набор (Извлечение информации)

Чтобы подготовить тренировочный набор, который используется для обучения новой модели машинного обучения, необходимо создать новый пакет документов. Для этого перейдите в **Машинное обучение - Пакеты документов** и нажмите кнопку **Создать**. См. Создание пакета документов.



Подготовка тренировочного набора состоит из нескольких шагов:

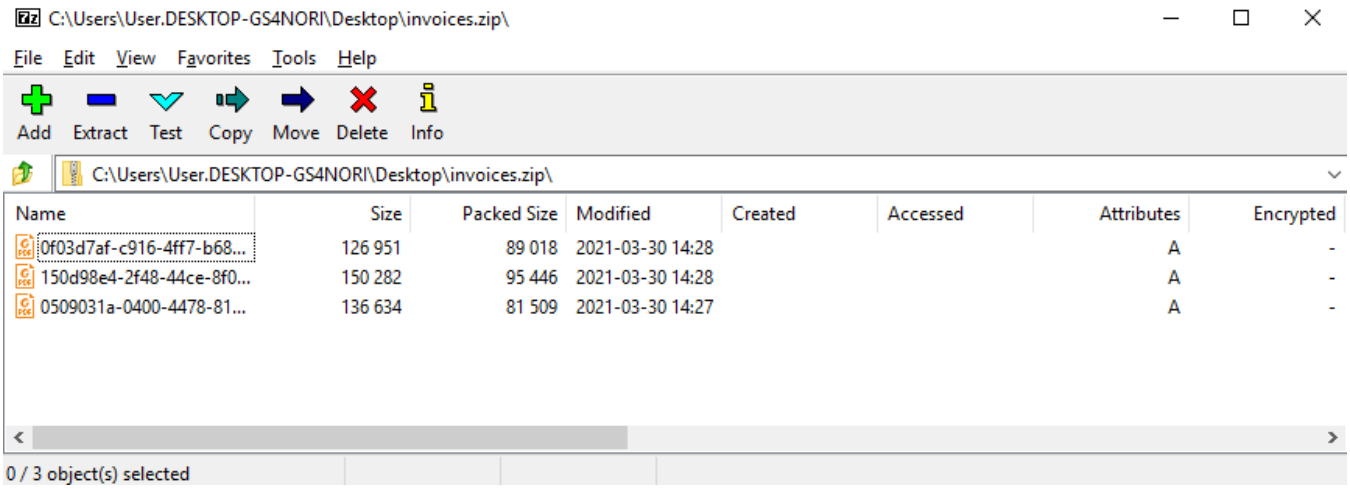
- Прикрепите ZIP-файл
- Выберите тип документов
- Выберите обработчик документов
- Сгенерируйте входные данные документа
- Переместите документы в пользовательскую задачу и разметьте их

Прикрепите ZIP-файл

Главная информация тренировочного пакета - документы. Нужно поместить в ZIP-архив документов, которые будут использоваться для обучения новой модели машинного обучения.

> По умолчанию Канцлер RPA поддерживает обработку документов PDF, изображений и HTML для обучения модели извлечения информации.

Архив должен выглядеть так:



Выберите тип документов

Выберите Тип документов, созданный в предыдущем шаге. Тип документов описывает, как документы должны отображаться в Пользовательской задаче для определения категорий и обучения Модели машинного обучения.

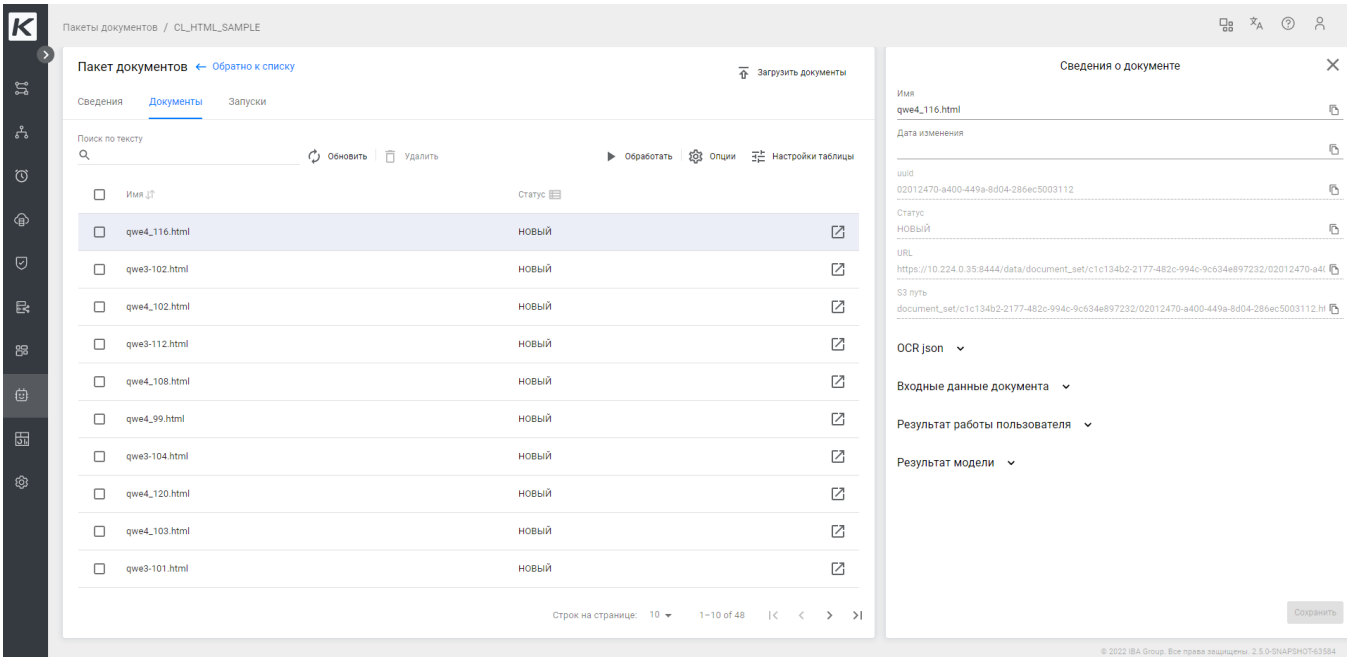
Выберите обработчик документов

Обработчик документов - это специальный процесс, который включает в себя этапы подготовки к отображению документов в Пользовательской задаче.

Для обработки документов в HTML-формате должен быть выбран обработчик "**IE HTML Document Processor**". Для обработки текстовых, PDF документов или изображений Канцлер RPA предлагает "**IE Document Processor**". Этот обработчик включает в себя:

- преобразование PDF-файлов в формат изображений
- улучшения изображений с помощью скриптов ImageMagick
- отправка изображений на OCR
- отправка документов с результатом OCR в Пользовательские задачи
- обработка результатов выполнения Пользовательской задачи и их преобразование в формат вводных данных Машинного обучения

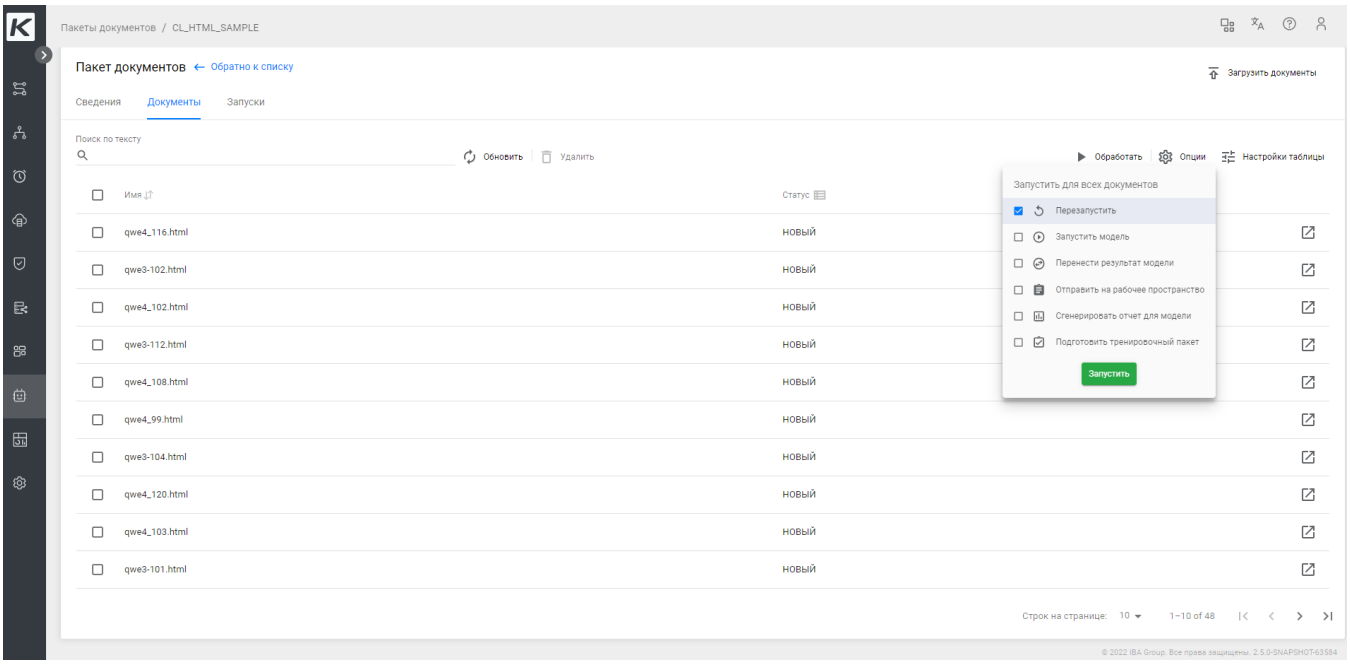
Пакет документов можно создать после предоставления ZIP-архива, выбрав Тип документов и Обработчик документов. Документы из Пакета документов будут загружены в Хранилище файлов. Сведения о каждом документе в пакете документов можно найти, открыв его:



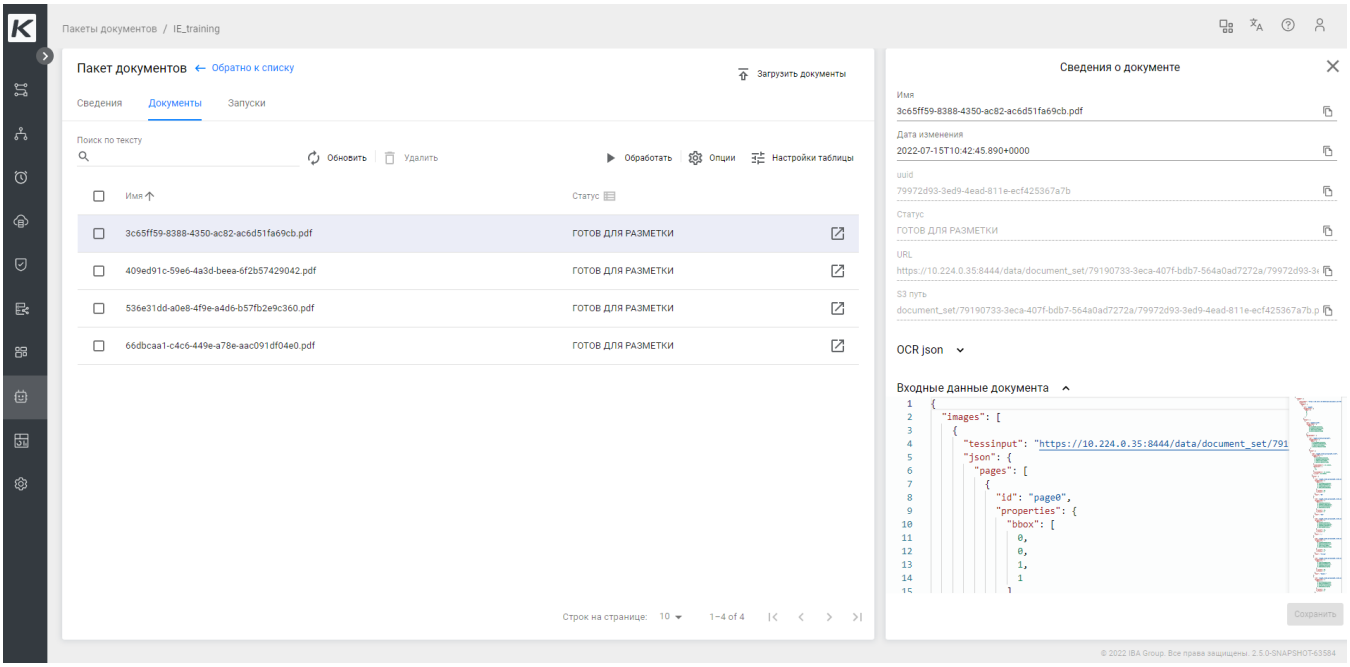
Поскольку этот Пакет документов является новым, **Входные данные документа**, **Результат работы пользователя** и **Результат модели** пусты.

Сгенерируйте входные данные документа

Входные данные документов — это данные, которые поступают в качестве входных данных для Пользовательских задач. Обработчик документов отвечает за подготовку этих входных данных. Для запуска обработчика документов и подготовки Входных данных документов необходимо выбрать «Перезапустить» в меню действий, а затем нажать на кнопку "Запустить":

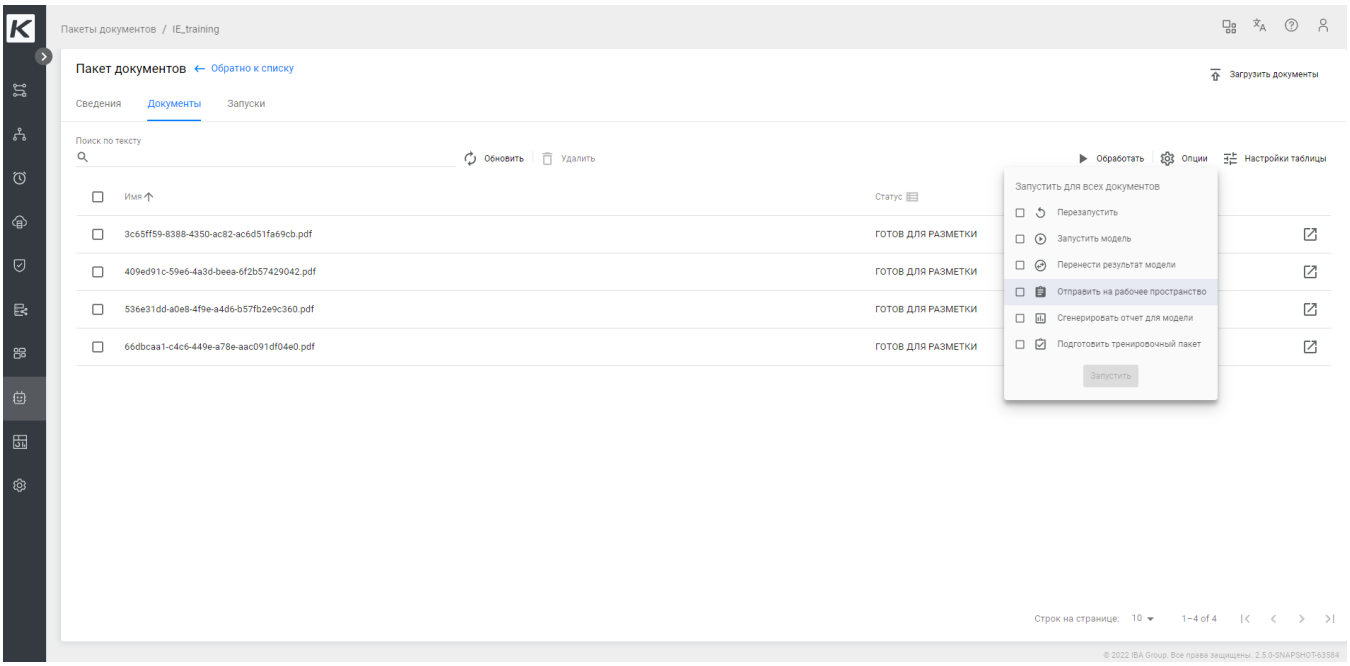


Обработчик документов начнет работать. Отслеживать ход его работы можно в разделе Управление запусками. Необходимо дождаться, пока все документы получат статус "ГОТОВ ДЛЯ РАЗМЕТКИ". В результате "Входные данные документа" будут сгенерированы, входные данные из Пользовательской задачи будут доступны. В этих деталях можно увидеть результат OCR в соответствующем формате:

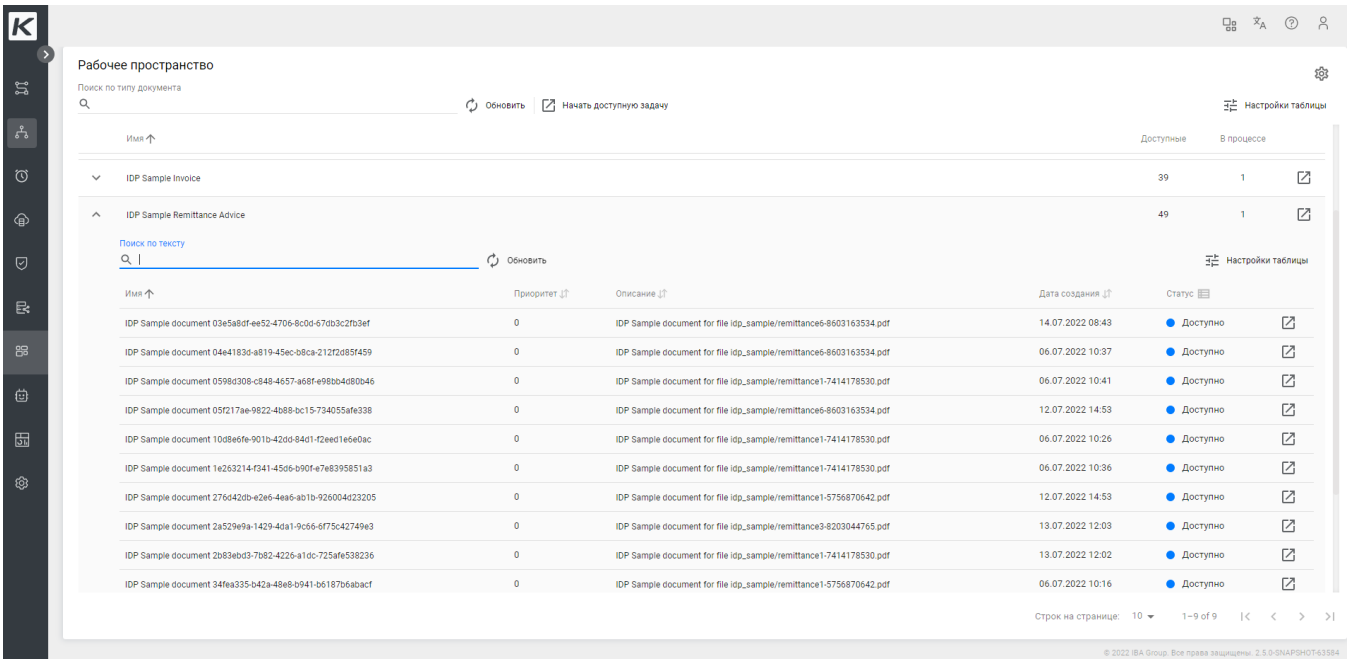


Переместите документы в пользовательскую задачу и разметьте их

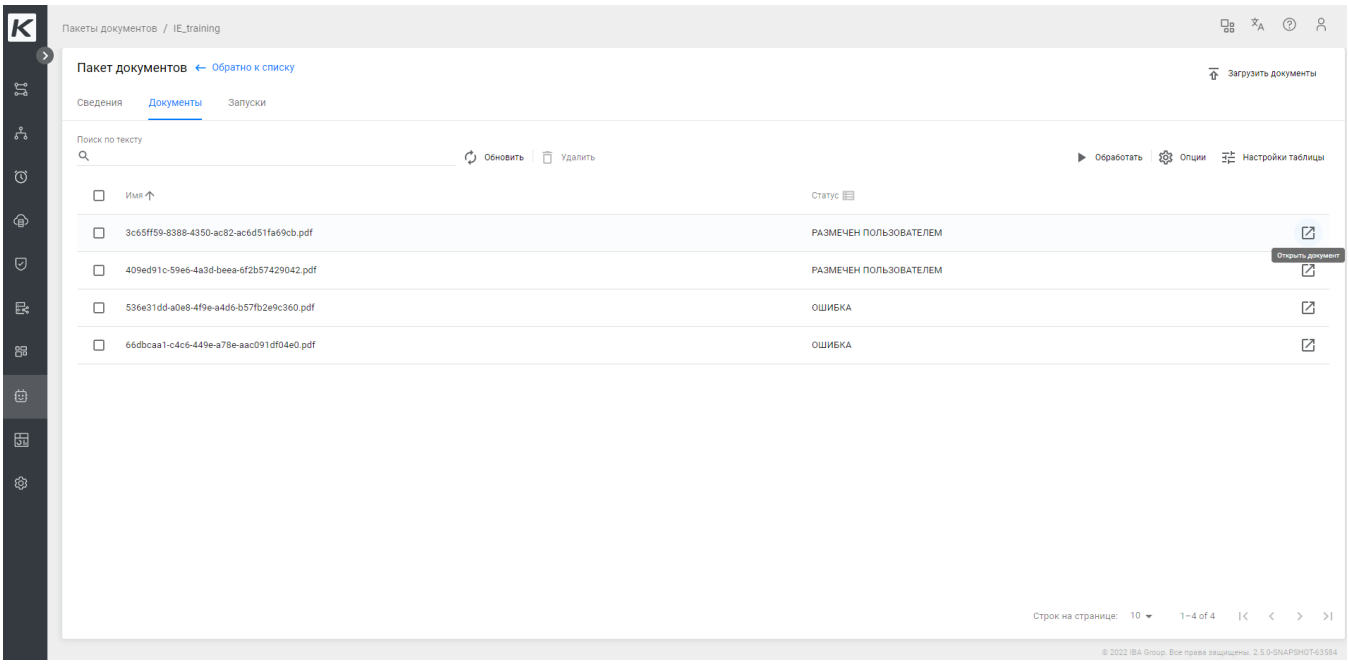
Следующим шагом является разметка документов. Результат разметки будет использоваться алгоритмами машинного обучения для обучения модели. Чтобы пакет документов переместился на рабочее пространство и чтобы работники могли разметить их с помощью Пользовательской задачи, необходимо выбрать действие **Отправить на рабочее пространство**:



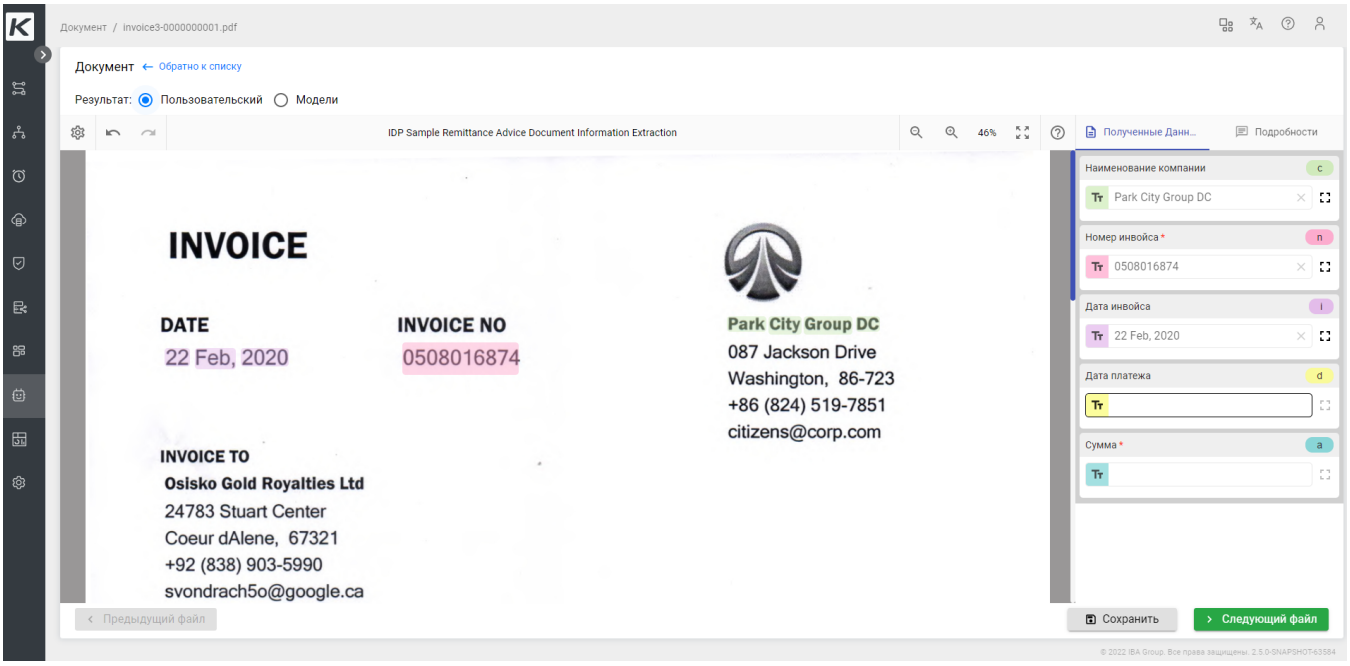
Документы должны отображаться на Рабочем пространстве в соответствующих Типах документов. Чтобы начать их разметку, нажмите кнопку "Начать работу":



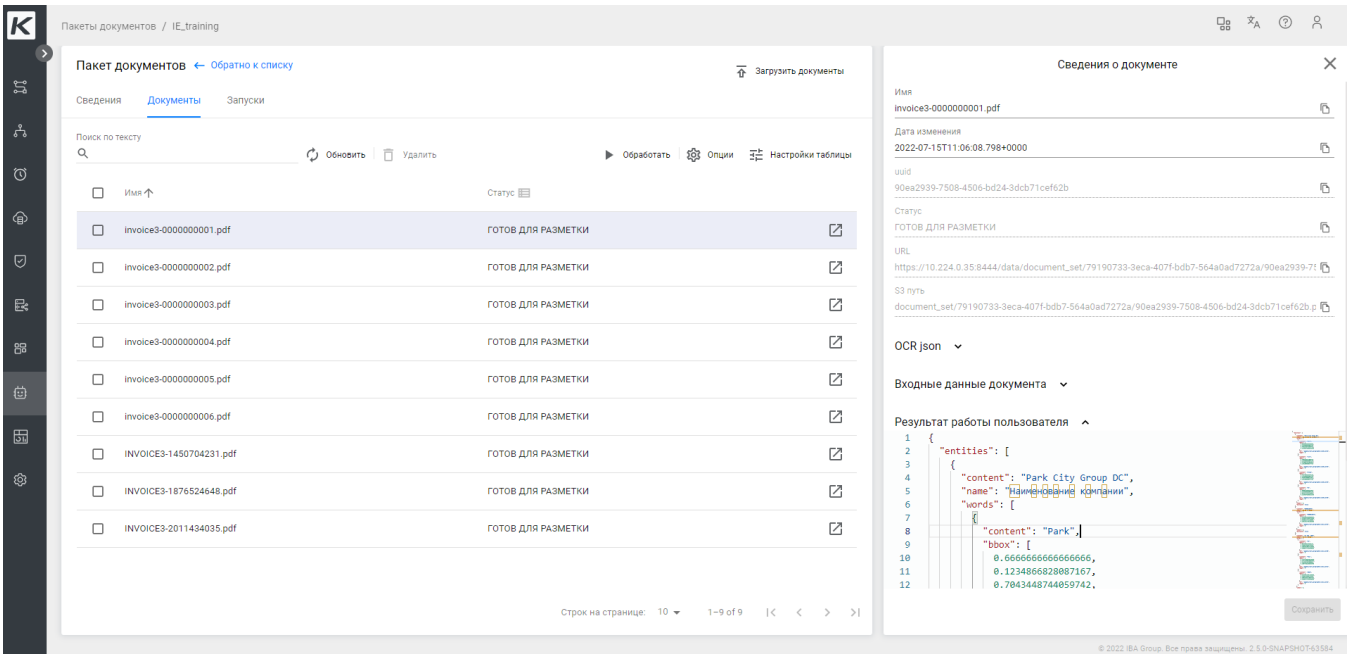
Документы также можно разметить, не отправляя на Рабочее пространство. Нажмите на кнопку "Открыть документ" рядом с меткой статуса в деталях Пакета документов:



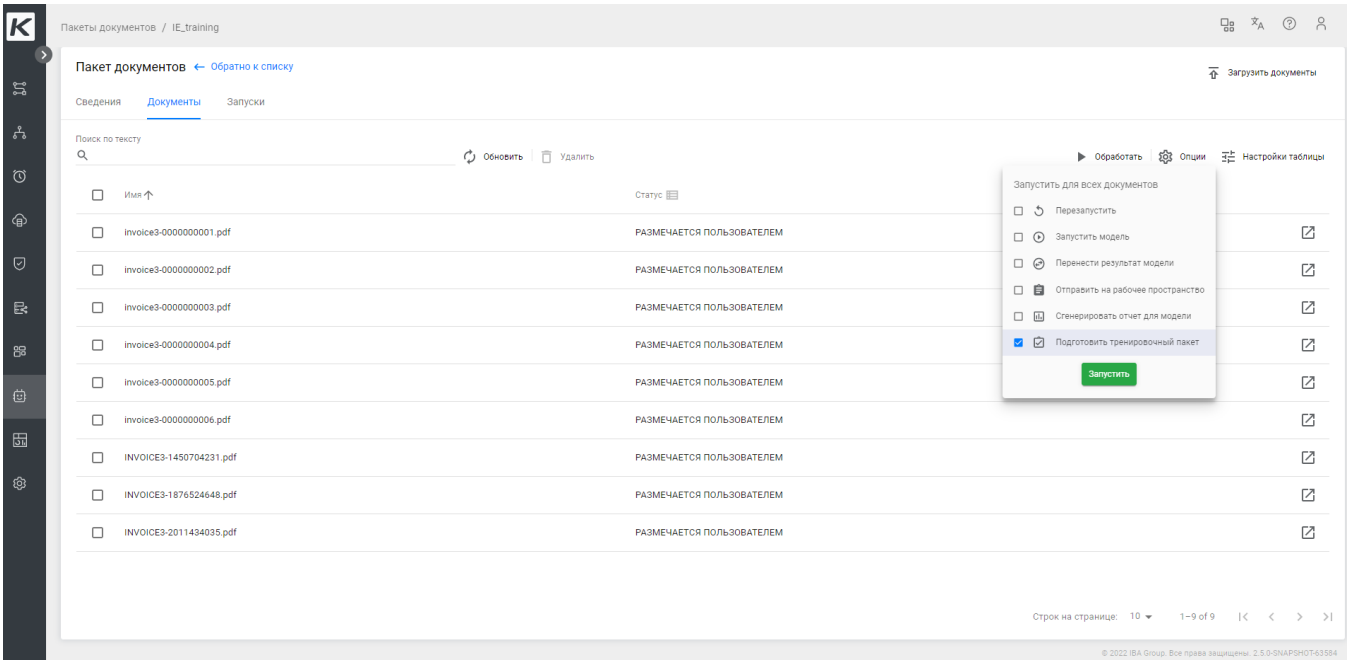
Можно начинать разметку документов:



После того, как документ размечен, для этого конкретного документа генерируется "Результат работы пользователя":



Чтобы завершить подготовку Тренировочного пакета документов, нажмите **Обработать - Подготовить тренировочный набор** и подтвердите свое действие с помощью кнопки **Запустить**:

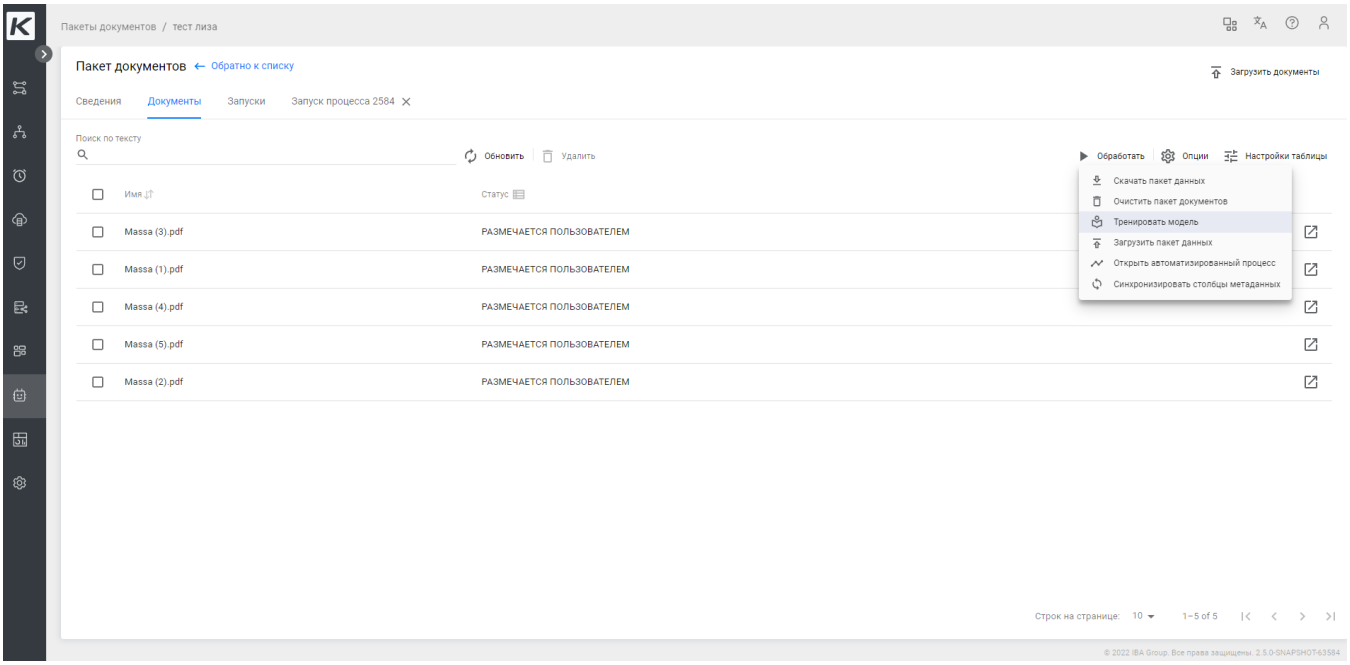


После того, как будут выполнены все вышеперечисленные действия, можно начинать тренировать модель.

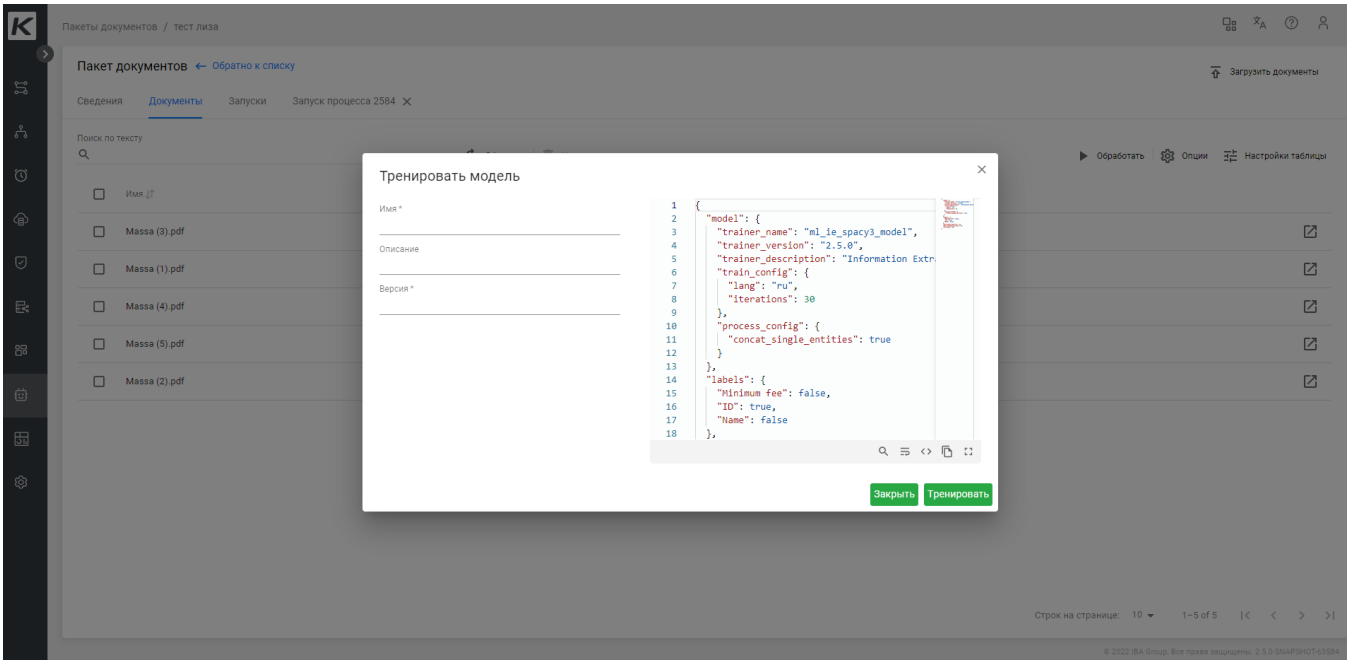
Шаг 3. Тренировка модели машинного обучения (Извлечение информации)

После того тренировочный набор был подготовлен и классифицированы все документы, можно приступать к тренировке Модели машинного обучения.

Для этого зайдите в Пакет документов, и нажмите **Опции - Тренировать модель**.



Появляется следующее диалоговое окно:

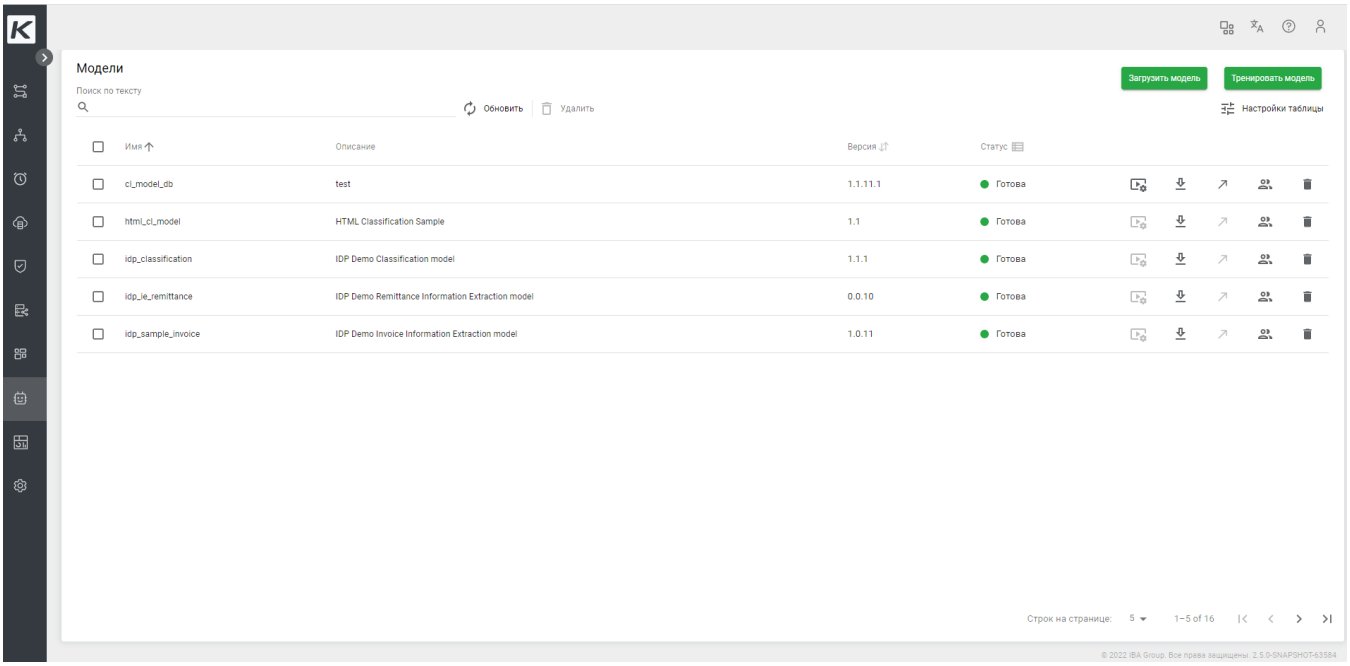


Необходимо указать:

- Имя модели
- Описание модели
- Версию модели

Конфигурация тренировки создается автоматически в формате JSON после выбора Пакета тренировочных документов. Для получения более подробной информации см. Модели извлечения информации

После нажатия кнопки "Тренировать" в диалоговом необходимо жождться окончания тренировки. Модель получит статус "Готова". Для получения более подробной информации см. Машинное обучение - Модели.



Разработка автоматизированного процесса (Извлечение информации)

Шаг 1. Подготовка входных документов (Извлечение информации)

Получите документы из источника данных

Процесс извлечения информации начинается с поиска документов. Обычно, в почтовом ящике, используя критерии поиска, находятся электронные письма. Вложения из этих электронных писем и являются входными данными.

Ниже приведен список примеров источников данных и рекомендации по работе с ними.

Источник данных	Рекомендации
Письма в каком-либо ящике	В зависимости от протокола вы можете получить доступ к почтовому ящику, вам нужно использовать разные библиотеки для его сканирования. Наиболее популярны 2 протокола: • IMAP • Exchange Канцлер RPA предоставляет утилиту почтового клиента, которая поддерживает оба протокола и может использоваться для сканирования и отправки электронных писем. Вы можете сканировать электронные письма, используя необходимые условия поиска, такие как ключевое слово в теме, электронные письма от определенного отправителя, электронные письма в диапазоне дат и т. д.
Файлы из общей сетевой папки	Иногда вам нужно просканировать определенную папку на сетевом диске, чтобы найти какие-то файлы. Доступ к сетевым папкам можно получить двумя способами: 1. Использование клиента протокола Samba для Java (например, https://github.com/AgNO3/jcifs-ng). 2. Вы можете подключить сетевой диск с помощью функции Windows (https://support.microsoft.com/en-us/windows/map-a-network-drive-in-windows-10-29ce55d1-34e3-a7e2-4801-131475f9557d).). После этого вы можете получить доступ к сетевой папке так же, как в локальной файловой системе.
Файлы из файлового хранилища Канцлер RPA	Если операторы бизнес-процессов помещают целевые документов для обработки в файловое хранилище КанцлеRPA, можно сканировать его с помощью существующего Storage Manager.
Файлы с FTP	Также, возможно, для получения входящих документов нужно сканировать FTP-сервер. Для этого используется клиентская библиотека FTP для Java (например, Apache Commons Net). Также обратите внимание, что иногда FTP-серверы имеют простой веб-интерфейс, поэтому возможно получить доступ к файлам с помощью обычных HTTP-запросов Get.

Загрузите документы в файловое хранилище Канцлер RPA

Все документы, полученные на предыдущем шаге, необходимо загрузить в хранилище файлов, потому что далее на шаге OCR и машинного обучения нужно работать с URL-адресами документов, а не с самими документами .

Данные в хранилище файлов возможно загрузить с помощью встроенного Storage Manager.

Пример кода

В данном пошаговом примере реализуется сканирование хранилище Канцлер RPA на наличие новых входящих счетов.

Класс автоматизированного процесса выглядит следующим образом:

InvoiceProcessingSample.java

```
package com.iba.samples.ie;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.samples.ie.task.GetIncomingInvoices;
import lombok.extern.slf4j.Slf4j;

@ApModuleEntry(name = "InvoiceProcessingSample")
@Slf4j
public class InvoiceProcessingSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        return doNotCareOfResult();
    }
}
```

Сама задача, которая сканирует все файлы PDF из определенной папки в хранилище файлов, а затем перемещает их в другую рабочую папку:

GetIncomingInvoices.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.AfterInit;
import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.utils.storage.StorageManager;
import com.iba.samples.ie.ExportConstants;
import com.iba.samples.ie.PropertyConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;

import javax.inject.Inject;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

@ApTaskEntry(name = "Get Documents from Storage")
@Slf4j
@InputToOutput
public class GetIncomingInvoices extends ApTask {

    @Inject
    private StorageManager storageManager;

    @Output(ExportConstants.INVOICE_DOCUMENT_S3_PATHS_LIST)
    private List<String> invoicesS3PathsList = new ArrayList<>();

    private String s3Bucket;
    private String s3FolderToScan;

    private String ocrS3WorkBucket;
    private String ocrS3WorkFolder;

    @AfterInit
    public void init() {
        s3Bucket = getConfigurationService().getConfiguration().get(PropertyConstants.S3_BUCKET);
        s3FolderToScan = getConfigurationService().getConfiguration().get(PropertyConstants.S3_FOLDER_TO_SCAN);
```

```
        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        ocrS3WorkFolder = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_FOLDER);
    }

    @Override
    public void execute() throws Exception {
        List<String> filesPathList = storageManager.listFiles(s3Bucket, s3FolderToScan, ".*\\.pdf");
        log.info("There're {} document(s) found to process on S3 bucket '{}' in folder '{}'", filesPathList.
size(), s3Bucket, s3FolderToScan);
        for (String s3Path : filesPathList) {
            String newS3Path = moveFile(s3Bucket, s3Path, ocrS3WorkBucket, ocrS3WorkFolder);
            invoicesS3PathsList.add(newS3Path);
        }
    }

    private String moveFile(String s3SourceBucket, String s3SourceFilePath, String s3DestBucket, String
s3DestFolder) throws IOException {
        String fileName = FilenameUtils.getName(s3SourceFilePath);
        String resultS3Path = s3DestFolder + "/" + fileName;
        copyFile(s3SourceBucket, s3SourceFilePath, s3DestBucket, resultS3Path);
        storageManager.deleteFile(s3SourceBucket, s3SourceFilePath);
        return resultS3Path;
    }

    private void copyFile(String s3SourceBucket, String s3SourceFilePath, String s3DestBucket, String
s3DestFilePath) throws IOException {
        try(InputStream fileInputStream = storageManager.getFile(s3SourceBucket, s3SourceFilePath)) {
            storageManager.uploadFile(s3DestBucket, s3DestFilePath, fileInputStream);
        }
    }
}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
}
```

ExportConstants.java

```
package com.iba.samples.ie;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices
```

```
ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
```

Шаг 2. Оцифровка документов с помощью OCR (Извлечение информации)

Целью этого шага является преобразование документа в формат HTML, поскольку при выполнении ручных задач и машинного обучения используют этот формат для обработки и отображения содержимого документа.

Если документ уже в HTML формате (например, входящий файл Excel был преобразован в HTML формат или источником документа было тело электронной почты), следует **пропустить этот шаг**.

Канцлер RPA предоставляет специальную Задача OCR, которая может быть использована в коде автоматизированного процесса.

Пример кода

Чтобы получить доступ к документам, полученным на предыдущем шаге, нужно добавить следующий код в автоматизированный процесс:

```
List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);
```

Затем нужно параллельно обработать каждый документ . Для этого используется CompletableFuture API. Перебирая пути к документам, нам нужно подготовить список Completable Futures:

```
List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
if(invoiceDocumentS3PathList.size() > 0) {
    for(String documentS3Path : invoiceDocumentS3PathList) {
        TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
        documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

        CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                execute(documentProcessingInput, PrepareInputForOcrTask.class)
                        .thenCompose(execute(OcrTask.class));

        invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
    }
    CompletableFutures.allOf(invoiceProcessingExecutions).get();
} else {
    log.info("No invoices for processing found");
}
```

Чтобы отправить документ в процесс OCR, нужно подготовить ввод для Задачи OCR в специальном формате со специальным ключом. Для этого существует специальная задача **PrepareInputForOcrTask**, которая отвечает за подготовку объекта **OcrTaskData** с ключом "**ocr.task.data**" в выходных переменных.

Весь пример выглядит следующим образом:

Класс автоматизированного процесса:

InvoiceProcessingSample.java

```
package com.iba.samples.ie;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.ie.task.GetIncomingInvoices;
import com.iba.samples.ie.task.PrepareInputForOcrTask;
import lombok.extern.slf4j.Slf4j;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceProcessingSample")
@Slf4j
public class InvoiceProcessingSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }
}
```

Новый класс, который подготавливает ввод для задачи OCR:

PrepareInputForOcrTask.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrFormats;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.ie.ExportConstants;
import com.iba.samples.ie.PropertyConstants;
import lombok.extern.slf4j.Slf4j;

import java.util.*;

@ApTaskEntry(name = "Prepare Input For OCR Task")
@Slf4j
@InputToOutput
public class PrepareInputForOcrTask extends ApTask {
    private static final String OCR_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY = "documents_bucket";
    private static final String OCR_TASK_CONFIGURATION_TESSERACT_OPTIONS_KEY = "tesseractOptions";
    private static final String OCR_TASK_CONFIGURATION_IMAGE_MAGICK_OPTIONS_KEY = "imageMagickOptions";
    private static final List<String> OCR_OUTPUT_FORMATS = Arrays.asList(OcrFormats.TEXT, OcrFormats.HOCR,
OcrFormats.JSON, OcrFormats.IMAGE);

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskData;
```

```
    private String ocrS3WorkBucket;
    private String ocrTesseractOptions;
    private String ocrImageMagickOptions;

    private Map<String, Object> ocrConfiguration = new HashMap<>();

    @AfterInit
    public void init() {
        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        ocrTesseractOptions = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_TESSERACT_OPTIONS);
        ocrImageMagickOptions = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_IMAGE_MAGICK_OPTIONS);

        ocrConfiguration.put(OCR_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY, ocrS3WorkBucket);
        ocrConfiguration.put(OCR_TASK_CONFIGURATION_TESSERACT_OPTIONS_KEY, MlTaskUtils.readObjectFromString
(ocrTesseractOptions, ArrayList.class));
        ocrConfiguration.put(OCR_TASK_CONFIGURATION_IMAGE_MAGICK_OPTIONS_KEY, MlTaskUtils.readObjectFromString
(ocrImageMagickOptions, ArrayList.class));
    }

    @Override
    public void execute() throws Exception {
        log.info("Prepare input for document in '{}' with ocr task with configuration: {}", documentS3Path,
ocrConfiguration);

        ocrTaskData = new OcrTaskData();
        ocrTaskData.setDocumentLocation(documentS3Path);
        ocrTaskData.getFormats().addAll(OCR_OUTPUT_FORMATS);
        ocrTaskData.setConfiguration(ocrConfiguration);
    }
}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";
}
```

ExportConstants.java

```
package com.iba.samples.ie;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]
```

Шаг 3. Применение и запуск модели машинного обучения (Извлечение информации)

Обученная модель используется в автоматизированном процессе. Необходимо знать **название** модели и **версию**.

Канцлер RPA предоставляет класс MlTask для вызова модели извлечения информации. Необходимо подготовить входные данные для MlTask. Это должен быть объект **MlTaskData** с ключом "**ml.task.data**" в выходных переменных.

Пример кода

Чтобы подготовить ввод для MlTask, необходимо создать класс **PrepareInputForMlTask** до вызова **MlTask**.

Таким образом, класс автоматизированного процесса выглядит следующий образом:

InvoiceProcessingSample.java

```
package com.iba.samples.ie;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.ie.task.GetIncomingInvoices;
import com.iba.samples.ie.task.PrepareInputForMlTask;
import com.iba.samples.ie.task.PrepareInputForOcrTask;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceProcessingSample")
@Slf4j
public class InvoiceProcessingSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
```

```
    }

}
```

Код **PrepareInputForMlTask:**

PrepareInputForMlTask.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.samples.ie.ExportConstants;
import com.iba.samples.ie.PropertyConstants;
import lombok.extern.slf4j.Slf4j;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@ApTaskEntry(name = "Prepare input for ML Task")
@Slf4j
@InputToOutput
public class PrepareInputForMlTask extends ApTask {
    private static final String ML_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY = "documents_bucket";

    @Input(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskOutput;

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(MlTask.ML_TASK_DATA_KEY)
    private MlTaskData mlTaskData;

    private String modelName;
    private String modelVersion;

    private String documentId;

    private Map<String, Object> mlConfiguration = new HashMap<>();

    @AfterInit
    public void init() {
        modelName = getConfigurationService().get(PropertyConstants.IE_MODEL_NAME);
        modelVersion = getConfigurationService().get(PropertyConstants.IE_MODEL_VERSION);

        documentId = UUID.randomUUID().toString();

        String ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        mlConfiguration.put(ML_TASK_CONFIGURATION_WORK_S3_BUCKET_KEY, ocrS3WorkBucket);
    }

    @Override
    public void execute() {
        log.info("Creating ML task input for document located at '{}' with assigned id '{}'", documentS3Path,
documentId);

        mlTaskData = new MlTaskData();
        mlTaskData.setModelName(modelName);
        mlTaskData.setModelVersion(modelVersion);
        mlTaskData.setConfiguration(mlConfiguration);
                MlTaskUtils.prepareIeMlTask(documentId, mlTaskData, ocrTaskOutput);
```

```
    }

}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";

    String IE_MODEL_NAME = "ie_model.name";
    String IE_MODEL_VERSION = "ie_model.version";
}
```

ExportConstants.java

```
package com.iba.samples.ie;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

ie_model.name=ordinary52withfixes
ie_model.version=0.0.2
```

Шаг 4. Чтение выходных данных машинного обучения и маршрутизация документов в пользовательскую задачу (Извлечение информации)

Для считывания выходных данных Машинного обучения необходимо получить значение по ключу "**ml.task.data**" из объекта **Task Output**. Оно содержит объект **MlTaskData**, который содержит список объектов **Document**, в котором вы можете просматривать страницы каждого документа для анализа извлеченных объектов.

Чаще всего пользователь отправляет один документ на обработку, поэтому здесь пример того, как получить извлеченное поле «**Invoice Number**»:

```
String extractedInvoiceNumber = null;
MlTaskData mlTaskDataOutput = previousTaskStepOutput.get(MlTask.ML_TASK_DATA_KEY, MlTaskData.class);
Document mlTaskProcessedDocument = mlTaskDataOutput.getDocuments().get(0);
for(Page page : mlTaskProcessedDocument.getPages()) {
        for(Entity entity : page.getEntities()) {
                if(entity.getName().equals("Invoice Number")) {
                        extractedInvoiceNumber = entity.getContent();
                }
        }
}
```

Обычно, если какая-то информация не была извлечена с помощью машинного обучения, документ отправляют на Пользовательскую задачу. Для подробной информации см. Создание пользовательской задачи.

Чтобы отправить документ на Пользовательскую задачу, вам нужно подготовить специальный ввод для встроенного класса **HumanTask**. Входные данные должны содержать объект **HumanTaskData** с ключом «**human.task.data**» во входных данных.

Пример кода

В класс автоматизированного процесса нужно добавить дополнительную функцию, которая выполняется асинхронно и считывает вывод с шага машинного обучения. Если поле «**Invoice Number**» не извлекается автоматически, асинхронно выполняется Пользовательская задача. Если все извлечено — возвращается завершенный Completable Future с выводом из предыдущего шага:

InvoiceProcessingSample.java

```
package com.iba.samples.ie;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.ie.task.*;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceProcessingSample")
@Slf4j
public class InvoiceProcessingSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
```

```
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class))
                                .thenCompose(previousTaskStepOutput -> {
                                    MlTaskData mlTaskDataOutput = previousTaskStepOutput.get(MlTask.
ML_TASK_DATA_KEY, MlTaskData.class);
                                                                        Map mlTaskResultMap = MlTaskUtils.
createIeHtOutputMap(mlTaskDataOutput);
                                                                        List<Map<String, Object>>
extractedFields = (List<Map<String, Object>>) mlTaskResultMap.get("entities");

                                                                        boolean invoiceNumberExtracted =
extractedFields.stream().anyMatch(extractedField -> extractedField.get("name").toString().equals
(ExportConstants.IE.INVOICE_NUMBER));
                                                                        if(!invoiceNumberExtracted) {
                                                                                return execute
(previousTaskStepOutput, PrepareInputForHumanTask.class)
                                                                                                        .thenCompose
(execute(HumanTask.class));
                                                                        } else {
                                                                                return CompletableFuture.
completedFuture(previousTaskStepOutput); //do nothing
                                                                        }
                                });

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }
}
```

Для отправки документа на Пользовательскую задачу нужно добавить дополнительный шаг, отвечающий за подготовку входных данных для него. Кроме того, должна выводиться специальная логическая переменная, которая помогает на следующих шагах определить, была ли запись обработана человеком или нет.

PrepareInputForHumanTask.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.*;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTaskData;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.engine.task.ocr.OcrTaskData;
import com.iba.kanclerrpa.utils.MlTaskUtils;
import com.iba.kanclerrpa.utils.storage.StorageManager;
import com.iba.samples.ie.ExportConstants;
import com.iba.samples.ie.PropertyConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;
```

```
import javax.inject.Inject;
import java.util.Map;
import java.util.HashMap;

@ApTaskEntry(name = "Prepare Human Task")
@Slf4j
@InputToOutput
public class PrepareInputForHumanTask extends ApTask  {

    @Inject
    private StorageManager storageManager;

    @Input(OcrTask.OCR_TASK_DATA_KEY)
    private OcrTaskData ocrTaskOutput;

    @Input(MlTask.ML_TASK_DATA_KEY)
    private MlTaskData mlTaskOutput;

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String documentS3Path;

    @Output(HumanTask.HUMAN_TASK_DATA_KEY)
    private HumanTaskData humanTaskData;

    @Output(ExportConstants.DOCUMENT_PROCESSED_BY_HUMAN)
    private boolean documentProcessedByHuman = true;

    private String ocrS3WorkBucket;
    private String documentType;

    @AfterInit
    public void init() {
        ocrS3WorkBucket = getConfigurationService().getConfiguration().get(PropertyConstants.
OCR_S3_WORK_BUCKET);
        documentType = getConfigurationService().getConfiguration().get(PropertyConstants.DOCUMENT_TYPE);
    }

    @Override
    public void execute() throws Exception {
        String documentName = FilenameUtils.getName(documentS3Path);
        log.info("Creating human task for document {} with document type {}", documentName, documentType);

        humanTaskData = new HumanTaskData();
                 humanTaskData.setInputJson(MlTaskUtils.createIeHtInputMap(ocrTaskOutput, storageManager,
ocrS3WorkBucket));
                humanTaskData.setOutputJson(new HashMap(MlTaskUtils.createIeHtOutputMap(mlTaskOutput)));

        humanTaskData.setName(documentName);
        humanTaskData.setDocumentType(documentType);
        humanTaskData.setDescription("Invoice Document: " + documentName);

        log.info("Sending task into workspace {} ", humanTaskData);
    }

}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
```

```
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";

    String IE_MODEL_NAME = "ie_model.name";
    String IE_MODEL_VERSION = "ie_model.version";

    String DOCUMENT_TYPE = "document_type";
}
```

ExportConstants.java

```
package com.iba.samples.ie;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
    String DOCUMENT_PROCESSED_BY_HUMAN = "DOCUMENT_PROCESSED_BY_HUMAN";

    interface IE {
        String INVOICE_NUMBER = "INVOICE_NUMBER";
    }
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

ie_model.name=ordinary52withfixes
ie_model.version=0.0.2

document_type=Products invoice
```

Шаг 5. Результат процесса извлечения (Извлечение информации)

После того, как все данные извлечены с помощью машинного обучения или с помощью пользователя, обычно они согласовываются с данными из бизнес-приложения с помощью RPA или вставляются в какое-либо приложение.

В таких задачах нужно получить извлеченные данные независимо от того, пришли ли данные из задачи машинного обучения или пользовательской задачи. В приведенном ниже примере можно увидеть, как проверяется, откуда поступили входные данные, и как провести простую валидацию.

Пример кода

В классе автоматизированного процесса мы добавили 2 дополнительных шага: **ValidateInvoiceData**, который должен отвечать за внедрение данных, и **PrepareResult**, который должен отвечать за отправку отчета о выполнении:

InvoiceProcessingSample.java

```
package com.iba.samples.ie;

import com.iba.kanclerrpa.engine.annotation.ApModuleEntry;
import com.iba.kanclerrpa.engine.apflow.ApModule;
import com.iba.kanclerrpa.engine.apflow.TaskInput;
import com.iba.kanclerrpa.engine.apflow.TaskOutput;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ml.MlTask;
import com.iba.kanclerrpa.engine.task.ml.MlTaskData;
import com.iba.kanclerrpa.engine.task.ocr.OcrTask;
import com.iba.kanclerrpa.utils.CompletableFutures;
import com.iba.samples.ie.task.*;
import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;

@ApModuleEntry(name = "InvoiceProcessingSample")
@Slf4j
public class InvoiceProcessingSample extends ApModule {

    public TaskOutput run() throws Exception {
        TaskOutput incomingInvoicesResult = execute(getInput(), GetIncomingInvoices.class).get();
        List<String> invoiceDocumentS3PathList = incomingInvoicesResult.get(ExportConstants.
INVOICE_DOCUMENT_S3_PATHS_LIST, List.class);

        List<CompletableFuture<TaskOutput>> invoiceProcessingExecutions = new ArrayList<>();
        if(invoiceDocumentS3PathList.size() > 0) {
            for(String documentS3Path : invoiceDocumentS3PathList) {
                TaskInput documentProcessingInput = new TaskInput(incomingInvoicesResult);
                documentProcessingInput.set(ExportConstants.INVOICE_DOCUMENT_S3_PATH, documentS3Path);

                CompletableFuture<TaskOutput> singleInvoiceProcessingExecution =
                        execute(documentProcessingInput, PrepareInputForOcrTask.class)
                                .thenCompose(execute(OcrTask.class))
                                .thenCompose(execute(PrepareInputForMlTask.class))
                                .thenCompose(execute(MlTask.class))
                                .thenCompose(previousTaskStepOutput -> {
                                                                        MlTaskData mlTaskDataOutput =
previousTaskStepOutput.get(MlTask.ML_TASK_DATA_KEY, MlTaskData.class);
                                                                        Map mlTaskResultMap = MlTaskUtils.
createIeHtOutputMap(mlTaskDataOutput);
                                                                        List<Map<String, Object>>
extractedFields = (List<Map<String, Object>>) mlTaskResultMap.get("entities");

                                                                        boolean invoiceNumberExtracted =
extractedFields.stream().anyMatch(extractedField -> extractedField.get("name").toString().equals
```

```
(ExportConstants.IE.INVOICE_NUMBER));
                                                                        if(!invoiceNumberExtracted) {
                                                                                return execute
(previousTaskStepOutput, PrepareInputForHumanTask.class)
                                                                                                        .thenCompose
(execute(HumanTask.class));
                                                                        } else {
                                                                                return CompletableFuture.
completedFuture(previousTaskStepOutput); //do nothing
                                                                        }
                                })
                                .thenCompose(execute(ValidateInvoiceData.class))
                                .thenCompose(execute(PrepareResult.class));

                invoiceProcessingExecutions.add(singleInvoiceProcessingExecution);
            }
            CompletableFutures.allOf(invoiceProcessingExecutions).get();
        } else {
            log.info("No invoices for processing found");
        }

        return doNotCareOfResult();
    }
}
```

ValidateInvoiceData проверяет, соответствует ли извлеченное поле номера счета требованиям по длине:

ValidateInvoiceData.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.InputToOutput;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTask;
import com.iba.kanclerrpa.engine.task.ht.HumanTaskData;
import com.iba.kanclerrpa.engine.task.ml.*;
import com.iba.samples.ie.ExportConstants;
import lombok.extern.slf4j.Slf4j;

import java.util.List;
import java.util.Map;

@ApTaskEntry(name = "Validate invoice data")
@Slf4j
@InputToOutput
public class ValidateInvoiceData extends ApTask {
         private static final String IE_OUTPUT_ENTITIES_KEY = "entities";
        private static final String IE_OUTPUT_ENTITY_NAME_KEY = "name";
        private static final String IE_TASK_OUTPUT_ENTITY_VALUE_KEY = "content";

    @Input(value = ExportConstants.DOCUMENT_PROCESSED_BY_HUMAN, required = false)
    private boolean documentProcessedByHuman;

    @Input(value = HumanTask.HUMAN_TASK_DATA_KEY, required = false)
    private HumanTaskData humanTaskOutput;

    @Input(value = MlTask.ML_TASK_DATA_KEY, required = false)
    private MlTaskData mlTaskOutput;

    @Output(ExportConstants.INVOICE_VALID)
    private String extractedInvoiceNumber;

    @Output(ExportConstants.INVOICE_VALID)
    private boolean invoiceValid;

    @Override
```

```
    public void execute() throws Exception {
        String invoiceNumber;
        if(documentProcessedByHuman) {
            invoiceNumber = extractInvoiceNumberFromHumanOutput(humanTaskOutput);
        } else {
            invoiceNumber = extractInvoiceNumberFromMlOutput(mlTaskOutput);
        }
        invoiceValid = validateInvoiceNumber(invoiceNumber);
        log.info("Validation result is {} for invoice number {}", invoiceValid, invoiceNumber);
    }

         private String extractInvoiceNumberFromHumanOutput(HumanTaskData humanTaskOutput) {
                List<Map<String, Object>> extractedFields = (List<Map<String, Object>>) humanTaskOutput.
getOutputJson().get(IE_OUTPUT_ENTITIES_KEY);
                return extractFieldFromIEOutput(extractedFields, ExportConstants.IE.INVOICE_NUMBER);
        }

        private String extractInvoiceNumberFromMlOutput(MlTaskData mlTaskOutput) {
                Map mlTaskResultMap = MlTaskUtils.createIeHtOutputMap(mlTaskOutput);
                List<Map<String, Object>> extractedFields = (List<Map<String, Object>>) mlTaskResultMap.get
("entities");
                return extractFieldFromIEOutput(extractedFields, ExportConstants.IE.INVOICE_NUMBER);
        }

        private String extractFieldFromIEOutput(List<Map<String, Object>> extractedFields, String
targetFieldName) {
                for(Map<String, Object> extractedField : extractedFields) {
                        if(extractedField.get(IE_OUTPUT_ENTITY_NAME_KEY).toString().equals(targetFieldName)) {
                                return extractedField.get(IE_TASK_OUTPUT_ENTITY_VALUE_KEY).toString();
                        }
                }
                return null;
        }

    private boolean validateInvoiceNumber(String invoiceNumber) {
        return invoiceNumber != null && invoiceNumber.length() > 8;
    }
}
```

PrepareResult генерирует вывод с окончательным текстом, действителен ли номер инвойса или нет:

PrepareResult.java

```
package com.iba.samples.ie.task;

import com.iba.kanclerrpa.engine.annotation.ApTaskEntry;
import com.iba.kanclerrpa.engine.annotation.Input;
import com.iba.kanclerrpa.engine.annotation.Output;
import com.iba.kanclerrpa.engine.apflow.ApTask;
import com.iba.samples.ie.ExportConstants;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;

@ApTaskEntry(name = "Prepare Result")
@Slf4j
public class PrepareResult extends ApTask {

    @Input(ExportConstants.INVOICE_DOCUMENT_S3_PATH)
    private String invoiceS3Path;

    @Input(ExportConstants.EXTRACTED_INVOICE_NUMBER)
    private String extractedInvoiceNumber;

    @Input(ExportConstants.INVOICE_VALID)
    private boolean invoiceValid;

    @Output(ExportConstants.PROCESSING_RESULT)
    private String processingResult;
```

```
    @Override
    public void execute() throws Exception {
        String validationResult = invoiceValid ? "valid" : "not valid";
        processingResult = "Invoice document '" + FilenameUtils.getName(invoiceS3Path) + "' with extracted
invoice number '" + extractedInvoiceNumber + "' is " + validationResult;
    }
}
```

PropertyConstants.java

```
package com.iba.samples.ie;

public interface PropertyConstants {
    String S3_BUCKET = "s3.bucket";
    String S3_FOLDER_TO_SCAN = "s3.folder_to_scan";

    String OCR_S3_WORK_FOLDER = "ocr.s3_work_folder";
    String OCR_S3_WORK_BUCKET = "ocr.s3_work_bucket";
    String OCR_TESSERACT_OPTIONS = "ocr.tesseract_options";
    String OCR_IMAGE_MAGICK_OPTIONS = "ocr.image_magick_options";

    String IE_MODEL_NAME = "ie_model.name";
    String IE_MODEL_VERSION = "ie_model.version";

    String DOCUMENT_TYPE = "document_type";
}
```

ExportConstants.java

```
package com.iba.samples.ie;

public interface ExportConstants {
    String INVOICE_DOCUMENT_S3_PATHS_LIST = "INVOICE_DOCUMENT_S3_PATHS_LIST";
    String INVOICE_DOCUMENT_S3_PATH = "INVOICE_DOCUMENT_S3_PATH";
    String DOCUMENT_PROCESSED_BY_HUMAN = "DOCUMENT_PROCESSED_BY_HUMAN";
    String INVOICE_VALID = "INVOICE_VALID";
    String EXTRACTED_INVOICE_NUMBER = "EXTRACTED_INVOICE_NUMBER";
    String PROCESSING_RESULT = "PROCESSING_RESULT";

    interface IE {
        String INVOICE_NUMBER = "INVOICE_NUMBER";
        String DEBIT_NOTE_ID = "DebitNoteId";
    }
}
```

apm_run.properties

```
# KanclerRPA Client Configuration File
s3.bucket=ie-sample
s3.folder_to_scan=incoming_invoices

ocr.s3_work_bucket=data
ocr.s3_work_folder=incoming_invoices_work_folder
ocr.tesseract_options=["-l", "eng","--psm","3","--oem","3","--dpi", "800"]
ocr.image_magick_options=["-resample", "450", "-density", "350", "-quality", "100", "-background" , "white" , "-
alpha" , "flatten"]

ie_model.name=ordinary52withfixes
ie_model.version=0.0.2
```

```
document_type=Products invoice
```

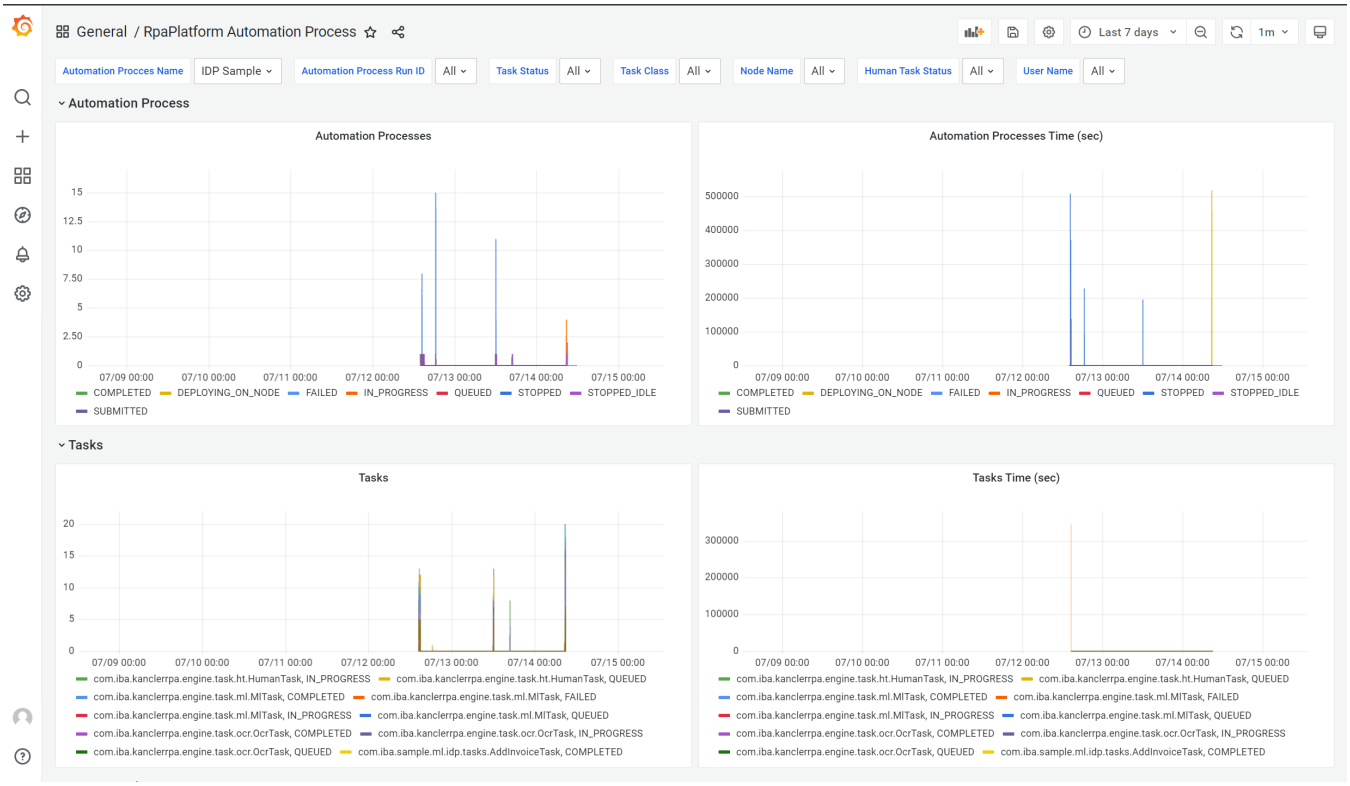
Метрики платформы

Обзор

Начиная с версии 2.3.0 платформа сохраняет различные метрики в InfluxDB. Установочный пакет содержит пользовательский интерфейс Grafana и предопределенные панели мониторинга для представления метрик.

https://<ControlServerAddress>/grafana/

Некоторые панели мониторинга предназначены для демонстрации в пользовательском интерфейсе сервера управления, а некоторые из них вы можете использовать в качестве шаблона для своих собственных панелей мониторинга.



Ниже вы можете найти справочное руководство по метрикам.

Метрики запуска задач

rpaplatform_task - counter

Срабатывает каждый раз, когда запущенный автоматизированный процесс меняет свой статус. Значение всегда равно 1.

rpaplatform_task_time - counterrpaplatform

Срабатывает каждый раз, когда запущенный автоматизированный процес меняет свой статус. Значение равно миллисекундам с момента последнего изменения статуса.

Метки метрик запуска задач

Метка	Описание
ap_id	ID автоматизированного процесса
ap_uuid	UUID автоматизированного процесса
ap_name	Название автоматизированного процесса
run_id	ID запуска автоматизированного процесса
run_uuid	UUID запуска автоматизированного процесса
task_uuid	Необязательная метка. UUID задачи, если запись представляет собой запуск задачи.
task_class	Необязательная метка. Имя класса задачи, если запись представляет собой запуск задачи.
node_id	Необязательная метка. ID узла, если узел уже назначен.
node_name	Необязательная метка. Имя узла, если узел уже назначен.
task_status	Статус запуска автоматизированного процесса: ОТПРАВЛЕН (SUBMITTED), В ОЧЕРЕДИ (QUEUED), ЗАГРУЖАЕТСЯ (DEPLOYING_ON_NODE), ВЫПОЛНЯЕТСЯ (IN_PROGRESS), ОСТАНАВЛИВАЕТСЯ (STOPPING), ОСТАНОВЛЕН (STOPPED), НЕ ВЫПОЛНЕН (FAILED), ВЫПОЛНЕН (COMPLETED), В РЕЖИМЕ ОЖИДАНИЯ (STOPPED_IDLE)
task_type	Задача или модуль. Запуск задачи или автоматизированного процесса

Метрики пользовательских задач

rpaplatform_htask - counter

Срабатывает каждый раз, когда пользовательская задача меняет свой статус. Значение всегда равно 1.

rpaplatform_htask_time - counter

Срабатывает каждый раз, когда пользовательская задача меняет свой статус. Значение равно миллисекундам прошедшим с момента последнего изменения статуса.

Метки метрик пользовательских задач

Метка	Описание

ap_id	ID автоматизированного процесса
ap_uuid	UUID автоматизированного процесса
ap_name	Название автоматизированного процесса
run_id	ID запуска автоматизированного процесса
run_uuid	UUID запуска автоматизированного процесса
task_uuid	Необязательная метка. UUID задачи, если запись представляет собой запуск задачи.
task_class	Необязательная метка. Имя класса задачи, если запись представляет собой запуск задачи.
node_id	Необязательная метка. ID узла, если узел уже назначен.
node_name	Необязательная метка. Имя узла, если узел уже назначен.
htask_status	Статус пользовательской задачи: ДОСТУПНО (AVAILABLE), В ПРОЦЕССЕ (IN_PROGRESS), ЗАВЕРШЕНО (COMPLETED), ОСТАНОВЛЕНА (STOPPED)
user_name	Имя пользователя, выполняющего действия в пользовательской задаче.
user_id	ID пользователя, выполняющего действия в пользовательской задаче.

Метрики узлов

rpaplatform_node_cpu - gauge

Срабатывает каждую минуту со значением, равным CPU узла.

rpaplatform_node_cpu_usage - gauge

Запускается каждую минуту со значением, равным загрузке CPU узла.

rpaplatform_node_memory_free - gauge

Запускается каждую минуту со значением, равным свободной памяти узла.

rpaplatform_node_memory_total - gauge

Запускается каждую минуту со значением, равным общей памяти узла.

rpaplatform_node_free_ui_executors - gauge

Запускается каждую минуту со значением, равным свободным исполнителям задач рабочего стола узла.

rpaplatform_node_free_task_executors - gauge

Запускается каждую минуту со значением, равным свободным исполнителям задач узла.

Метки метрик узла

Метка	Описание
node_id	ID узла
node_name	Название узла

rpaplatform_queue - counter

Срабатывает для каждой операции OCR или ML в очереди.

Метки метрик узла

Метка	Описание
ap_id	ID автоматизированного процесса
ap_uuid	UUID автоматизированного процесса
ap_name	Название автоматизированного процесса
run_id	ID запуска автоматизированного процесса
run_uuid	UUID запуска автоматизированного процесса
queue_name	Имя очереди: ML, OCR
queue_operation	Операция очереди: отправить (send), получить (receive)

Метрики контейнера Docker

Для сбора метрик используется докер-плагин Telegraf . Пожалуйста, обратитесь к документации плагина для спецификации метрики.

https://github.com/influxdata/telegraf/tree/master/plugins/inputs/docker

Метрики ActiveMQ

Для сбора метрик используется плагин Telegraf ActiveMQ . Пожалуйста, обратитесь к документации плагина для спецификации метрики.

https://github.com/influxdata/telegraf/tree/master/plugins/inputs/activemq

Панели мониторинга платформы

Панели мониторинга отображаются в модуле Панели мониторинга и предоставляют доступ к графикам и визуализации данных из различных источников данных. Панели мониторинга размещаются на платформе Grafana. Панель мониторнга может быть создана двумя способами:

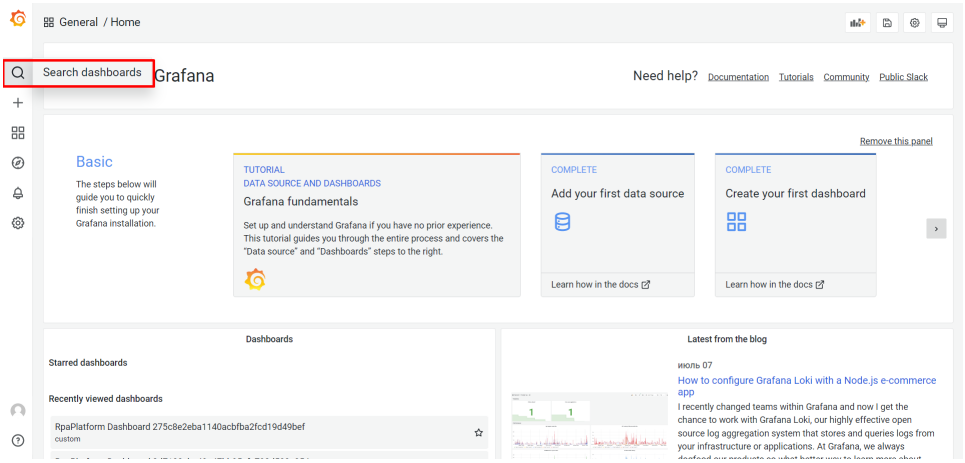
- из существующей в Grafana панели - панель мониторинга на сервере управления связана с панелью в Grafana через **ID панели.** Такой панелью мониторинга можно управлять из Grafana.
- используя **Grafana dashboard JSON -** платформа переносит панель мониторинга в Grafana, и в этом случае вы не можете управлять ею из Grafana. Такие панели мониторинга отображаются в следующей папке: **Grafana dashboards\custom folder**.

Создание панели мониторинга из существующей в Grafana панели

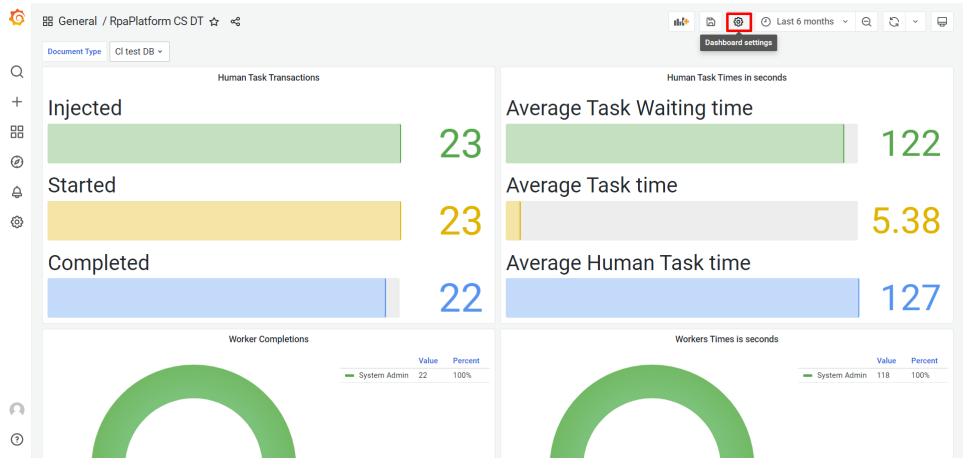
Чтобы создать новую панель мониторинга из существующей в Grafana панели, выполните следующие действия:

Создание панели мониторинга с использованием значения "uid"

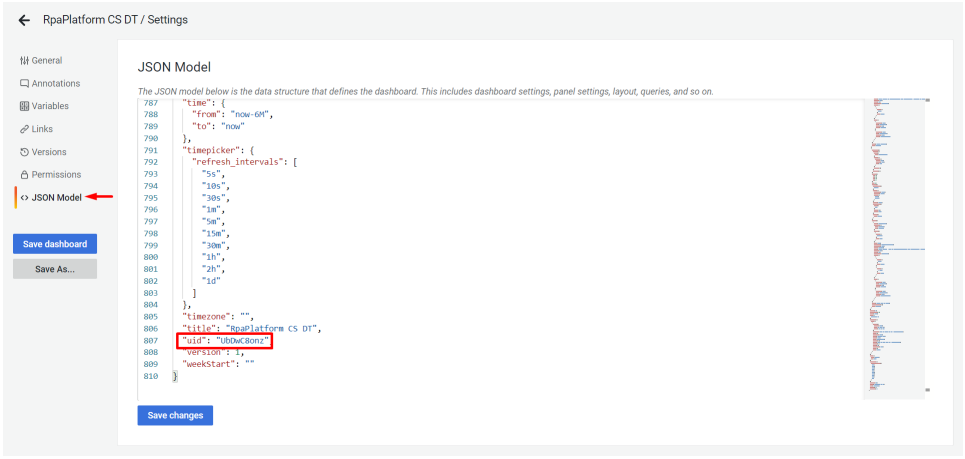
1. Перейдите в Grafana, нажмите на иконку лупы и ссылку **Search dashboards**.



2. Выберите панель, которую хотите использовать как источник данных для панели мониторинга. Нажмите иконку **Dashboard settings**.



3. Перейдите в модуль **JSON Model** и скопируйте значение из поля **"uid"**.



4. Перейдите в модуль **Панели мониторинга** Канцлер RPA и создайте новую панель мониторинга, вставив скопированное значение в поле **ID панели**. Пожалуйста, обратитесь к статье Создание новой панели мониторинга для получения более подробной информации.

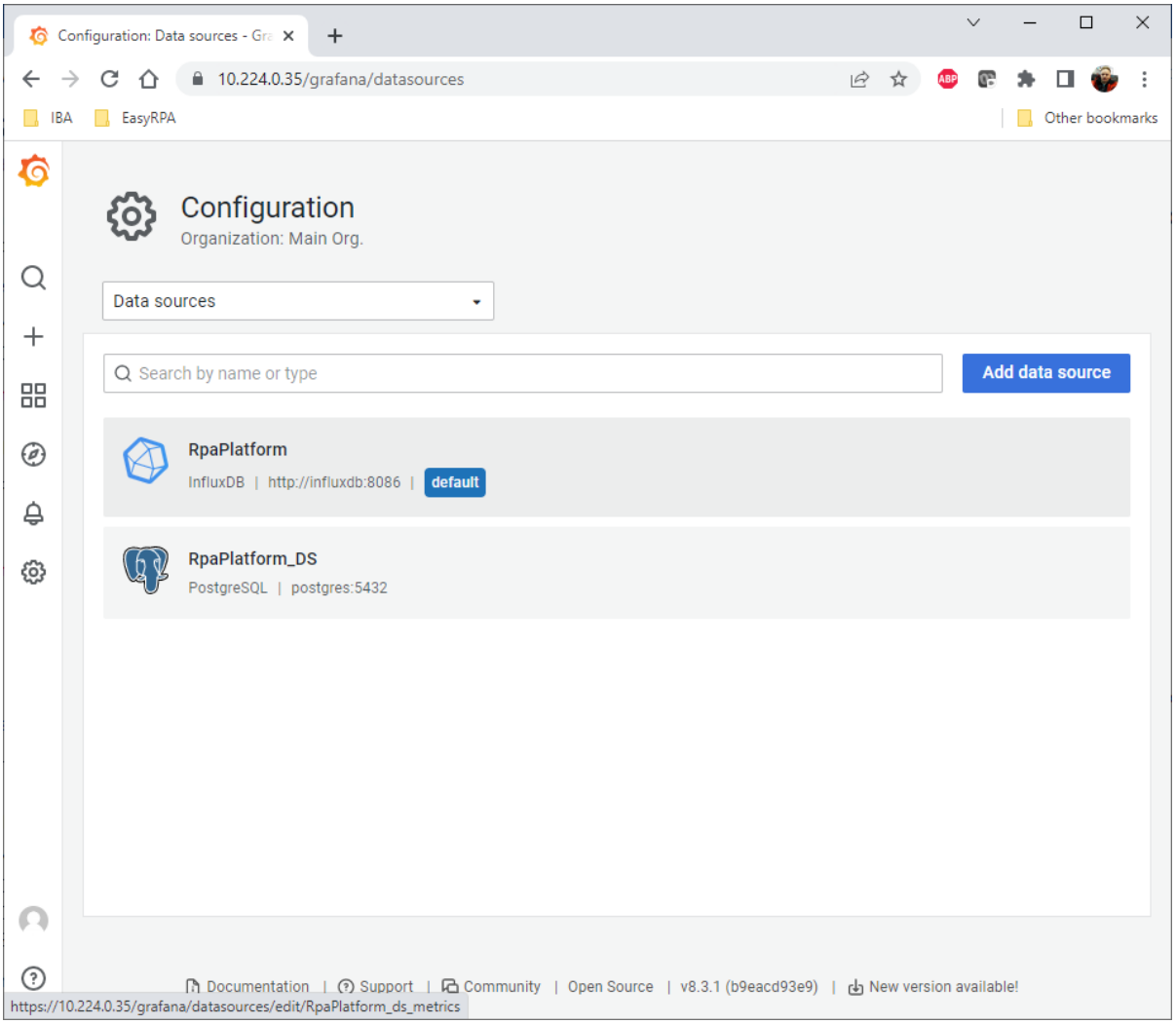
Создание панели мониторинга с использованием Grafana dashboard JSON

1. Перейдите в Grafana, нажмите на иконку лупы и ссылку **Search dashboards**.
2. Выберите панель, которую хотите использовать как источник данных для панели мониторинга. Нажмите иконку **Dashboard settings**.
3. Перейдите в модуль **JSON Model** и скопируйте полностью данные из поля **JSON Model**.
4. Сохраните скопированные данные в файл с расширением .json
5. Перейдите в модуль **Панели мониторинга** Канцлер RPA и создайте новую панель мониторинга используя кнопку **Загрузить JSON панели**. Пожалуйста, обратитесь к статье Создание новой панели мониторинга для получения более подробной информации.

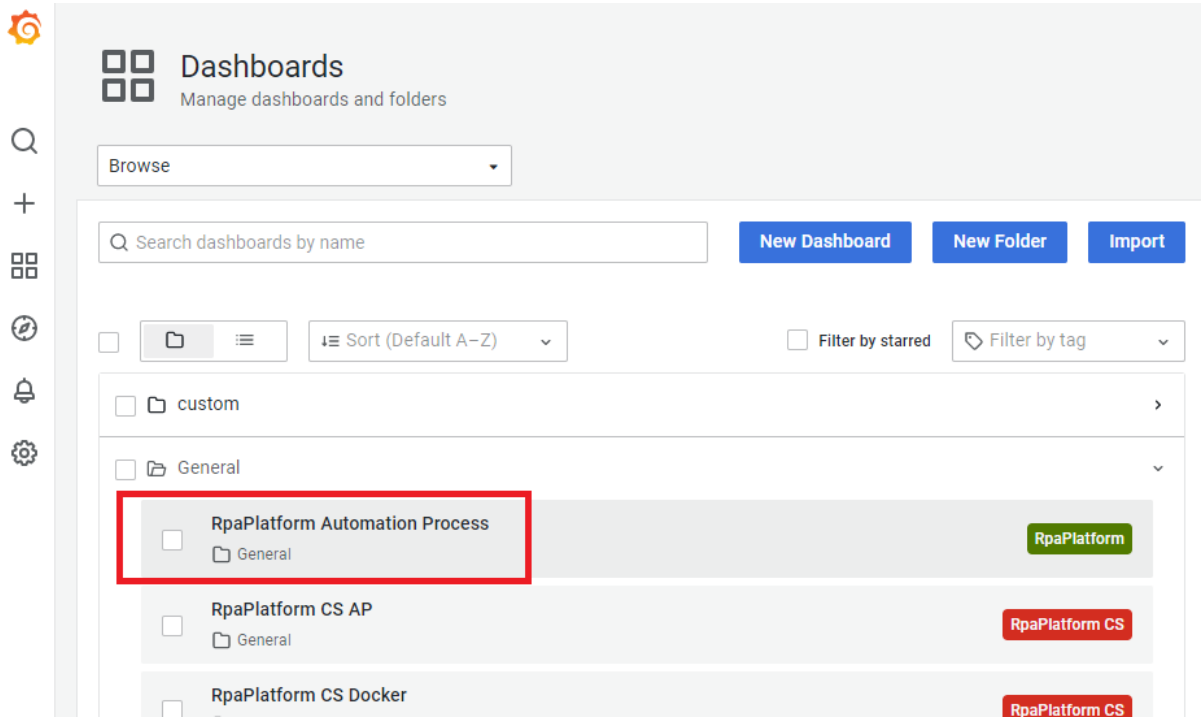
Источники данных для панели

На данных момент платформа предоставляет два источника данных:

- RpaPlatform
- RpaPlatform_DS



Источник данных **RpaPlatform** является источником данных временных рядов InfluxDB. Пожалуйста, обратитесь к статье Метрики платформы для получения более подробной информации. Вы можете использовать существующие панели как шаблоны для создания собственных.

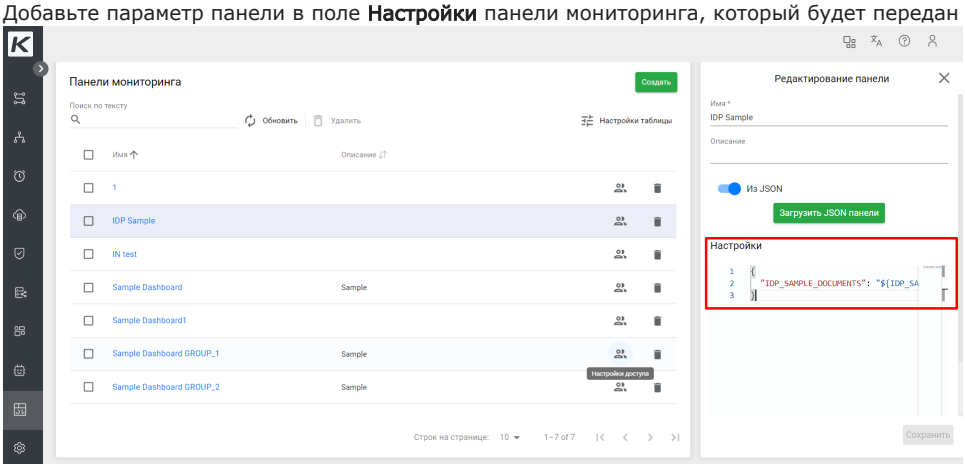


Источник данных **RpaPlatform_DS** обращается к Postgres Data Source таблицам. Пожалуйста, обратитесь к статье Хранилища данных для получения более подробной информации.

Использование источника данных хранилища данных

RpaPlatform_DS - это соединение postgres со схемой хранилища данных, которая содержит таблицы для хранилищ данных и пакетов документов. Чтобы передать имя хранилища данных/пакета документов в имя таблицы, используйте следующий подход:

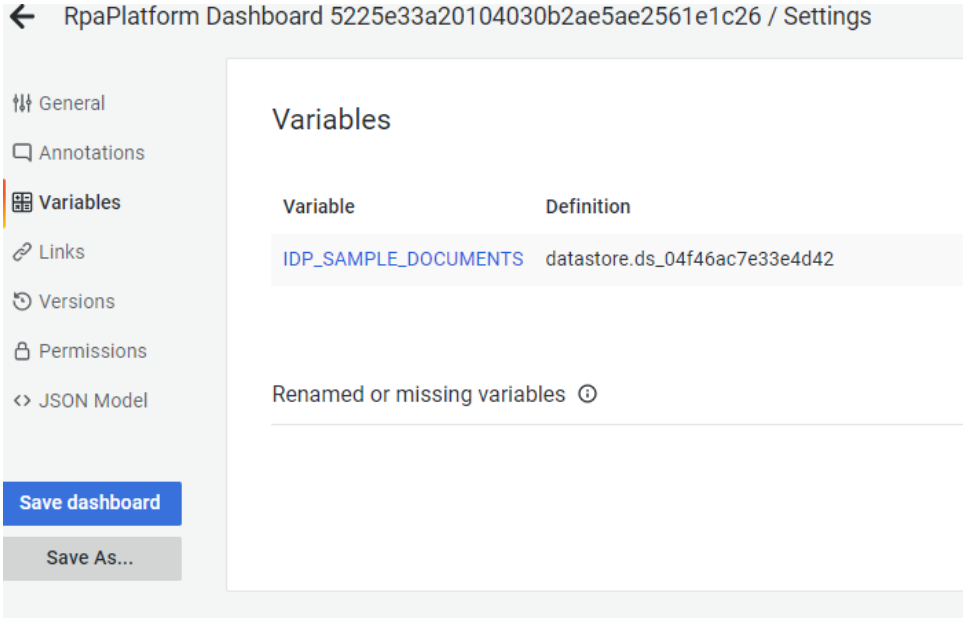
1. Добавьте параметр панели в поле **Настройки** панели мониторинга, который будет передан платформой в имя таблицы.



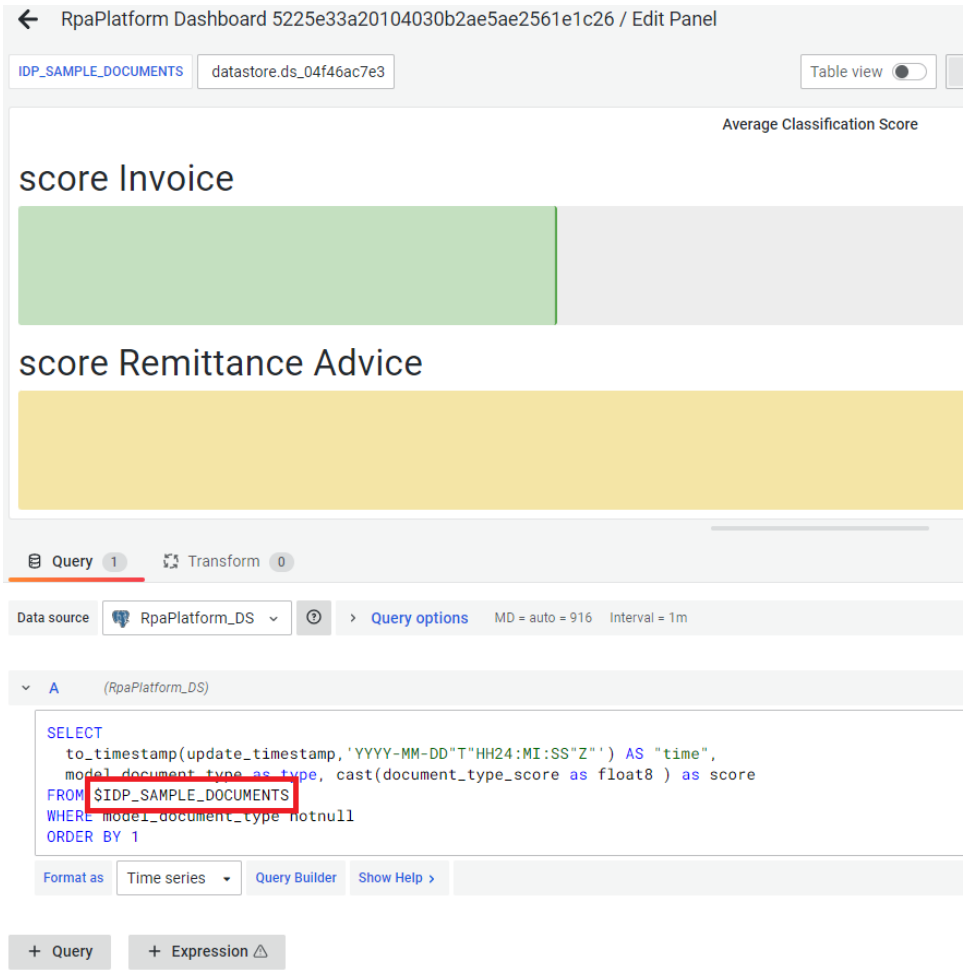

```
{

  "IDP_SAMPLE_DOCUMENTS": "${IDP_SAMPLE_DOCUMENTS}"

}
```

2. Панель Grafana получает параметр **IDP_SAMPLE_DOCUMENTS**, который содержит имя таблицы хранилища данных **IDP_SAMPLE_DOCUMENTS**. Параметр **${NAME}** ищет хранилище данных по имени и возвращает имя таблицы.

   Используйте параметр в панели Grafana в модуле **Variables**:



Используйте variable для обращения к таблице:



Вы можете найти пример панели мониторинга в примере Intelligent Document Processing (IDP).