.BSC

User guide

AUTE FRAMEWORK

Инструкция для пользователя

АВТОР: AUTE FRAMEWORK TEAM

ВЕРСИЯ: 1.4.1

Дата: 29.10.2018

Аннотация

В настоящем документе описывается интерфейс инструмента для проведения автоматизированного тестирования AuTe Framework.

СОДЕРЖАНИЕ

Термины и сокращения

API – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы.

MQ – промежуточное ПО для сообщения (Message Oriented Middleware). Оно позволяет независимым и, возможно, работающим не одновременно приложениям в распределённой системе обмениваться данными друг с другом.

Mock – функция, которая заменяет реальный объект в условиях теста и не выполняющая никакого осмысленного действия, содержит заранее запрограммированные ответы вызовов. То же самое, что заглушка метода. Мокировать можно MQ, HTTP и т.п.

MQ mocker – модуль для мокирования сообщений в очередь по специальным инструкциям, поступающим от AuTe Framework.

SOAP – протокол обмена структурированными сообщениями в распределённой вычислительной среде.

JSON – это открытый текстовый формат, который использует человеко-читаемый текст для обмена данными в виде объектов, состоящих из пар ключ-значение.

XPath – язык запросов к элементам XML-документа.

Введение

1.1. Наименование системы

AuTe Framework – Фреймворк, используемый для автоматизированного тестирования REST API.

1.2. Назначение системы

Фреймворк для автоматизации изолированного или комплексного тестирования REST API компонентов системы.

ПО позволяет выполнять:

- Автоматический регулярный запуск автоматических тестов
- Поддержку встраивания тестов в конвейер CI
- Параллельный запуск автоматических тестов вручную/по расписанию
- Возможность объединять тесты в наборы (запуск наборов)
- Хранение истории запуска тестов и их результатов
- Эмуляцию конечных систем настраиваемыми заглушками

Реализована гибкость конфигурирования:

- Подключаемая эмуляция конечных систем заглушками
- Разные ответы заглушек для разных тестов (настраиваемые заглушки)
- Доработка тестов/заглушек без участия разработчика
- Поддержка заглушками MQ, REST, SOAP
- Возможность проведения изолированного автоматизированного тестирования, заменяя всех поставщиков данных (функций) заглушками

1.3. Описание основных бизнес-функций

ПО предоставляет следующий набор основных функций:

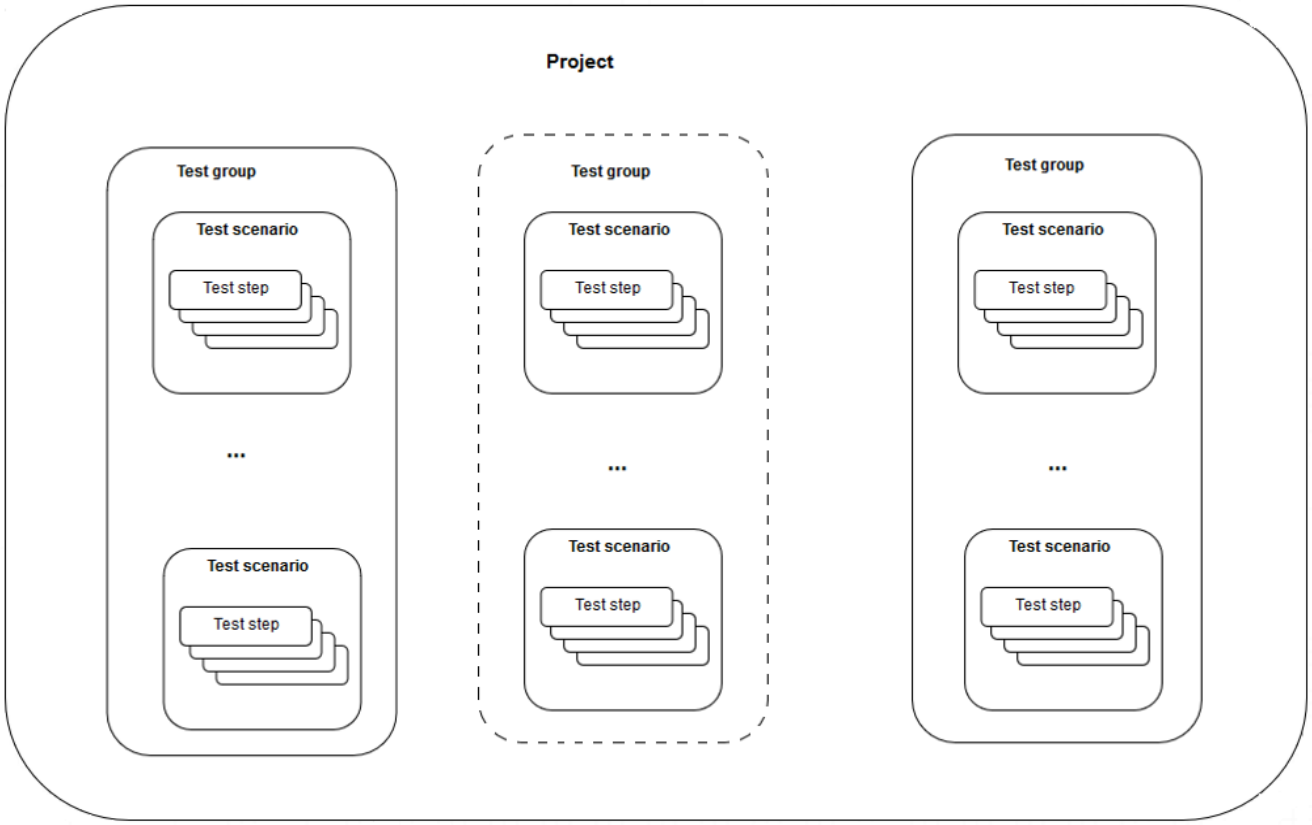
- Управление тестовыми сценариями
- Управление группами сценариев
- Управление общими параметрами проекта
- Использование управляемых заглушек
- Использование и тестирование очередей
- Автоматический и ручной запуск тестовых сценариев
- Получение детального отчета о результатах выполнения сценария
- Экспорт отчета о результатах выполнения сценария

1.4. Требования к рабочему месту пользователя

Для работы с AuTe Framework на рабочем месте пользователя должны быть установлены следующие программные средства:

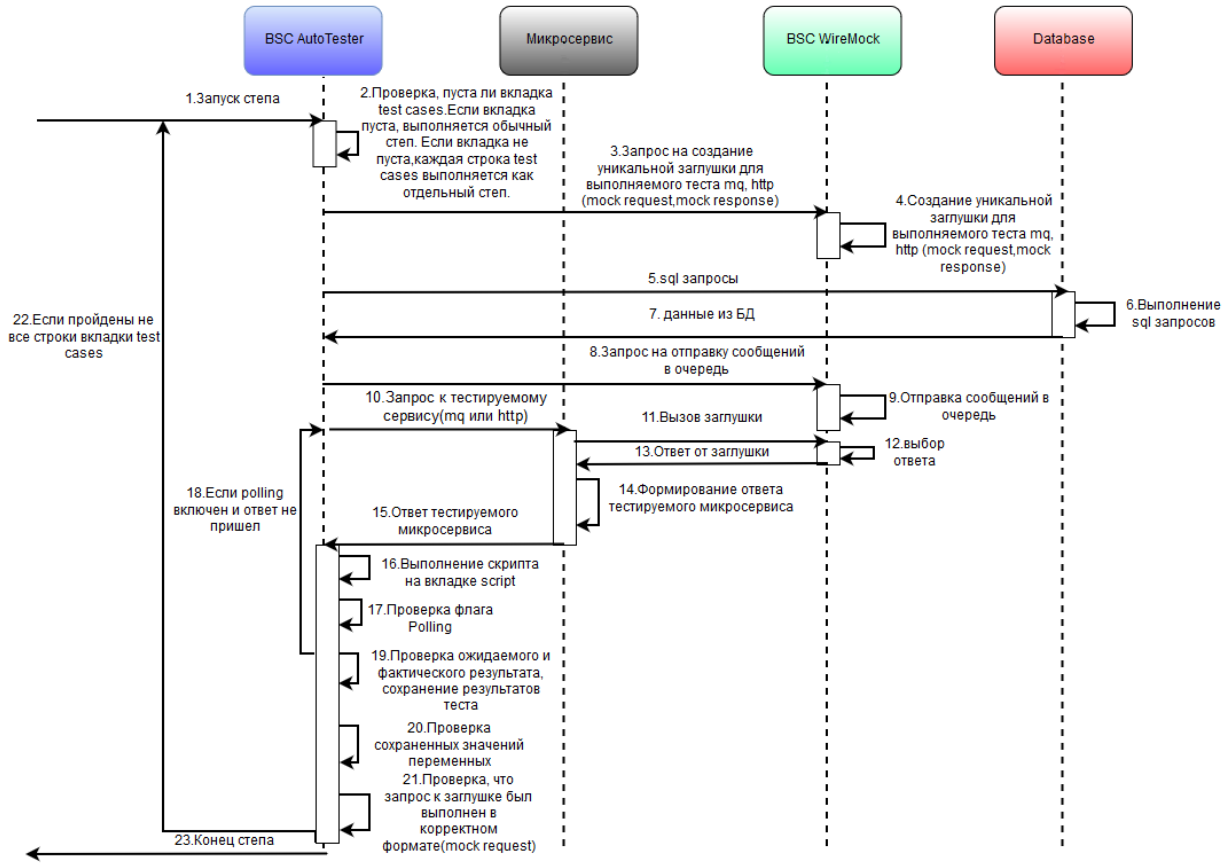
- Интернет-браузер (Mozilla Firefox, Microsoft Edge – для функционала построения отчётов, для остальной функциональности возможно использование Mozilla Firefox, Microsoft Edge, Google Chrome, Яндекс Браузер)
- JRE версия 1.8 и выше

2.Структура проекта с автотестами



На рисунке представлена модель проекта. На каждый микросервис создается отдельный проект. В проекте тестовые сценарии можно логически объединять в группы, а так же тестовый сценарий может не принадлежать ни к одной из групп. Каждая группа состоит из тестовых сценариев, тестовые сценарии состоят из шагов.

Следующий рисунок дает представление о принципе выполнения шагов тестовых сценариев:


BSC AutoTester
Микросервис
BSC WireMock
Database
1.Запуск степа
2.Проверка, пуста ли вкладка test cases.Если вкладка пуста, выполняется обычный степ. Если вкладка не пуста,каждая строка test cases выполняется как отдельный степ.
3.Запрос на создание уникальной заглушки для выполняемого теста mq, http (mock request,mock response)
4.Создание уникальной заглушки для выполняемого теста mq, http (mock request,mock response)
5.sql запросы
6.Выполнение sql запросов
7. данные из БД
8.Запрос на отправку сообщений в очередь
9.Отправка сообщений в очередь
10.Запрос к тестируемому сервису(mq или http)
11.Вызов заглушки
12.выбор ответа
13.Ответ от заглушки
14.Формирование ответа тестируемого микросервиса
15.Ответ тестируемого микросервиса
16.Выполнение скрипта на вкладке script
17.Проверка флага Polling
18.Если polling включен и ответ не пришел
19.Проверка ожидаемого и фактического результата, сохранение результатов теста
20.Проверка сохраненных значений переменных
21.Проверка, что запрос к заглушке был выполнен в корректном формате(mock request)
22.Если пройдены не все строки вкладки test cases
23.Конец степа

3.Подготовка и настройка

3.1. Настройка BSC-WIREMOCK

Необходимые компоненты для запуска сервера управляемых заглушек:

- **BscWireMock** (/atf-wiremock-<версия>/)

Описание файлов в директории **BscWireMock**:

- **lib** - библиотеки для работы с IBM MQ;
- **mappings** - описание маппингов REST-заглушек, редактируется через UI;
- **__files** - файлы ответов, используемые в REST-заглушках;
- **application.properties** - файл настроек приложения;
- **mq-properties-rc.yml** - файл описания мокируемых очередей;
- **atf-wiremock-<версия>.jar** - запускаемое приложение.

Для работы мокирования очередей необходимо в файле application.properties указать параметры для подключения к провайдеру:

- **mq.manager** - Тип провайдера (IBM_MQ, RABBIT_MQ, ACTIVE_MQ);
- **mq.host** - Хост (пример: mq.bscmsc.ru);
- **mq.port** - Порт (пример: 9011);
- **mq.username** - Имя пользователя;
- **mq.password** – Пароль.

В этом же файле требуется указать значения следующих параметров:

- **properties.yaml.file** - имя файла с описанием мокируемых очередей (MQ);
- **test.id.header.name** - название свойства сообщения, по которому определяется запускаемый тест в AuTe Framework. Данное значение должно соответствовать параметру, указанному в Автотестере: Проект -> "Настройки" -> "Название заголовка testId".
- **logging.file=bsc-wiremock.log** - название файла журнала для логирования.

3.2. Запуск BSC-WIREMOCK

Для запуска WIREMOCK необходимо выполнить команду:

java -Dloader.path=lib/ -Dfile.encoding=UTF-8 -jar atf-wiremock-<версия>.jar

Приложение будет запущено на порту, который указан в параметре **server.port** в файле *application.properties*.

В Автотестере в env.yml необходимо указать адрес в параметре wireMockUrl, например:

wireMockUrl: 'http://10.2.7.146:1397'

Интерфейс для управления REST-заглушками и просмотр журналов доступны по адресу:

http://10.2.7.146:1397/ui/

3.3. Остановка и перезапуск BSC-WIREMOCK

Необходимые компоненты для настройки количества логируемых вызовов MQ Mock:

– **BscWireMock** (/atf-wiremock-<версия>/)

Остановка и перезапуск **BscWireMock** требуется в случае необходимости изменить какие-либо настройки компонента (см. п. 3.1).

Например, для изменения количества логируемых вызовов MQ Mock требуется выполнить следующие действия:

- в интерфейсе **BscWireMock** в верхнем правом углу нажать «**Save to back storage**» - для сохранения маппингов из памяти на диск;
- остановить приложение;
- в файле **application.properties** изменить значение **mq.requestBufferSize** на требуемое (по умолчанию установлено значение = 1000);
- запустить приложение (см. п. 3.2).

3.4. Настройка MQ заглушек

Для мокирования очереди необходимо:

- Тестируемый сервис настроить на новую очередь QUEUE_OUT_MOCK.
- Mq-mocker настроить (в файле properties.yml) на проксирование сообщений: чтение из очереди QUEUE_OUT_MOCK и запись в очередь QUEUE_OUT.

Таким образом, все сообщения, поступающие в очередь QUEUE_OUT_MOCK будут пересылаться в очередь QUEUE_OUT при условии, что в mq-mocker отсутствуют инструкции для особой обработки сообщения (инструкции могут поступать из Автотестера по http-api).

Пример содержимого файла properties.yaml:

```
mockMessageList:
  - sourceQueueName: 'GETDEPOSITPRODUCTSOUTBOUNDQUEUE_MOCK'
    testId: null
    httpUrl: null
    XPath: null
    responses:
  - destinationQueueName: 'GETDEPOSITPRODUCTSOUTBOUNDQUEUE'
    responseBody: null
```

- Поле "sourceQueueName" - очередь, которую "слушает" mq-mocker
- Поле "destinationQueueName" - очередь, в которую будет направлен результат. Если все остальные поля не назначены (null), то сообщение будет переслано из sourceQueueName в destinationQueueName.
- Поле "testId" – параметр в заголовке сообщения, по которому следует фильтровать входящие сообщения.
- Поле "XPath" - используется для проверки тела сообщения, которое получил Wiremock.
- Поле "httpUrl" – url, если поле заполнено, то wiremock делает HTTP POST запрос по этому url, передавая туда тело сообщения.
- Поле "responseBody" - тело ответного сообщения, которое отправит wiremock.

Также есть возможность задать несколько ответов на одно входящее сообщение. Пример:

```
mockMessageList:
  - sourceQueueName: 'GETDEPOSITPRODUCTSOUTBOUNDQUEUE_MOCK'
      testId: null
      httpUrl: null
```

```
    XPath: null
  responses:
  - destinationQueueName: 'DEST_Q_1'
      responseBody: "response 1"
  - destinationQueueName: 'DEST_Q_2'
      responseBody: "response 2"
```

3.5. Запуск автотестера

Необходимые компоненты автотестера:

- atf-application-<версия>.jar (/test_dir/atf-application-<версия>.jar)
- env.yml (шаблон файла настроек, /test_dir/env.yml)

Для запуска необходимо выполнить следующие действия:

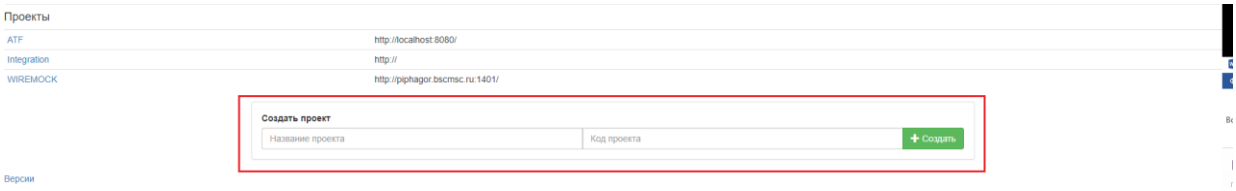
1. Создать директорию (рабочая директория) для хранения автотестов.
2. В файле настроек env.yml указать путь к рабочей директории и URL тестируемого сервиса.

   При необходимости указать параметры подключения к БД, к брокеру AMQP и к сервису WireMock.
3. В файле run.bat можно изменить порт для работы приложения (по умолчанию 8080).
4. Запустить run.bat и открыть в браузере: http://localhost:8080 (или другой порт, в зависимости от настроек, выполненных в п.3).

4.Как сделать …

4.1. Создать и настроить новый проект

1. Запустить приложение
2. На главной странице создать новый проект,указав название и код проекта



3. Для запуска тестовых сценариев необходимо указать значения следующих параметров в блоке **projectStandMap** файла env.yml:

   serviceUrl - содержит базовый url тестового стенда (например *http://test-application:8080/*)

   dataBase – параметры подключения к базе данных:

   url - строка подключения к базе данных (например jdbc:oracle:thin:@test:1521:TEST)

   user – пользователь для подключения к базе данных

   password - пароль для подключения к базе данных

   wireMockUrl - ссылка на сервис заглушек BSCWireMock (пример: http://10.2.7.165:1397)

   amqpBroker - параметры подключения к очередям MQ

   host – адрес сервера очередей

   mqService - тип брокера сообщений (например IBM_MQ, RABBIT_MQ)

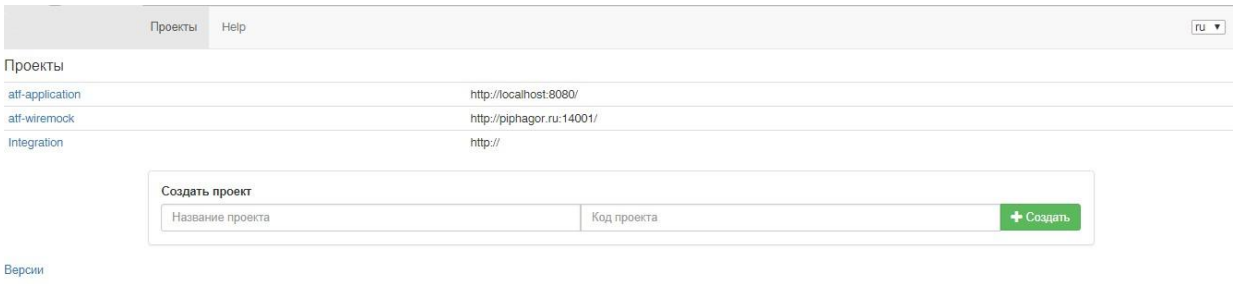
   username - пользователь, от имени которого будет вестись работа с очередью

   password - пароль для подключения

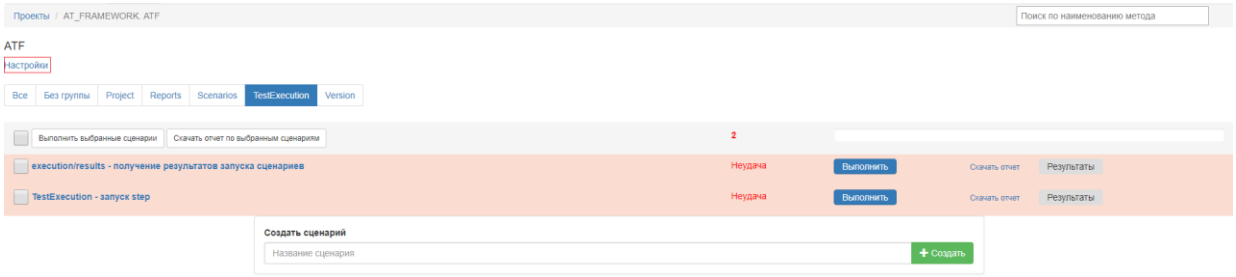
   port - порт для подключения
4. Перезапустить приложение

Так же некоторые настройки можно выполнить в самом приложении. Для этого необходимо:

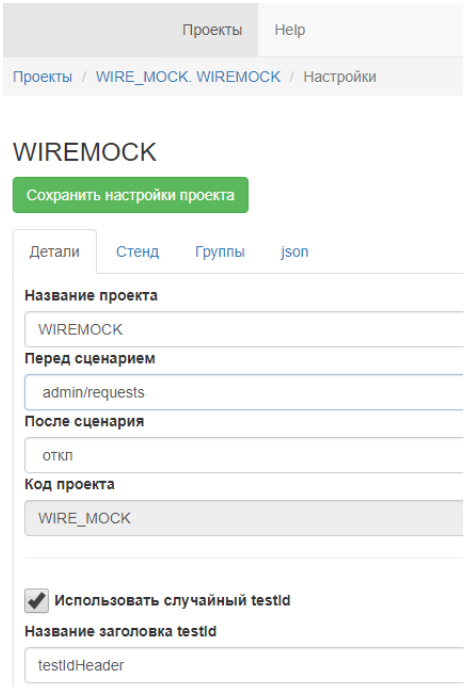
1. Выбрать необходимый проект на главной странице



2. Кликнуть на странице проекта на ссылку “Настройки”



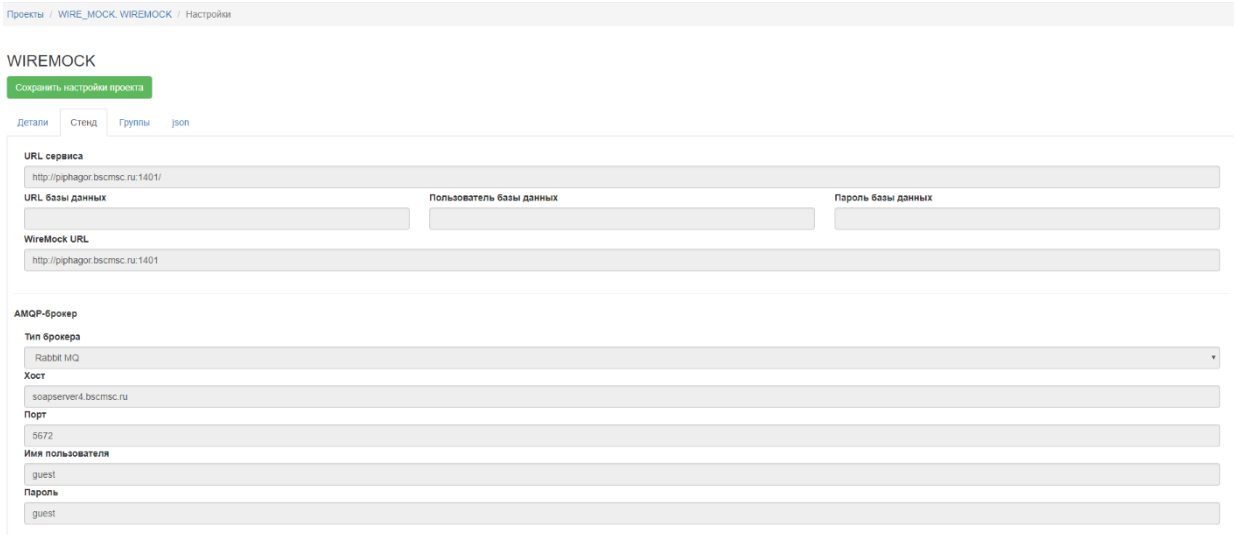
3. В открывшемся меню выбрать вкладку “Детали”



На вкладке доступны следующие элементы:

- Поле, содержащее «Название проекта». Поле доступно для редактирования.
- Поле «Перед сценарием» используется для указания сценария, который необходимо выполнять при запуске перед каждым сценарием проекта (например, авторизация).
- Поле «После сценария» используется для указания сценария, который необходимо выполнять при каждом запуске после каждого сценария проекта (например, логаут).
- Чек-бокс «Использовать случайный testId». Wiremock – сервис заглушек должен иметь возможность отличать запросы, направляемые в рамках разных шагов, для их проверки и отправки заданных в шаге ответов. При включенном чекбоксе этот функционал реализуется, к каждому запросу от AuTe Framework к тестируемому микросервису добавляется http-заголовок со случайно генерируемым уникальным ID. Тестируемый продукт должен пересылать указанный ID в сервис заглушек.
- Название http-заголовка указывается в поле «Название заголовка TestId».

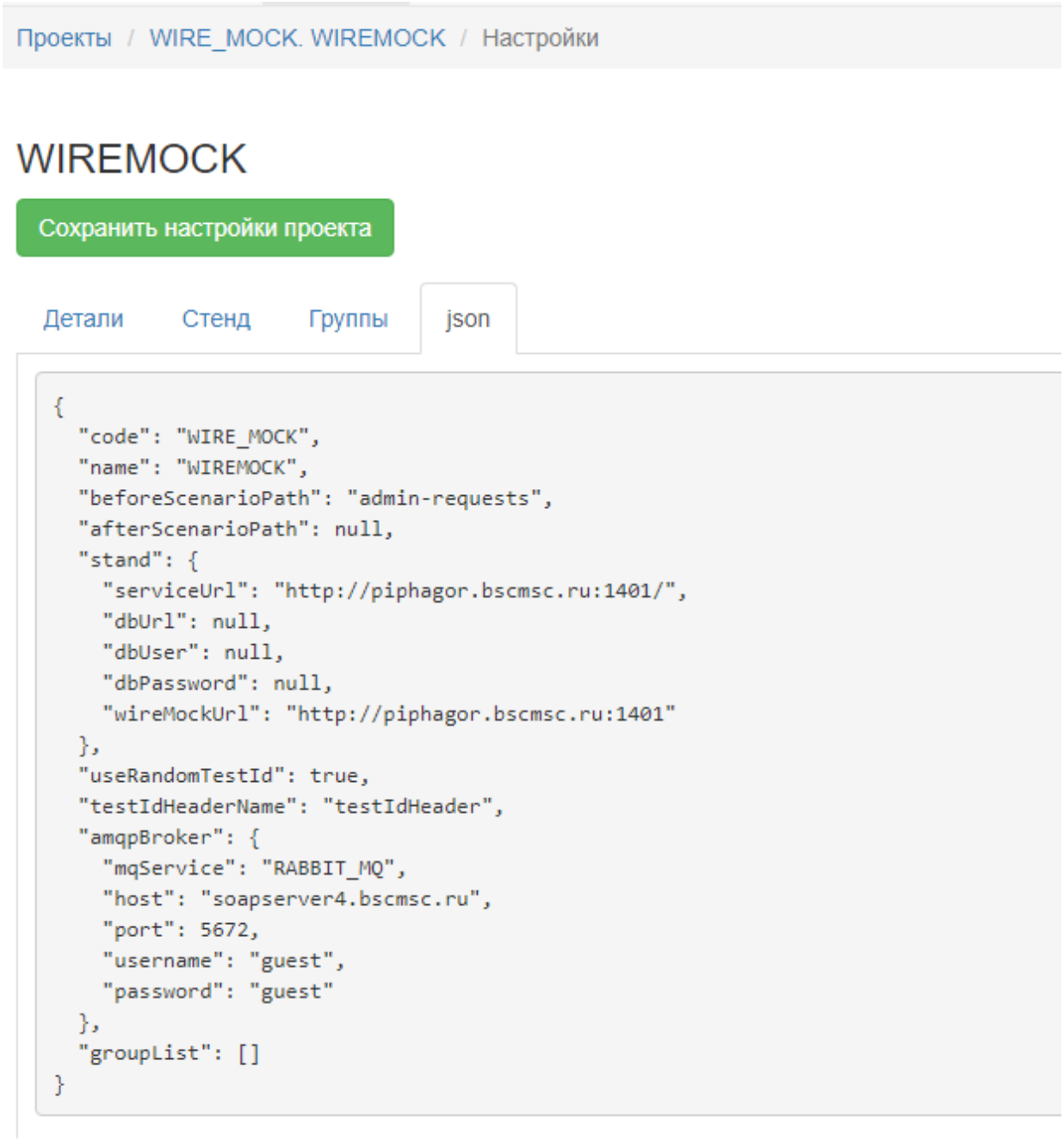
4. В открывшемся меню выбрать вкладку “Стенд”



На данной вкладке отображаются параметры тестового стенда, которые указаны в файле env.yml.

Редактирование параметров на форме недоступно. Изменить параметры можно только непосредственно в файле env.yml с последующим перезапуском AuTe Framework.

5. В открывшемся меню выбрать вкладку “json”

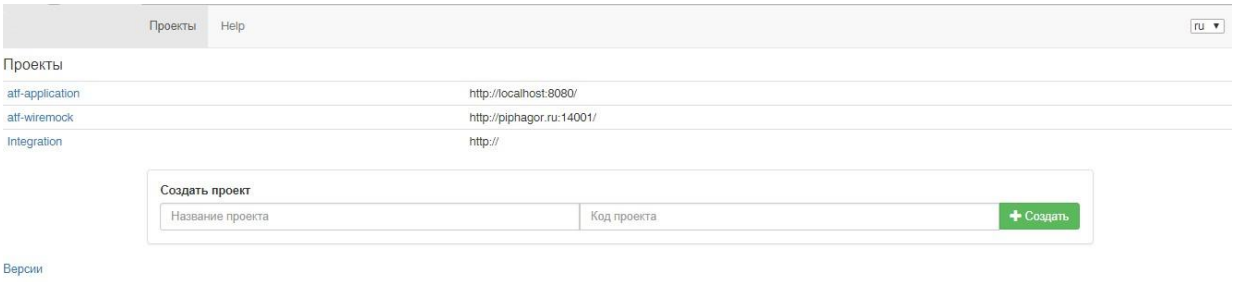


На данной вкладке отображается блок, в котором представлены все настройки проекта в формате json.

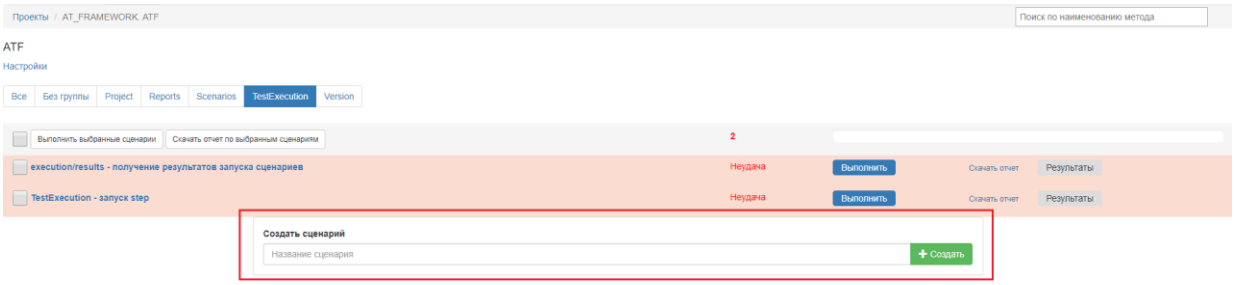
4.2. Создать новый сценарий

Чтобы создать новый сценарий, необходимо:

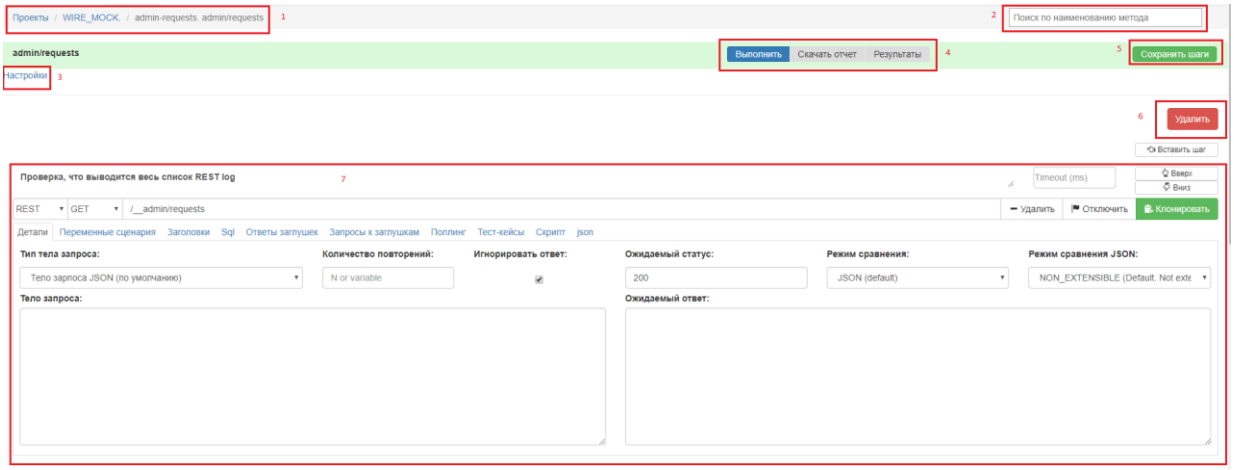
1. Запустить приложение
2. Выбрать необходимый проект на главной странице

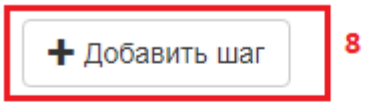


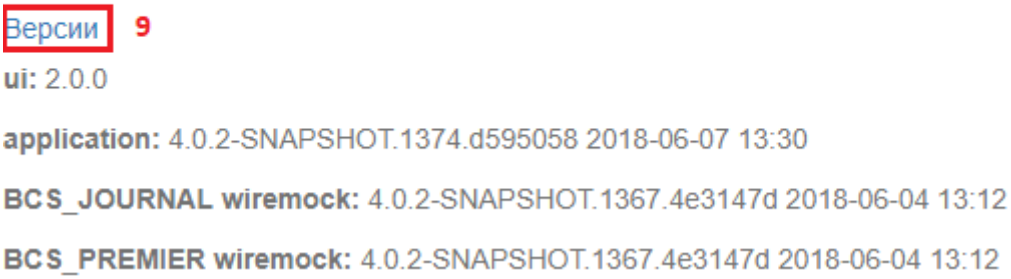
3. Указать название нового сценария и нажать кнопку "**Создать**". Сценарий будет создан в текущей группе.



Если кликнуть на сценарий в списке, откроется страница сценария, на которой доступны следующие поля и действия (нумерация также приведена на демонстрационных рисунках):





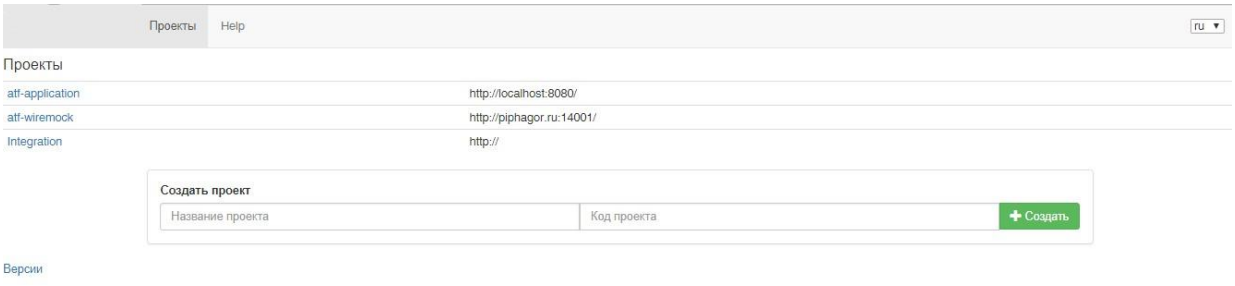


1. Информация о расположении теста в файловой системе
2. Поиск по наименованию REST-запроса
3. Переход к настройкам сценария по кнопке «Настройки»
4. Блок, содержащий кнопку запуска исполнения сценария, а так же результаты прогона сценария
5. Сохранение изменений сценария по кнопке «Сохранить шаги»

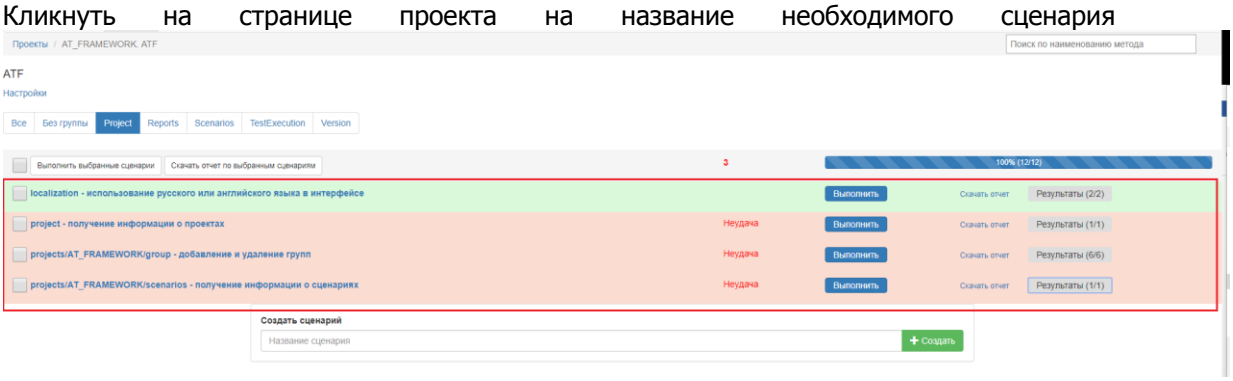
6. Удаление сценария по кнопке «Удалить»
7. Блоки с описанием существующих шагов сценария
8. Добавление нового шага сценария по кнопке «Добавить шаг»
9. Просмотр версии AuTe Framework по ссылке «Версии»

Далее следует настроить тестовый сценарий:

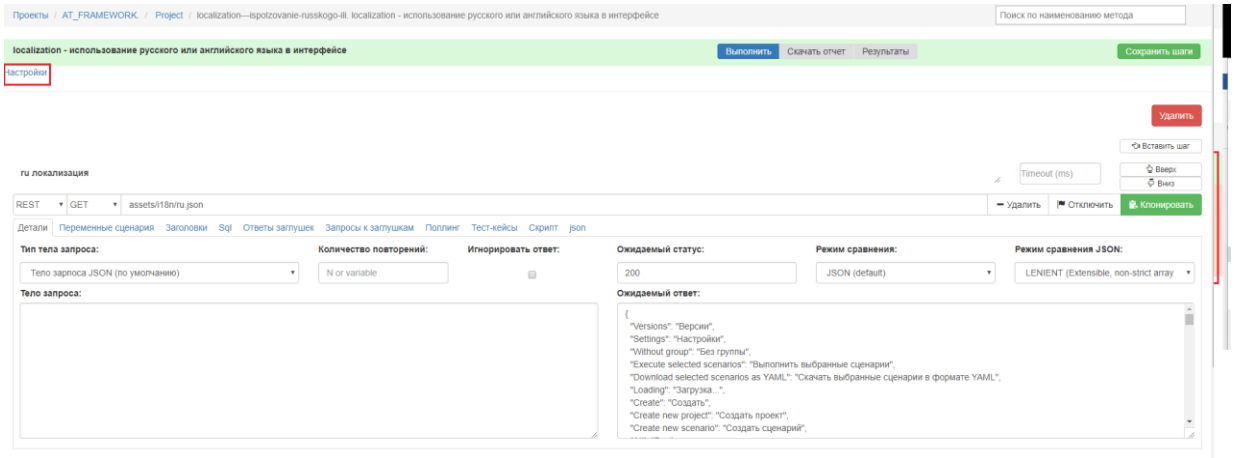
1. выбрать необходимый проект на главной странице



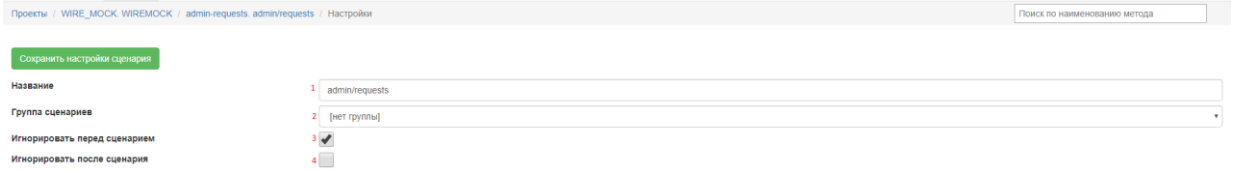
2. Кликнуть на странице проекта на название необходимого сценария



3. Нажать ссылку “Настройки”



На форме доступны следующие элементы (нумерация также приведена на демонстрационном рисунке):



1. Поле, содержащее Название сценария. Поле доступно для редактирования. При переименовании сценария переименовывается папка, в которой хранится сценарий.
2. Поле «Группа сценариев», в котором указана группа, к которой принадлежит текущий сценарий. В данном поле может быть указана только одна группа. Папка сценария в файловой системе располагается в соответствующей папке группы.

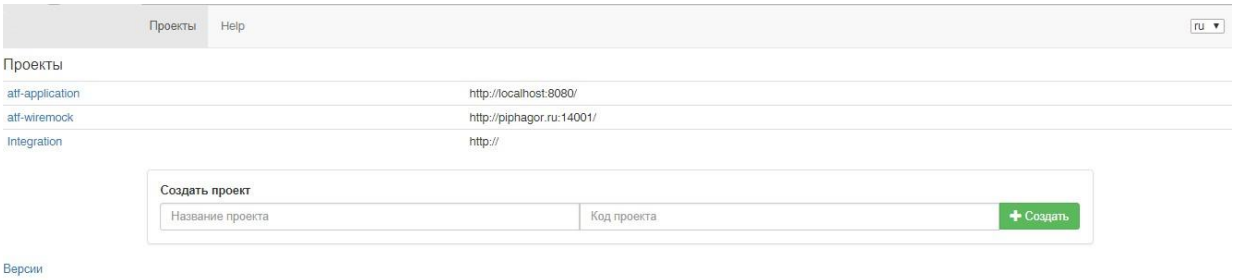
3. Поле «Игнорировать перед сценарием» используется для отключения воспроизведения соответствующего сценария, указанного в настройках проекта, перед выполнением текущего сценария.
4. Поле «Игнорировать после сценария» используется для отключения воспроизведения соответствующего сценария, указанного в настройках проекта, после выполнения текущего сценария.

Сохранение настроек производится при нажатии на кнопку **«Сохранить настройки сценария».**

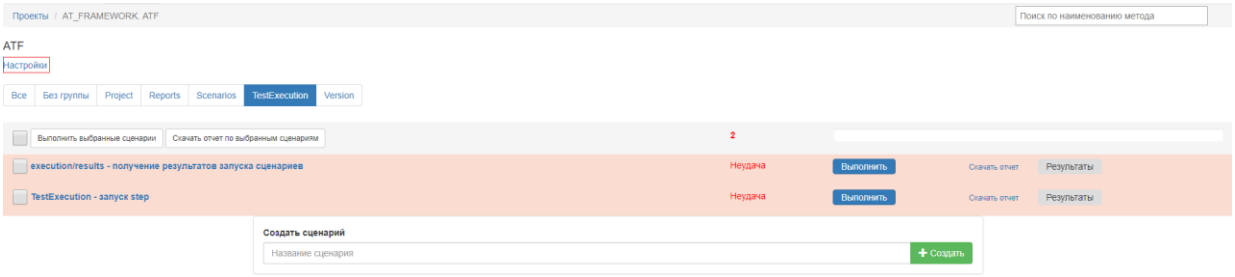
4.3. Работать с группами

Чтобы создать новую группу:

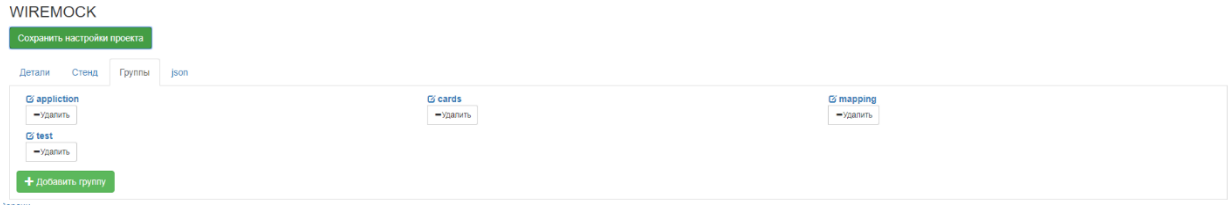
1. Запустить приложение
2. Выбрать необходимый проект на главной странице



3. Кликнуть на странице проекта на ссылку “Настройки”



4. В открывшемся меню выбрать вкладку “Группы”



5. Нажать на кнопку “Добавить группу”, ввести имя новой группы, кликнуть ОК
6. Нажать кнопку “Сохранить настройки проекта”

Также на вкладке “Группы” отображается набор групп, созданный пользователем. С группами доступны следующие действия:

Редактирование – нажать на наименование существующей группы.

Удаление – нажать на кнопку «Удалить» и подтвердить удаление (группа удаляется вместе со всеми вложенными сценариями).

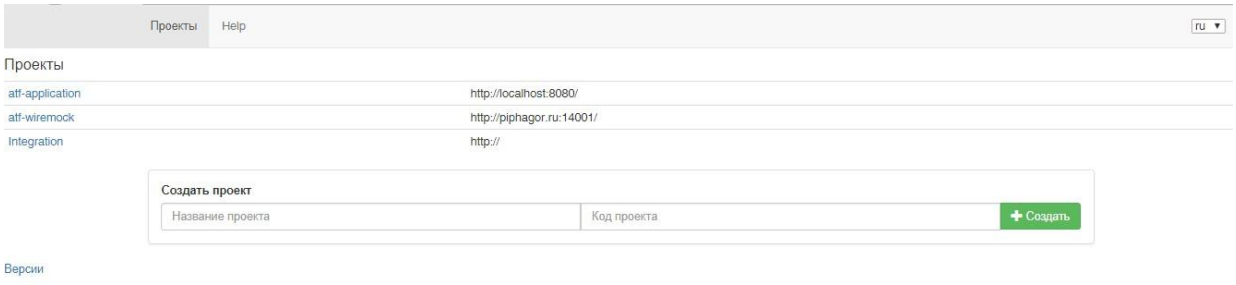
Чтобы сохранить изменения, необходимо нажать на кнопку «Сохранить настройки проекта».

Все группы созданных сценариев соответствуют директориям в папке /<project_dir>/scenarios/. Поддерживается только один уровень вложенности.

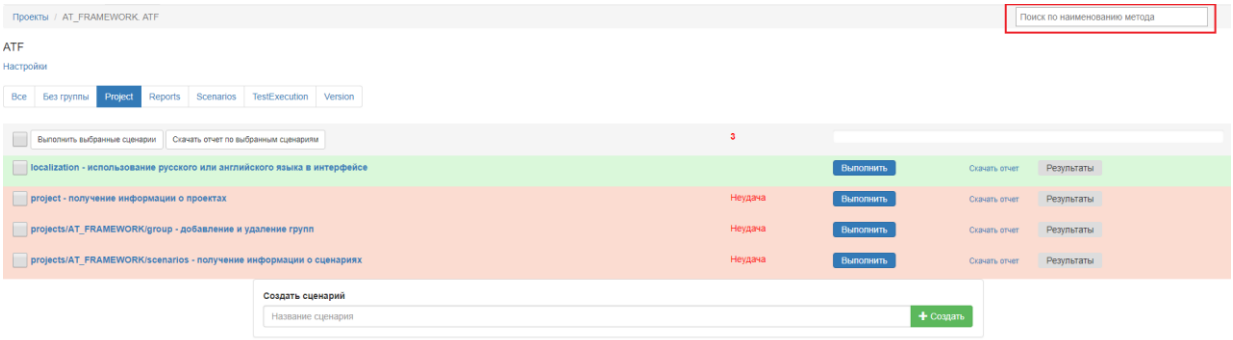
4.4. Найти существующий тест

Чтобы использовать поиск по существующим сценариям, необходимо:

1. Запустить приложение
2. Выбрать необходимый проект на главной странице



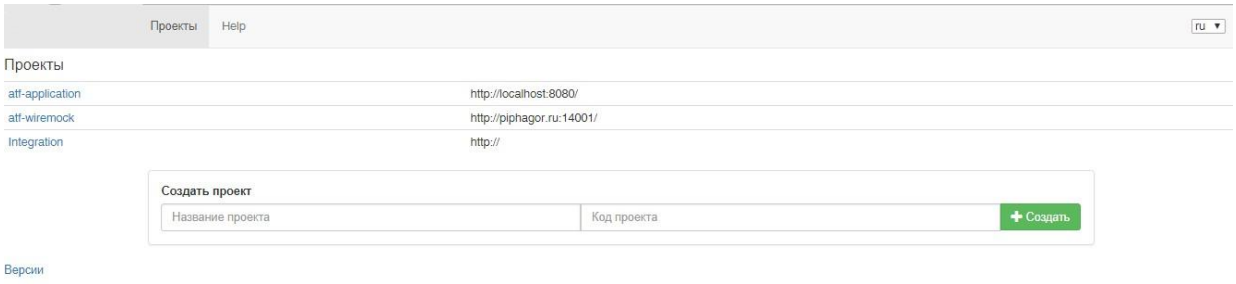
3. Ввести текст для поиска в блок “Поиск”



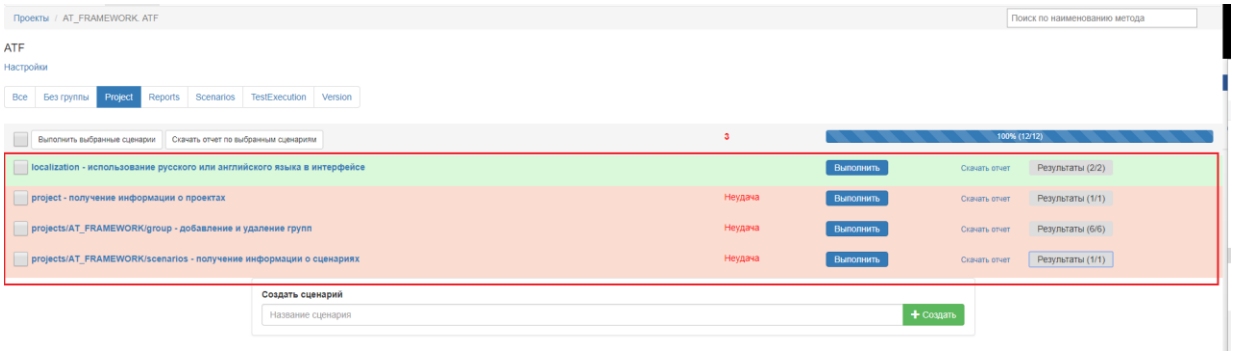
4.5. Добавить, удалить, переместить шаги (и другие действия с шагами сценария)

Чтобы в сценарий добавить шаг:

1. Запустить приложение
2. Выбрать необходимый проект на главной странице



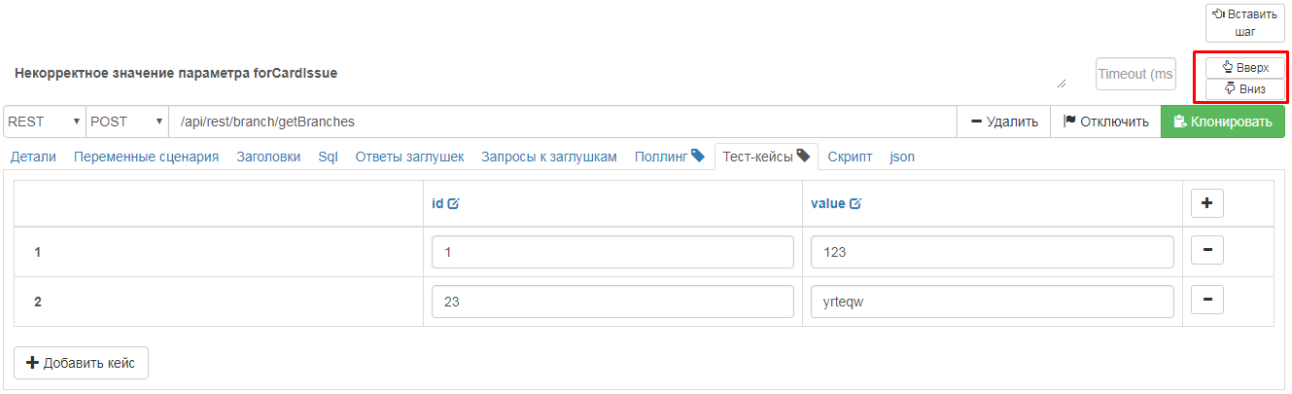
3. Кликнуть на странице проекта на название необходимого сценария



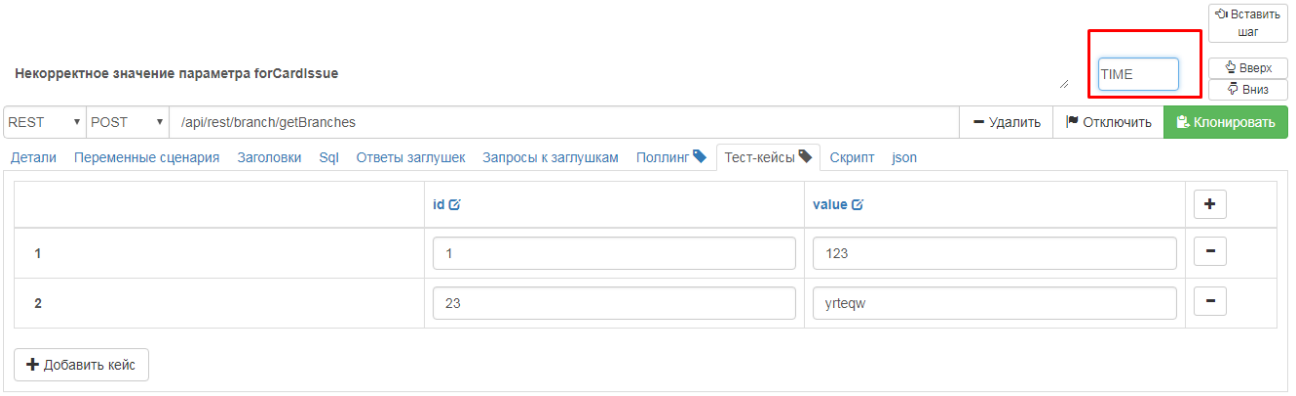
4. Нажать кнопку “Добавить шаг”

В списке шагов доступны следующие действия:

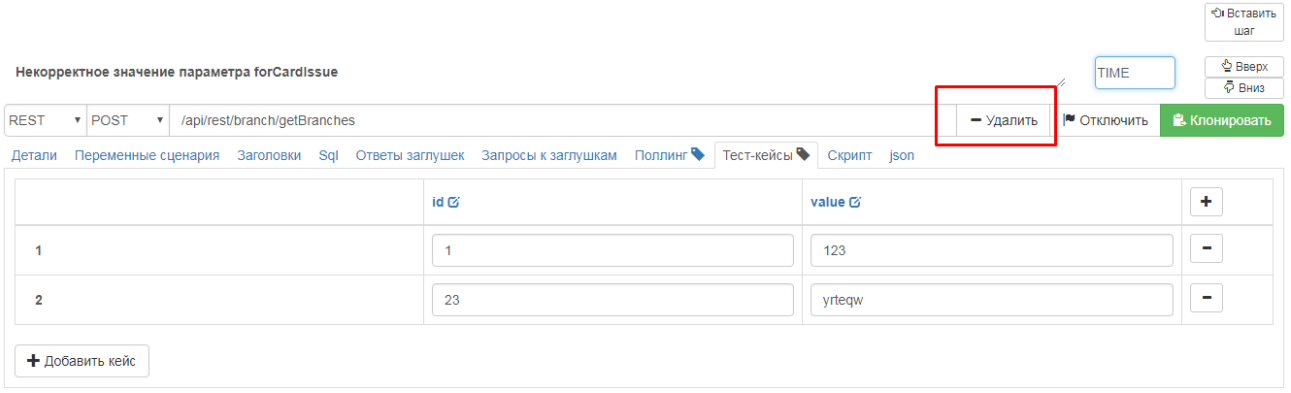
- Переместить вверх\вниз – для изменения положения шага в рамках тестового сценария;



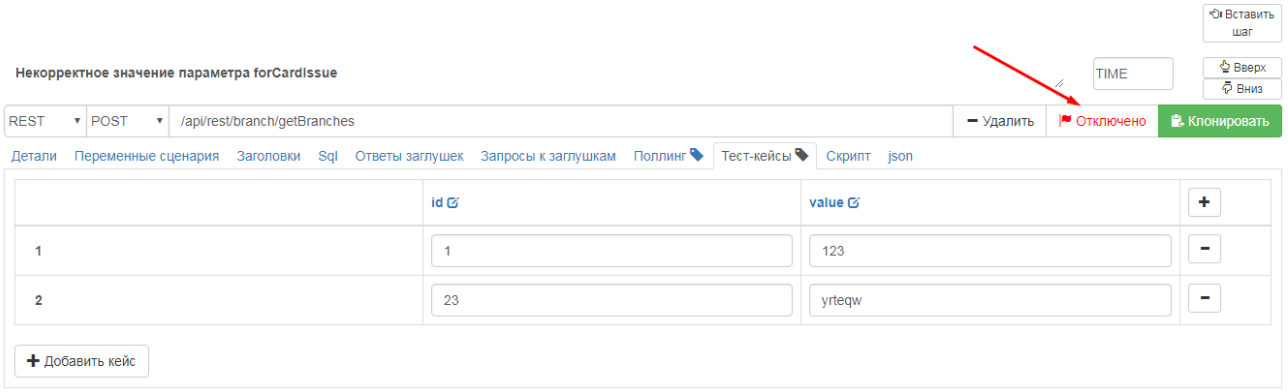
- Добавить задержку перед шагом – используется для повышения гибкости выполнения сценария, позволяет указывать числа (в ms) или переменные;



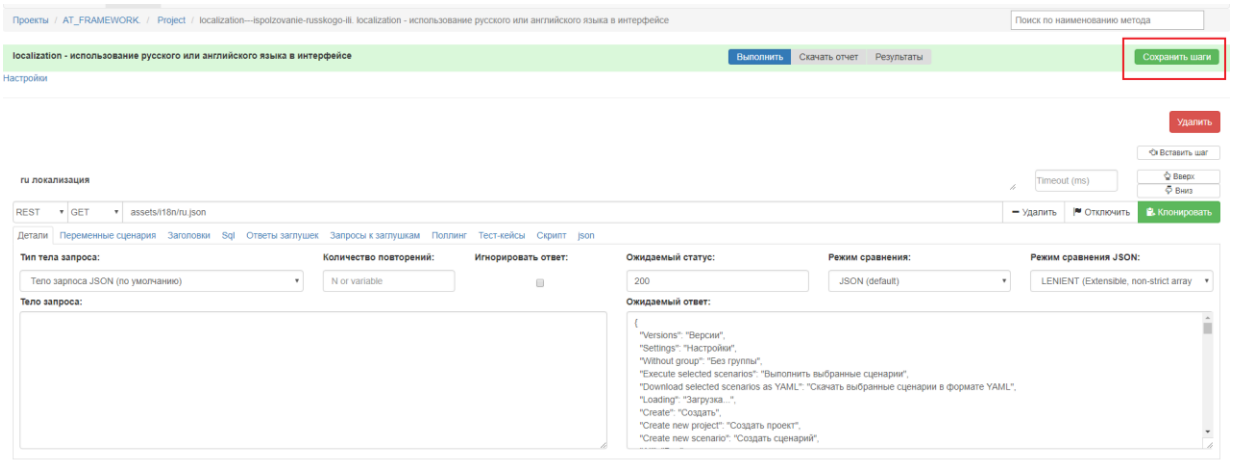
- Удалить – используется для удаления шага из сценария;



- Отключить при прогоне сценария – используется при необходимости проигнорировать выбранный шаг при прогоне сценария;



- Клонировать – используется для создания полной копии шага, данное действие становится доступным после первого сохранения, клонированный шаг отображается сразу после клонируемого шага;
- Добавить – используется для добавления нового шага;
- Вставить шаг – используется для добавления скопированного шага в указанное место в сценарии;
- Сохранить шаги



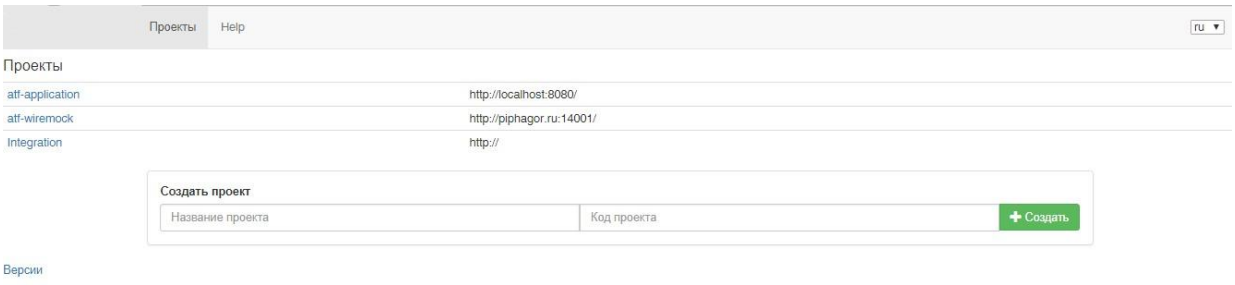
4.6. Запустить сценарии

Запуск сценариев возможен двумя способами:

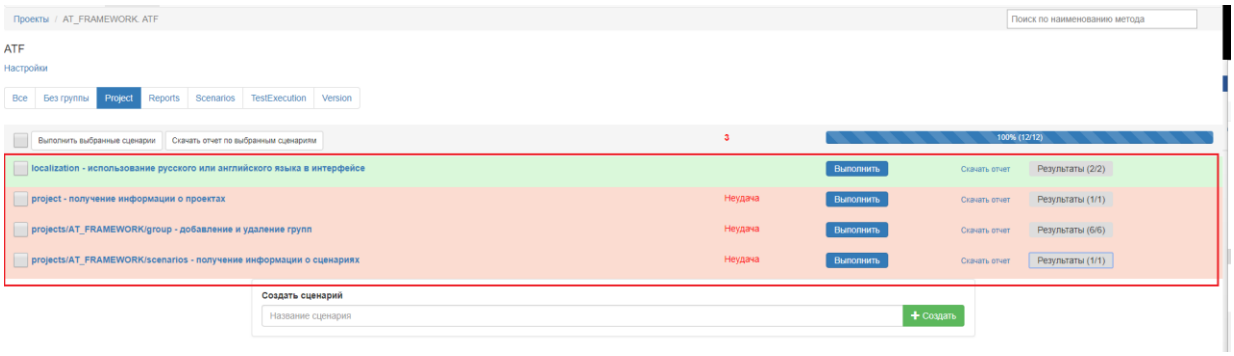
1. Через CI систему, например, Jenkins
2. Через UI AT Frmework'a

Чтобы осуществить запуск через UI AT Frmework'a, нужно:

1. Запустить приложение
2. Выбрать необходимый проект на главной странице



3. Выбрать необходимые сценарии из списка, выделить галочкой.



4. Нажать кнопку “Выполнить”

Выбранные и запущенные сценарии выполняются параллельно.

5.Описание пользовательского интерфейса

5.1. Главная страница проекта, возможности

Переход на главную страницу проекта осуществляется при выборе проекта из списка.

На странице доступны следующие элементы и действия (нумерация также приведена на демонстрационном рисунке):

1. Поиск по названию REST-запроса
2. Переход к настройкам проекта
3. Список групп сценариев
4. Блок со списком сценариев и действиями для их выполнения и экспорта отчетов
5. Блок для создания нового сценария.

По умолчанию отображаются все сценарии проекта.

Список маркируется следующим образом:

- Зеленым цветом окрашены сценарии, последний запуск которых завершился успешно.
- Красным цветом окрашены сценарии, последний запуск которых завершился с ошибками.
- Без окраски отображаются сценарии, для которых нет данных о запуске *(*например, новый сценарий, который не был запущен ни разу*).*

Отчет по результатам выполнения сценария доступен сразу после завершения теста и остается доступным до следующего запуска теста, даже если приложение будет перезапущено. В отчетах отображаются статусы выполнения каждого шага с возможностью просмотра тела запроса, фактического и ожидаемого результатов с указанием различающихся строк и детали с описанием возникшей ошибки.

5.2. Блок описания шага тестового сценария

При добавлении нового шага, все поля в его блоке по умолчанию пустые. Для каждого шага можно добавить его описание и время отсрочки запуска (в мс).

Для выполнения шага может быть заполнена строка с url REST-запроса. В строке указывается относительный url. Так же есть возможность указывать в запросе переменные сценария (пример: /rest/items/{itemId }).

Для шага существует 2 режима тестирования: REST и JMS. По умолчанию выбран режим REST.

Слева от строки url можно выбрать тип REST-запроса. На данный момент реализованы следующие типы: GET, POST, PUT, DELETE.

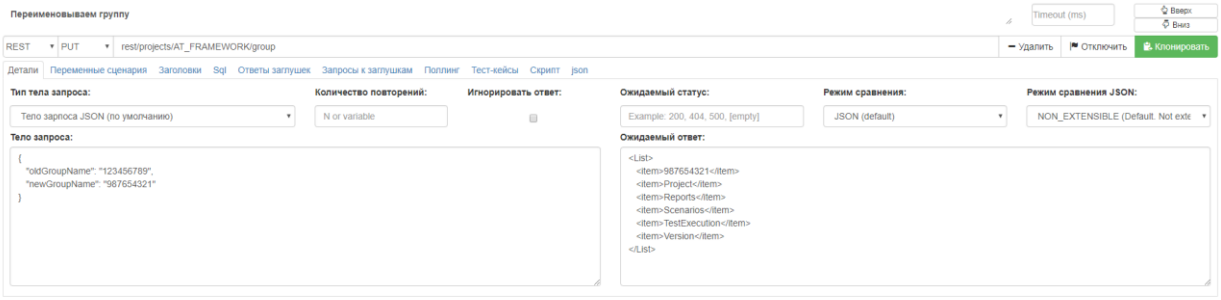
5.2.1. Вкладки для описания шага тестового сценария

В данном разделе приведено описание пунктов меню и доступных действия для описания шага тестового сценария.

5.2.1.1. Вкладка «Детали»

Наполнение данной вкладки зависит от режима выполнения шага: REST или JMS. Ниже приведено описание обоих случаев.

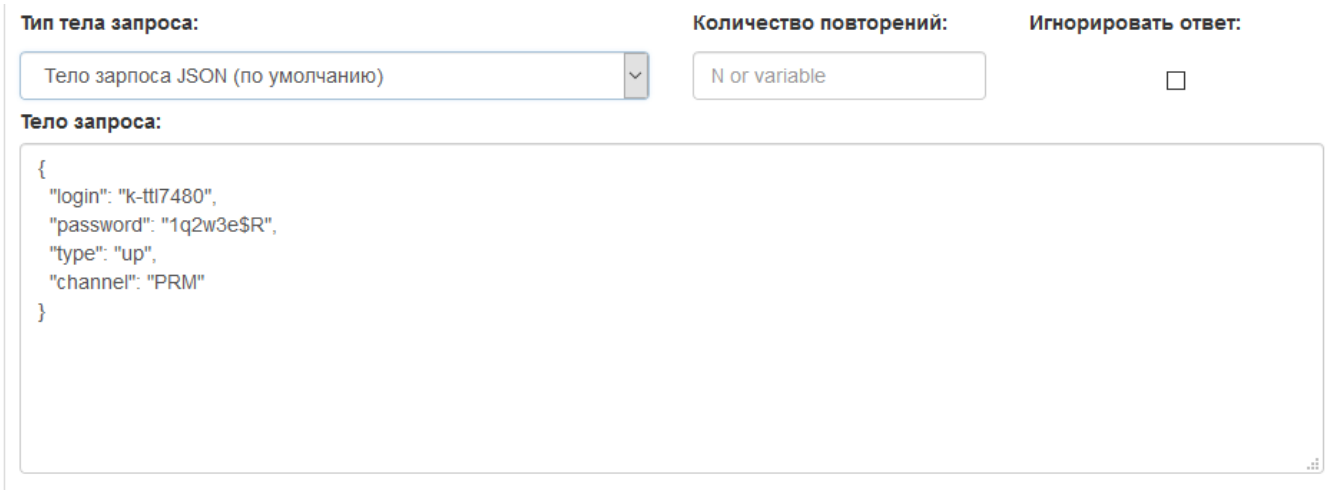
5.2.1.1.1. Режим выполнения REST



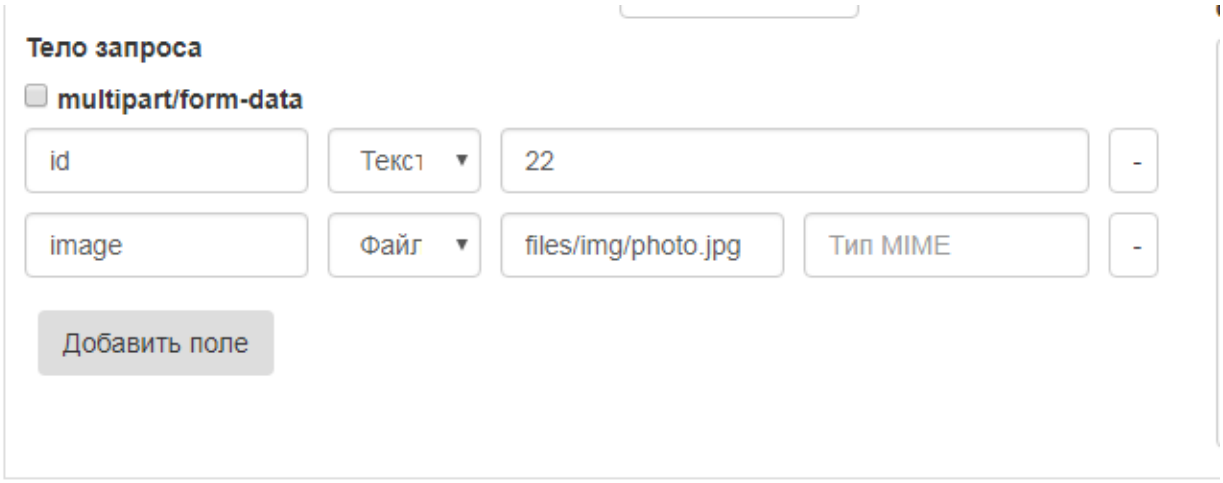
Вкладка предназначена для описания содержимого отправляемого запроса, а также для указания ожидаемого результата запроса.

Тело запроса можно описать двумя способами:

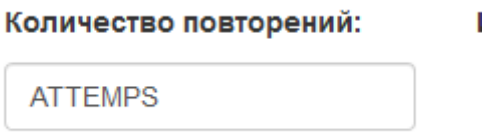
- В формате JSON (по умолчанию) – в результате выполнения шага отправится тело запроса в json – формате:



- FORM-data – в результате выполнения шага отправится форма с текстовыми полями или файлами. В случае отправки файла путь к нему указывается относительно директории с проектом. Например: если файл расположен в директории /projects/< project_dir >/files/img/photo.jpg, то на форме необходимо указать: files/img/photo.jpg



В поле **«Number of repetitions»** указывается сколько раз будет отправлен запрос и произведено сравнение ответов. Поле не обязательно для заполнения – по умолчанию запрос будет отправлен 1 раз. В поле можно указать значение «0» - тогда шаг не будет выполнен. Так же в данное поле можно подставлять значение переменной, прописывая только название переменной

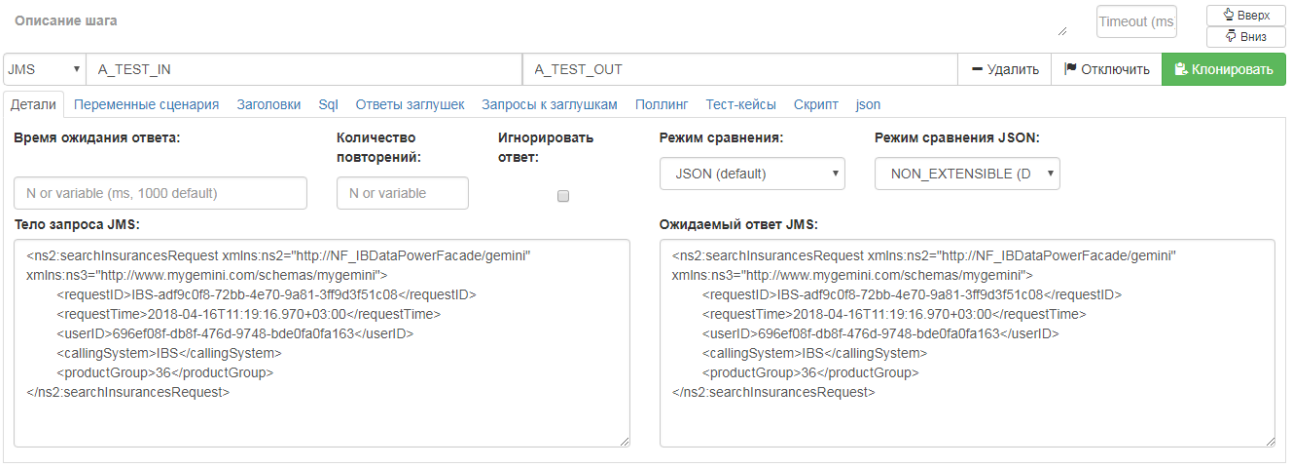


При включении чек-бокса **«Игнорировать ответ»** сравнение ответов производиться не будет, однако если заполнено соответствующее поле, то будет проверяться код ожидаемого статуса ответа.

В поле **«Ожидаемый статус»** указывается код ожидаемого статуса ответа *(например: 200, 403, 500).*

В блоке **«Ожидаемый ответ»** допустимо использование ключевого слова *ignore* для игнорирования значений некоторых параметров *(например: системное время, динамический id и т.д.)*.

5.2.1.1.2. Режим выполнения JMS



Для корректной работы должен быть заполнен блок подключения к MQ *amqpBroker* в файле env.yml.

В поле **«Тело запроса JMS»** необходимо указать запрос, который отправляется в очередь.

В поле **«Время ожидания ответа»** необходимо указать время задержки проверки ответа от очереди.

В инструменте доступно несколько вариантов сравнения фактического ответа с ожидаемым.

1. **JSON** (используется по умолчанию) – для сравнения двух json-объектов. Существует несколько режимов сравнения:
   1.1 *NON_EXTENSIBLE* (режим по умолчанию) - не расширяемый, нестрогий порядок элементов в массивах.
   1.2 *STRICT* - не расширяемый, строгий порядок элементов в массивах.
   1.3 *LENIENT* - расширяемый, нестрогий порядок элементов в массивах.
   1.4 *STRICT_ORDER* - расширяемый, строгий порядок элементов в массивах.
2. **Full match –** фактический результат проверяется на полное соответствие ожидаемому.
3. **Mask *ignore* -** Сравнение ответов как строк с возможностью игнорирования части строки. Игнорируемая часть строки указывается ключевым словом *ignore*.

Пример:

Ожидаемый результат: <xml><datetime>*ignore*</datetime><name>Item name</name></xml>

Фактический результат: <xml><datetime>2018-01-22 17:50:24</datetime><name>Item name</name></xml>

В теле ответа могут использоваться переменные, а так же скрипты, например тело ответа:

```
{
  "header": {
    "serverTime": "*ignore*"
  },
  "result": {
    "weekends": [
      <f>result = []; index = 0; scenarioVariables.QUERY1.forEach(function(element) { result.push('"' + element.WEEKEND + '"'); }); result.join(',');</f>
    ]
  }
}
```

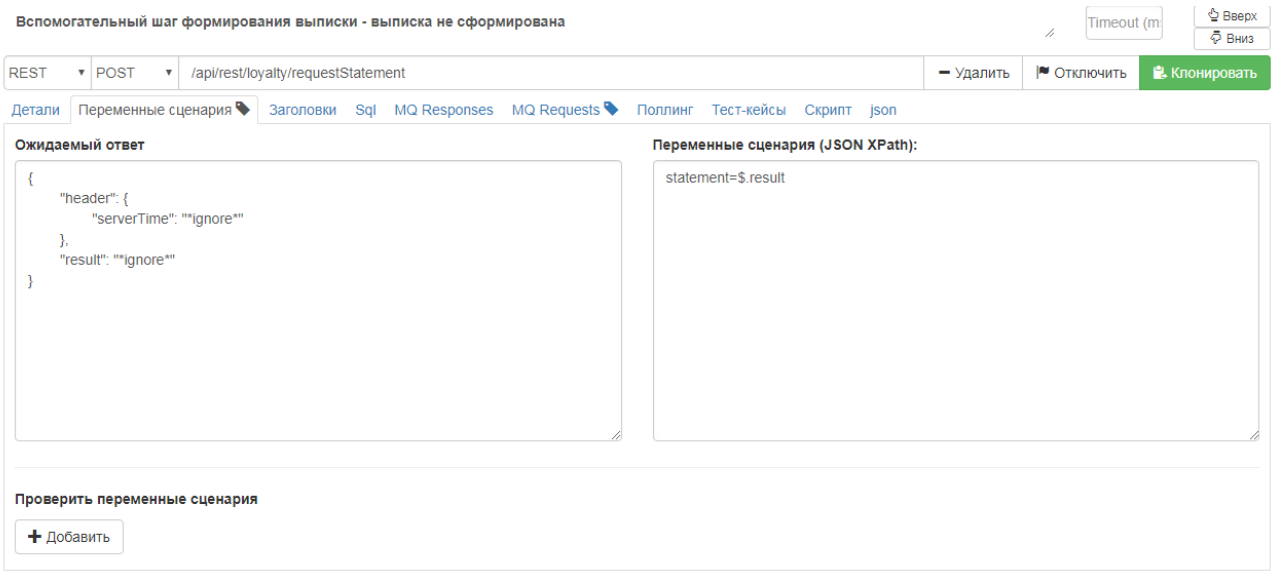
5.2.1.2. Вкладка «Переменные сценария»

Вкладка используется для указания переменных, которые будут хранить значения, полученные при ответе на текущий запрос. В переменные можно записывать и те значения, которые в ожидаемом ответе помечены как *ignore*.

Переменные сценария можно вызывать, а так же переопределять в последующих шагах.

Переменные описываются в XPath. Пример:

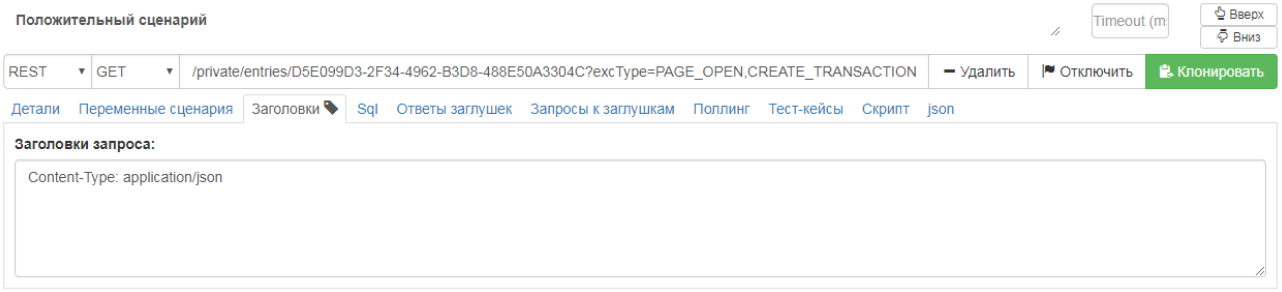
parameterName = $.element.items[2].title



При необходимости можно проверить сохраненное значение переменной. Шаг будет неуспешным, если ожидаемое значение и фактическое не совпадут. Проверять значение переменной можно на любом шаге.

5.2.1.3. Вкладка «Заголовки»

Вкладка используется для описания заголовков, которые будут переданы в запросе при прохождении шага.



5.2.1.4. Вкладка «SQL»

Блок используется для отправки SQL-запроса к базе данных. В текст запроса допустимо подставлять сохраненные ранее переменные.

Для работы должны быть указаны настройки БД тестируемого стенда в параметре *dataBase* файла env.yml.

В поле **«Сохраняемые значения»** описывается имя переменной, в которой сохраняется значение при выполнении запроса.

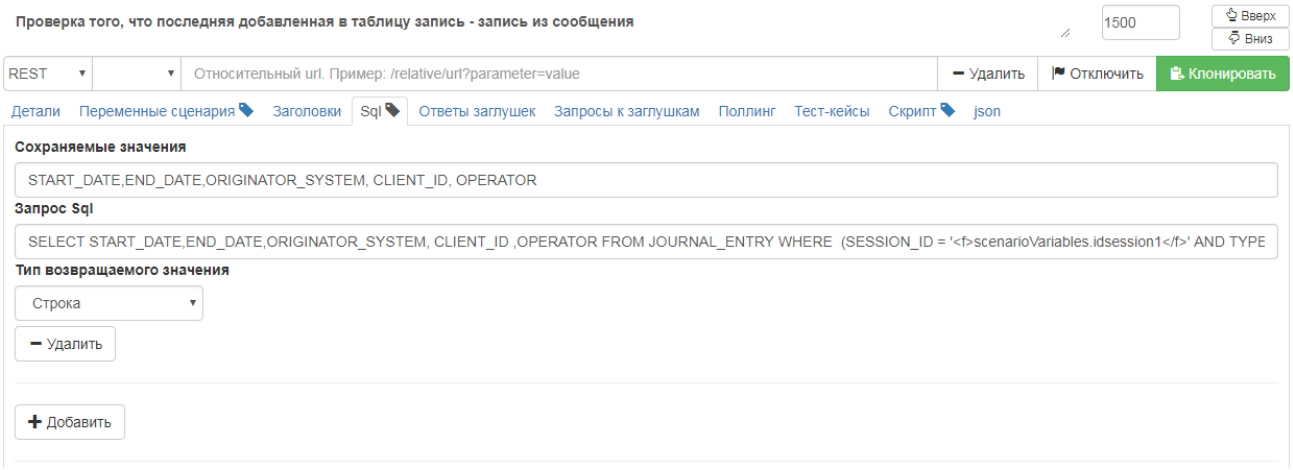
В поле «**Запрос Sql**» прописывается непосредственно сам запрос в БД.

В поле «**Тип возвращаемого значения**» выбирается тип переменной, в которой сохранено значение.

Есть несколько типов возвращаемых значений:

1. Объект *(возвращает первое значение первого параметра)*
2. Список *(возвращает множество значений первого параметра)*

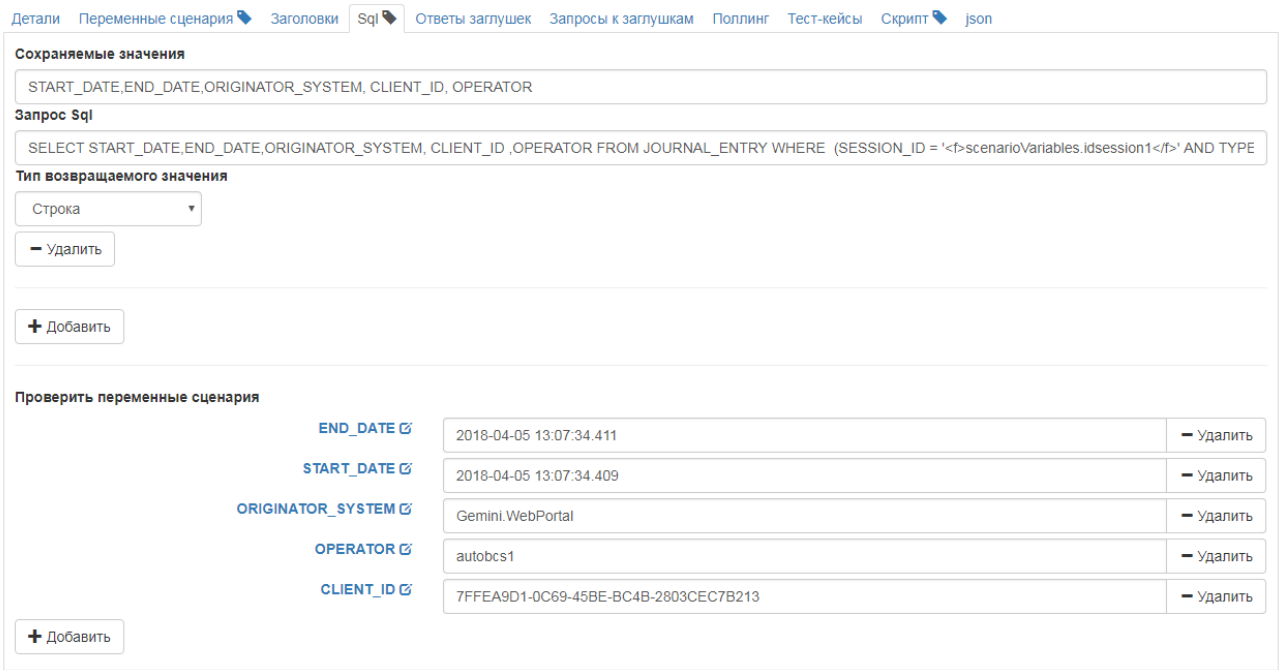
3. Map *(по умолчанию - возвращает одно или множество значений формата «поле – значение»)*
4. Строка *(возвращает множество значений одного или нескиольких параметров в формате строки, соответственно объявленным сохраняемым параметрам)*



Нельзя использовать запросы, меняющие данные в БД.

Пример использования переменной типа map в теле запроса: "eventDate": "<f>scenarioVariables.CALLDATE[0].CALLDATE</f>"

Так же в общем блоке проверяемых значений можно проверять значения сохраненных переменных в результате выполнения запросов:



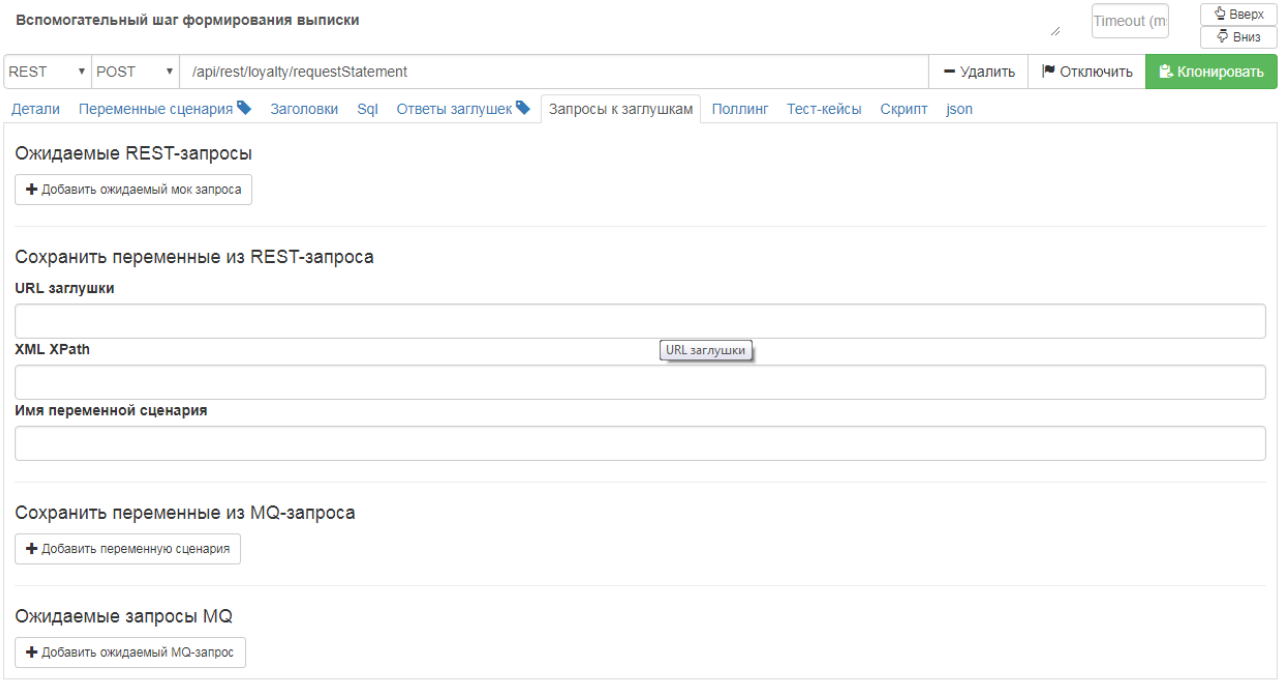
Значения сохраненных переменных проверяются в соответствии с форматом, в котором они были сохранены:

1. Объект – просто значение параметра
2. Список – значения параметра в формате *[значение1,значение2]*

3. Map – значения первого параметра в формате *[{параметр=значение1},{параметр=значение2}].*
4. Строка – значение параметра для каждого параметра запроса.

В одном шаге сценария можно сохранять несколько значений из БД, а так же осуществлять несколько проверок переменных. Сохраненные переменные могут использоваться в скрипте в рамках того же шага.

5.2.1.5. Вкладка «Запросы к заглушкам»



На данной вкладке возможно использовать нижеописанные варианты ожидаемых запросов к заглушкам.

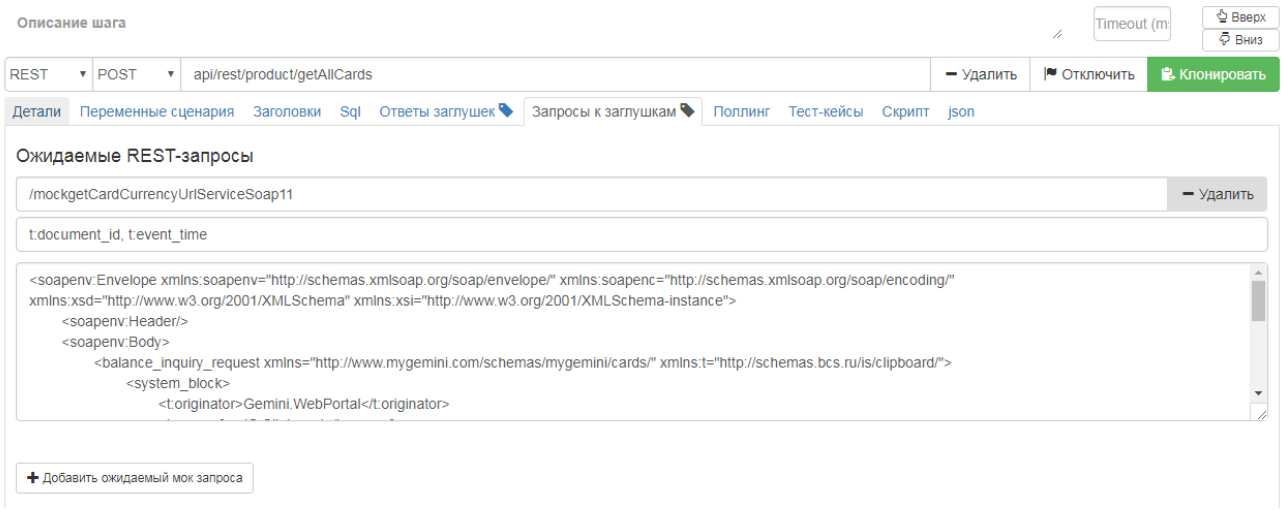
5.2.1.5.1. Ожидаемые REST-запросы

Блок используется для того, чтобы проверить вызов порталом сервиса по прописанной маске.

В **«Названии сервиса»** указывается название вызываемого сервиса.

В поле **«Ожидаемый запрос»** указывается сообщение, отправляемое порталом в формате xml.

При необходимости можно указать теги, содержание которых не будет приниматься во внимание, в поле «**Игнорируемые теги**».



Переменные из запроса можно сохранить в блоке **«Сохранить переменные из REST-запроса».**

В поле **«Mock URL»** указывается относительный url к сервису.

В поле **«XML XPath»** указывается путь к нужному полю в запросе, значение которого требуется сохранить.

В поле **«Scenario variable name»** указывается наименование создаваемой переменной, в которую будет сохранено значение поля, путь к которому указан в поле «XML XPath».

5.2.1.5.2. Ожидаемые запросы MQ

Вкладка используется для того, чтобы проверить обращение портала к очереди по прописанной маске.

В поле **«Source queue name»** указывается название очереди тестируемого сервиса, которое прописано в параметре *sourceQueueName* файла properties.yml.

В поле **«Ожидаемый запрос»** указывается сообщение, отправляемое порталом в формате xml. В сообщение можно подставлять сохраненные переменные сценария.

При необходимости можно указать теги, значения которых будут игнорироваться в поле «**Игнорируемые теги**».

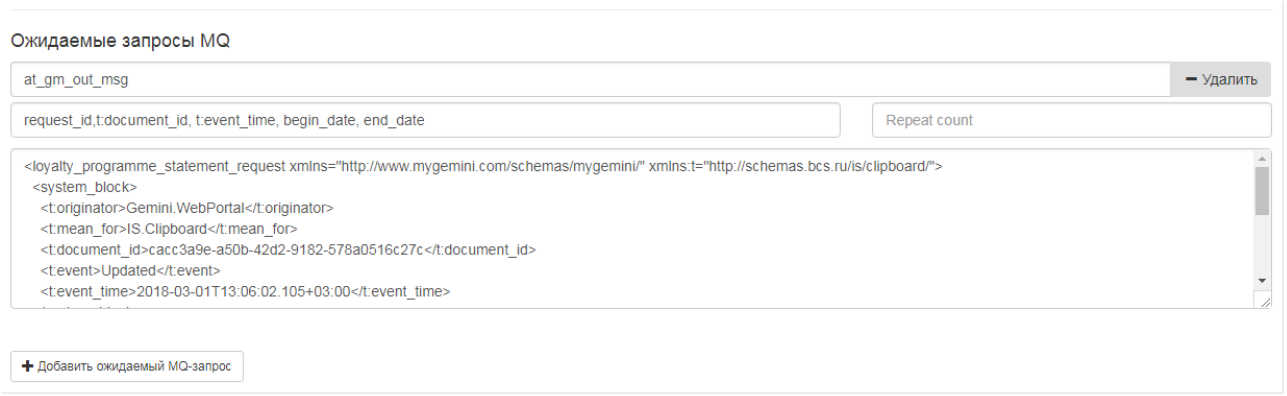
Так же на данной вкладке можно сохранять в переменные значения тегов из сообщения, отправляемого с портала в блоке «**Save to variables**».

В поле **«Source queue name»** указывается название очереди тестируемого сервиса, которое прописано в параметре *sourceQueueName* файла properties.yml.

В поле **«XPath»** указывается маска для выбора тега.

В поле «**Variable name**» указывается имя переменной, в которую сохраняется значение.

В поле «**Repeat count**» указывается ожидаемое число запросов в указанную очередь. Зачастую используется в связке с поле **«Number of repetitions»** на вкладке Детали.

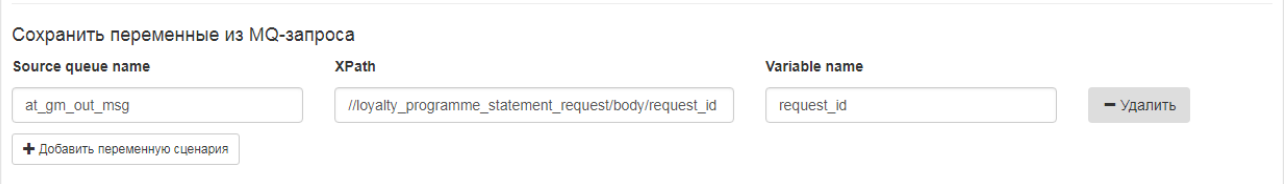


Переменные из запроса можно сохранить в блоке **«Сохранить переменные из MQ-запроса».**

В поле **«Source queue name»** указывается наименование очереди.

В поле **«XPath»** указывается путь к нужному полю в запросе, значение которого требуется сохранить.

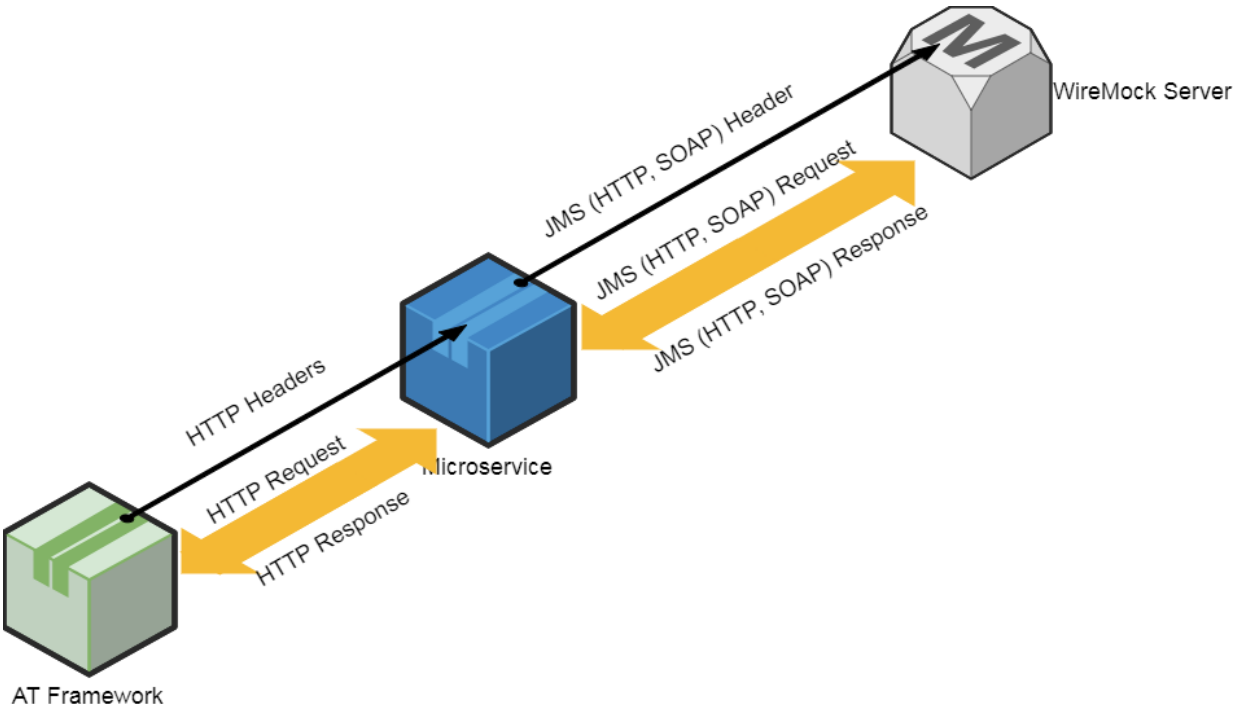
В поле **«Variable name»** указывается наименование создаваемой переменной.

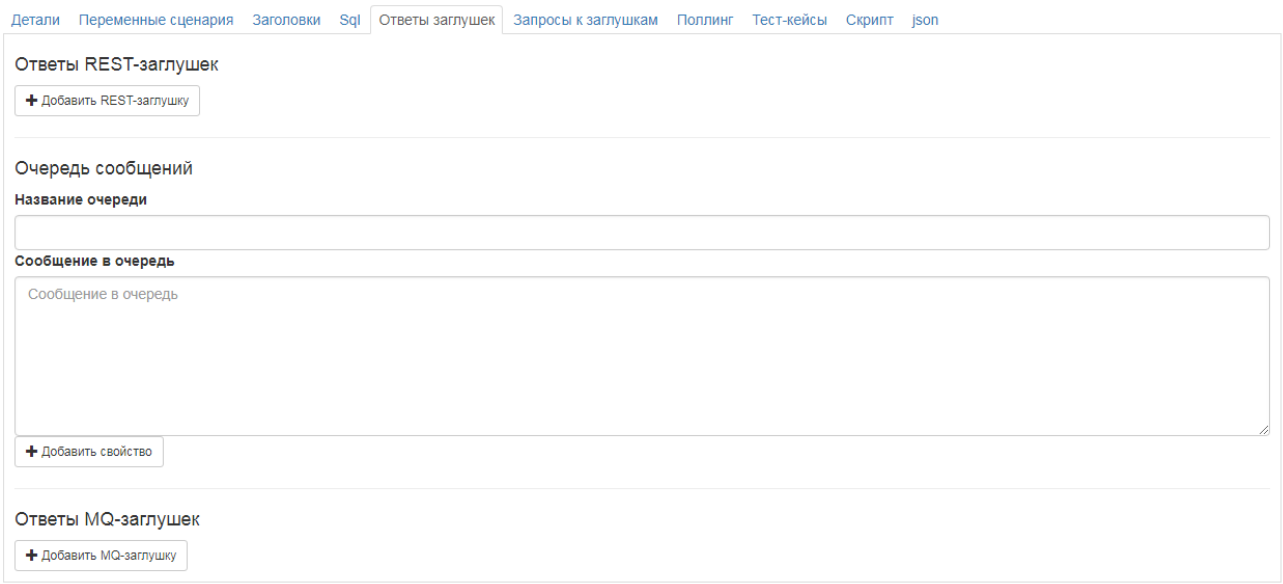


5.2.1.6. Вкладка «Ответы заглушек»

Для корректной работы ответов требуется включить параметр **«Использовать случайный testId»** в настройках проекта и указать название http-заголовка. Так же важно убедиться, что тестируемый портал отправляет http-заголовок в заглушки.

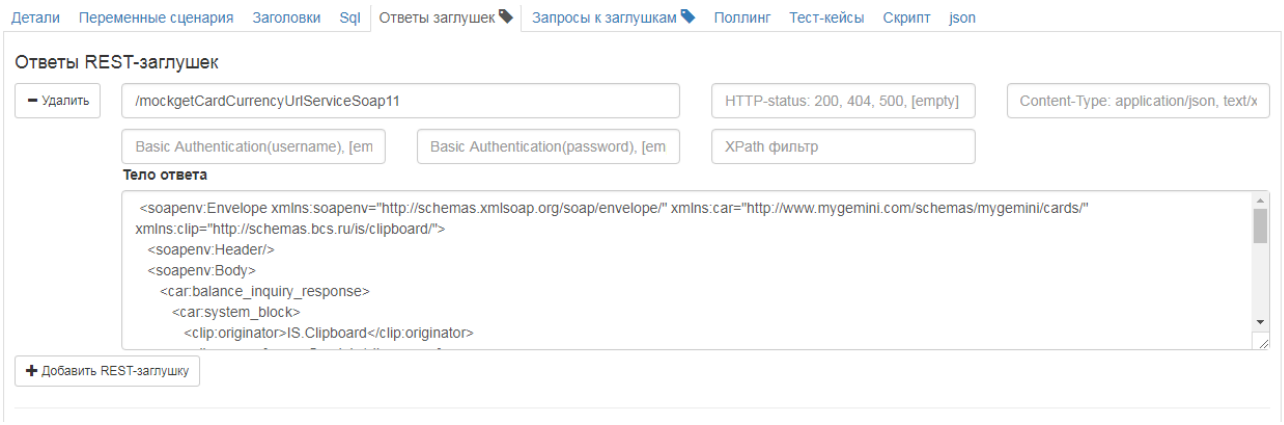
Используя случайный **testId** в указанном заголовке, можно создавать и использовать динамические заглушки. Это осуществляется следующим способом: wiremock создает по указанному урлу заглушку, привязывая ее к определенному значению **testId**, полученному от AuTe – Framework. При обращении микросервиса к wiremock передается заголовок с соответствующим **testI**, по которому wiremock определяет, какую заглушку следует использовать для ответа.





На данной вкладке есть возможность использования нижеописанных вариантов ожидаемых ответов заглушек.

5.2.1.6.1. Ответы REST-заглушек



Используется для указания ответа, который WireMock сервер (заглушка) возвращает на запросы портала.

В **«URL сервиса»** указывается url вызываемого сервиса. Если при обращении к сервису портал дописывает в URL какие-либо параметры, которые в тесте не нужны (например ip), их можно проигнорировать, указав маску, например:

/detectAddressLoan(.)*

Таким же образом можно указать URL в Wiremock, прописав его в поле **Url pattern**.

В поле **«Тело ответа»** указывается сообщение, возвращаемое заглушкой на запросы портала.

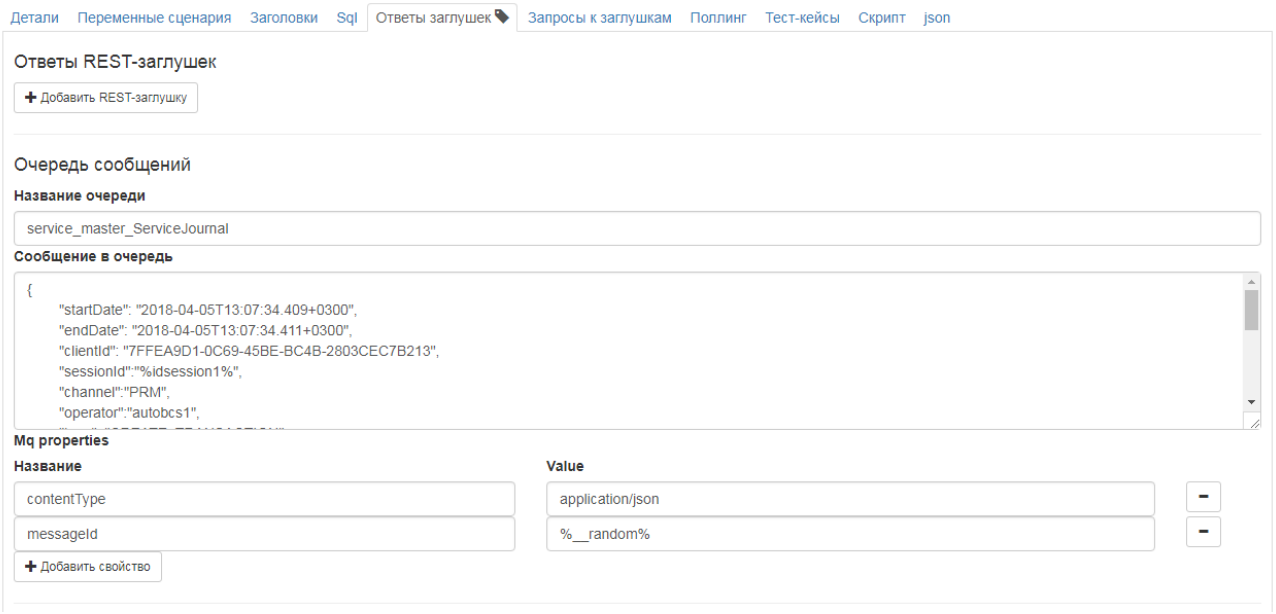
Можно, при необходимости, указать код ожидаемого статуса ответа *(например: 200, 404, 500)* в поле «**HTTP-status**», а так же Content-Type ответа в поле «**Content-Type**».

Так же существует возможность указания **Логина/Пароля** в случае, если для доступа к ответу WireMock требуется пройти basic-авторизацию.

5.2.1.6.2. Очередь сообщений

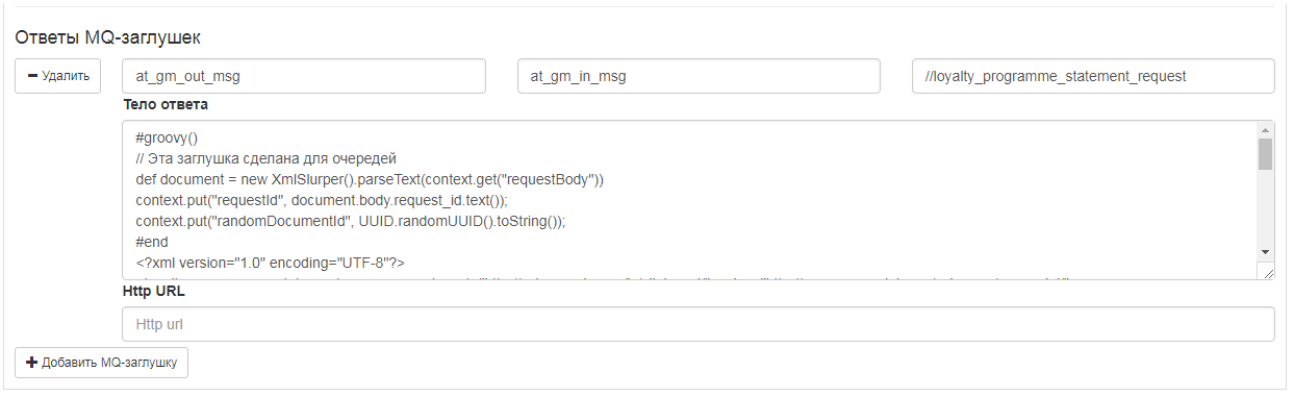
Требуется указать настройки подключения к серверу тестируемого стенда в параметре *amqpBroker* файла env.yml. На данной вкладке прописывается сообщение (тело сообщения прописывается в поле «**Сообщение в очередь**»), которое будет отправлено в очередь с указанным названием в поле «**Название очереди**» в ходе прохождения шага.

В AuTe Framework есть возможность указания property сообщения, которые будут отправлены в очередь вместе с телом сообщения. Для стандартных property необходимо указывать общепринятые наименования *(пример – contentType).* В качестве значений можно использовать сохраненные переменные в формате *%переменная%.* Для полей, где требуется уникальное значение *(например messageId)* можно использовать встроенную переменную **__random**.



5.2.1.6.3. Ответы MQ-заглушек

Блок используется для указания ответов, которые заглушка возвращает на запросы портала с использованием очередей.



В поле **«Source queue name»** указывается название очереди тестируемого сервиса, которое прописано в параметре *sourceQueueName* файла properties.yml. Это очередь, с которой будет работать mq-mocker.

В поле **«Destination queue name»** указывается название очереди, которое прописано в параметре *destinationQueueName* файла properties.yml. Это очередь, в которую направляется результат.

В поле **«XPath фильтр»** указывается маска для фильтрации сообщений в очереди, если их пришло несколько.

В поле **«Тело ответа»** необходимо указать тело ответа, которое будет отправлено в указанную очередь.

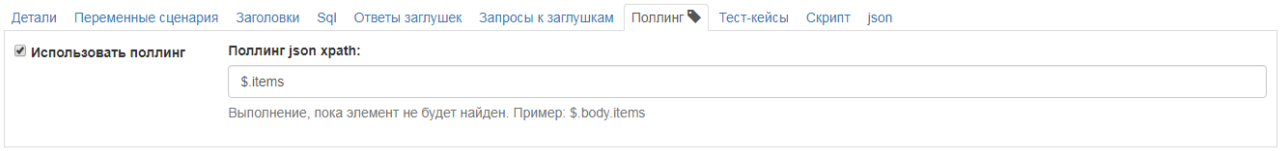
Если ответное сообщение в очередь будет отправлено в результате вызова сервиса, в поле **«Http URL»** указывается полный путь до нужного сервиса. Если значение данного поля указано, тело перехваченного сообщения (Ожидаемые запросы MQ) будет отправлено в POST запросе по указанному адресу, а ответ, полученный в результате запроса, будет использован как тело ответного сообщения.

Заполняется либо тело ответа, либо URL, в зависимости от архитектуры приложения.

5.2.1.7. Вкладка «Поллинг»

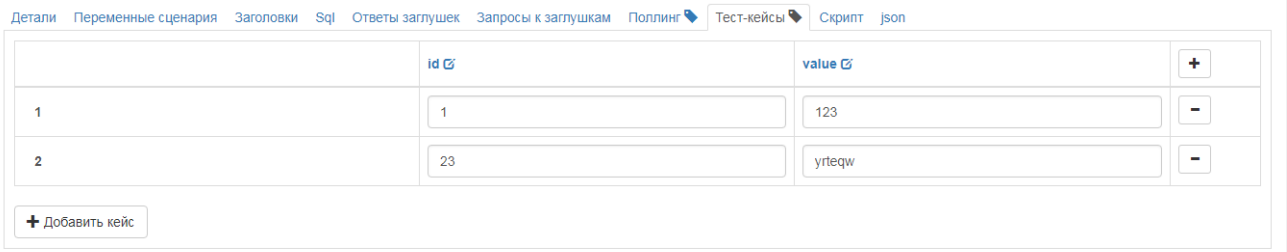
Используется для асинхронных ответов. Если чек-бокс **«Использовать поллинг»** включен, то запрос будет отправляться многократно до тех пор, пока указанный в поле «**Поллинг json xpath:**» JSON-параметр не будет найден. Для указания искомого параметра используется JSON XPath. Пример: $.body.items.

Запросы повторяются с периодом: 1 секунда, максимум: 50 раз.



5.2.1.8. Вкладка «Тест-кейсы»

Блок используется для указания переменных и их значений для многократного выполнения шага с разными наборами данных. В первой строке указываются наименования параметров, в последующих – значения этих параметров. В колонках указываются названия переменных, в строках - значения.



5.2.1.9. Вкладка «Скрипт»

В поле вода можно указать скрипт (используя javascript) для более гибкой работы AuTe Framework на данном шаге.

Пример:

```
if (scenarioVariables.GMW37.length > 0) {
 scenarioVariables.WEEKEND37 = 'true';
/* stepStatus.exception = 'Сегодня - выходной37';*/
 }
 else {
 scenarioVariables.WEEKEND37 = 'false';
/* stepStatus.exception = 'Сегодня - рабочий день37';*/
 }
```

Можно записать в переменные значения из заголовков тела ответа:

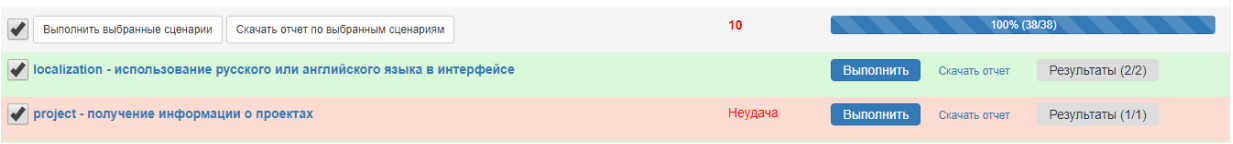
```
scenarioVariables.CONT = response.headers.get('Content-Type')[0];
scenarioVariables.DSCR = response.headers.get('Content-Disposition')[0];
```

5.2.1.10. Вкладка «JSON»

Вкладка содержит полное представление шага в формате JSON. Здесь можно просмотреть все параметры шага (в том числе и не заполненные, они будут со значением "null").

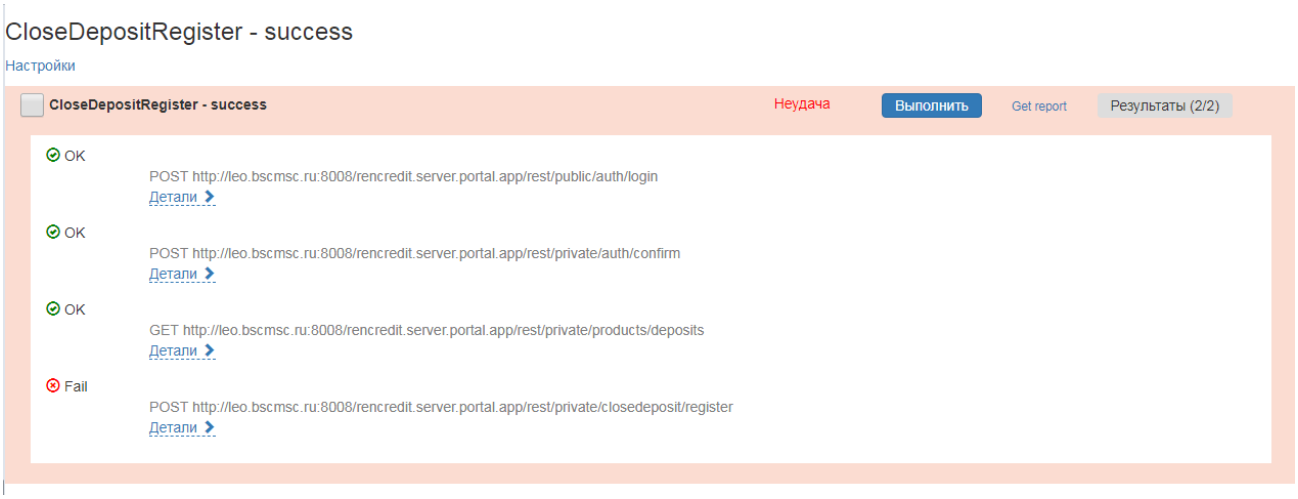
5.3. Просмотр результатов

5.3.1. Просмотр результатов через UI

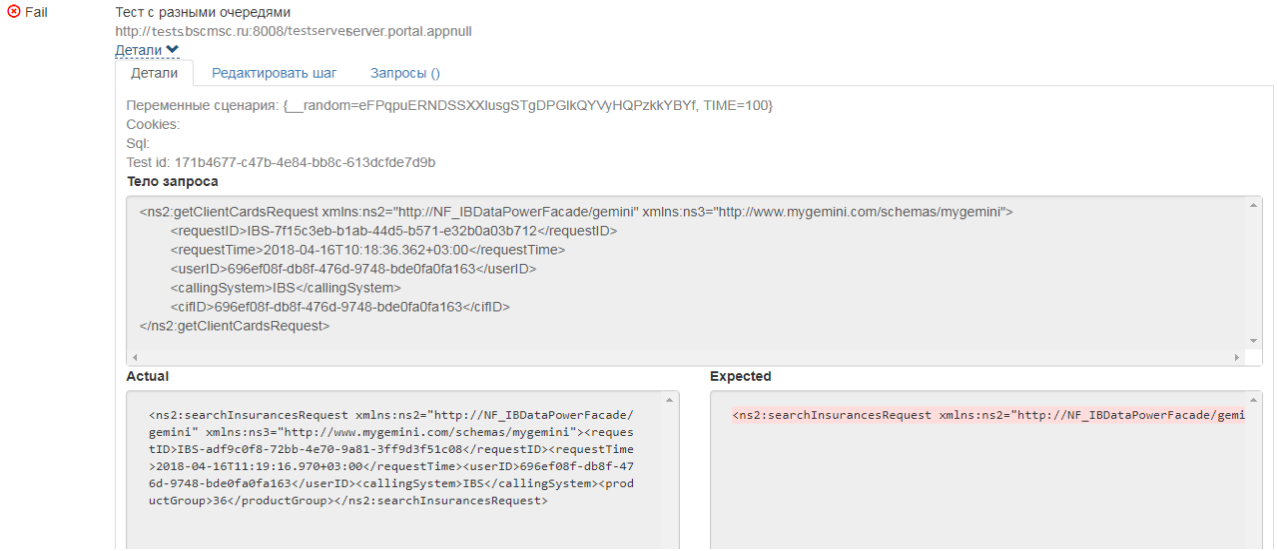


Чтобы просмотреть результаты через UI AuTe Framework, необходимо в строке теста кликнуть на кнопку Результаты. Открывшийся блок отображает результат предыдущего прогона теста. При нажатии «**Выполнить**» последовательно прогоняются шаги теста, добавляя те, которые в настройках проекта установлены как стартовый и конечный.

После выполнения сценария становится доступной возможность просмотра результатов прогона. При клике на кнопку «**Результаты**» раскрывается подложка с указанием всех шагов сценария. У каждого шага стоит статус его прогона.

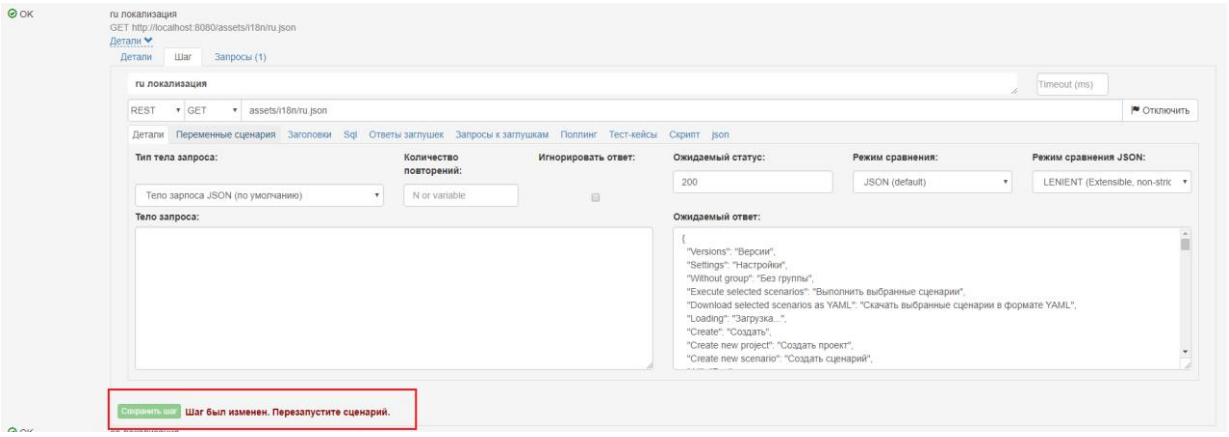


У каждого шага можно раскрыть блок «**Детали**», где указаны содержание запроса, переменные, ожидаемый и фактический результаты и детали ошибки, если она есть.

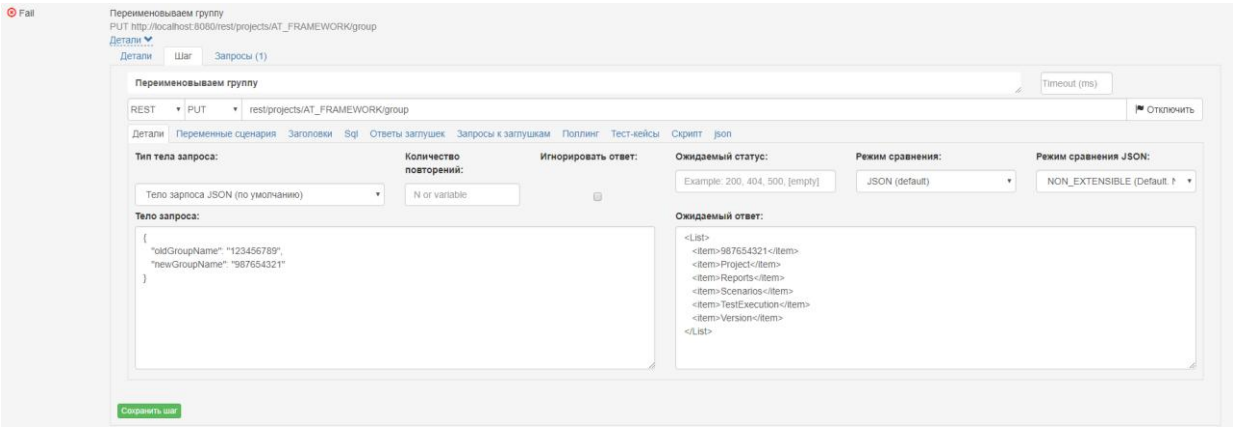


Для шагов, описанных в текущем сценарии, в блоке «**Детали**» доступна вкладка **«Шаг»**. Шаги, которые включены в сценарии и исполняются до и после текущего сценария согласно настройкам проекта, редактировать и просматривать здесь нельзя.

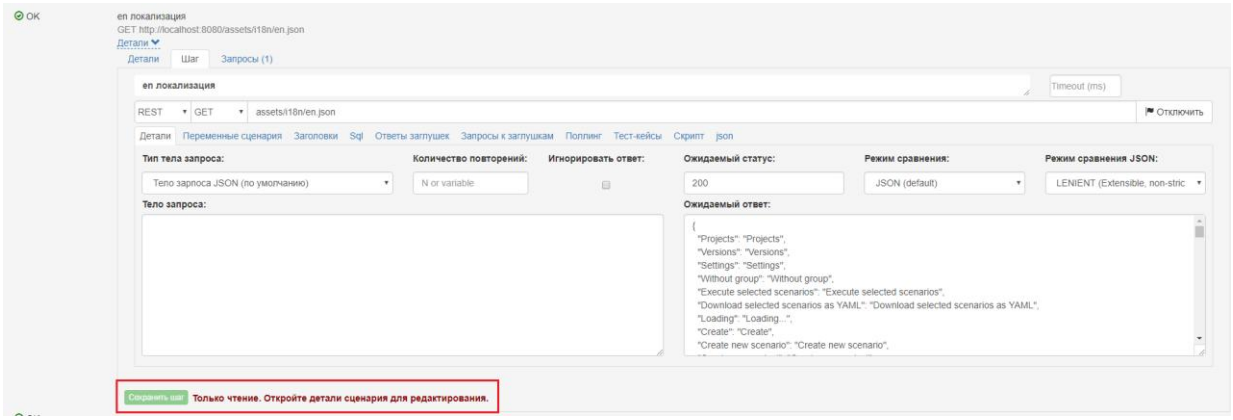
Вкладка содержит данные шага в том состоянии, в котором они находились во время последнего запуска сценария. Т.о. редактировать шаг посредством данной вкладки можно только на странице **списка шагов** *(детали сценария)* и при условии, что в основном блоке описания шага не производилось никаких изменений. Если данные изменялись, то для редактирования через форму **Результаты** нужно повторно запустить прогон сценария:



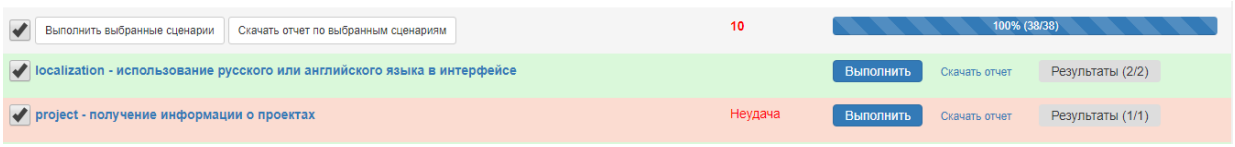
Если редактирование доступно, то все вносимые изменения в блоке **Результаты** сразу же переносятся в основные блоки редактирования шагов.



Со страницы **списка сценариев** редактирование во вкладке **«Шаг»** невозможно.



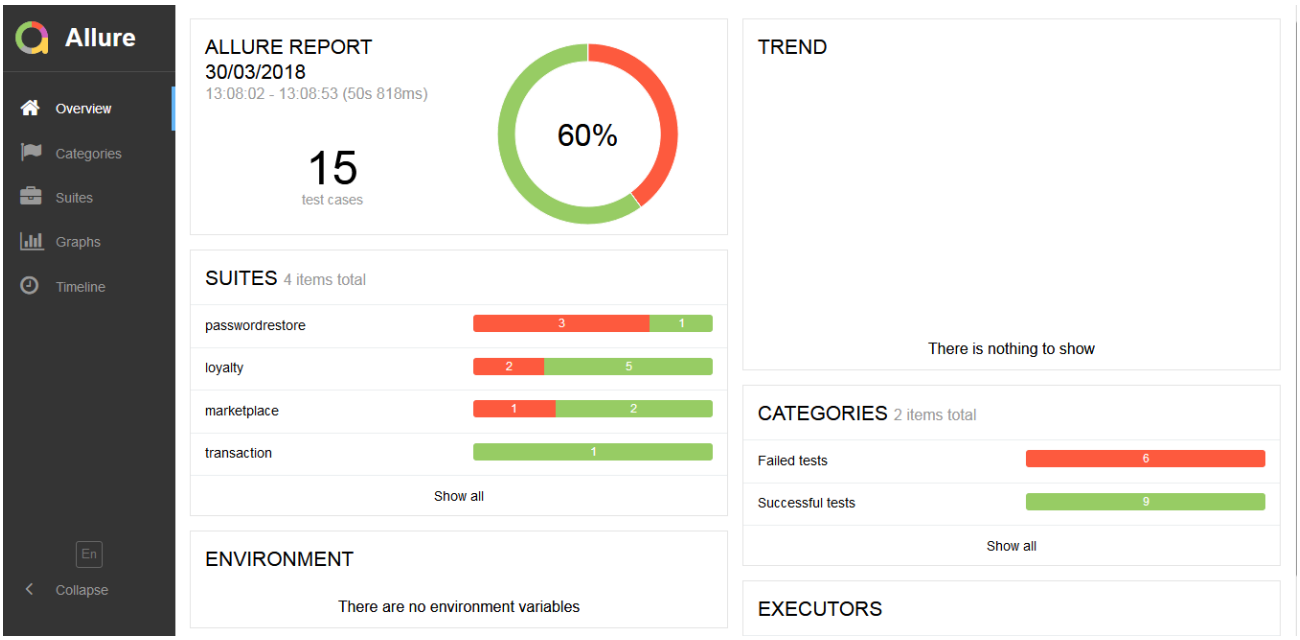
5.3.2. Экспорт отчёта



Отчет можно экспортировать двумя способами:

1) Кликнуть в строке сценария на ссылку “Скачать отчет”
2) Выделить тесты (отметив слева в чекбоксе галочкой), по которым необходимо получить отчет, а затем нажать на кноку “Скачать отчет по выбранным сценариям”

Отчет выгружается архивом, который содержит файлы отчетности с полной информацией о сценарии. Архив представляет собой статичный сайт и открывается с помощью браузеров Mozilla Firefox или Microsoft Edge:

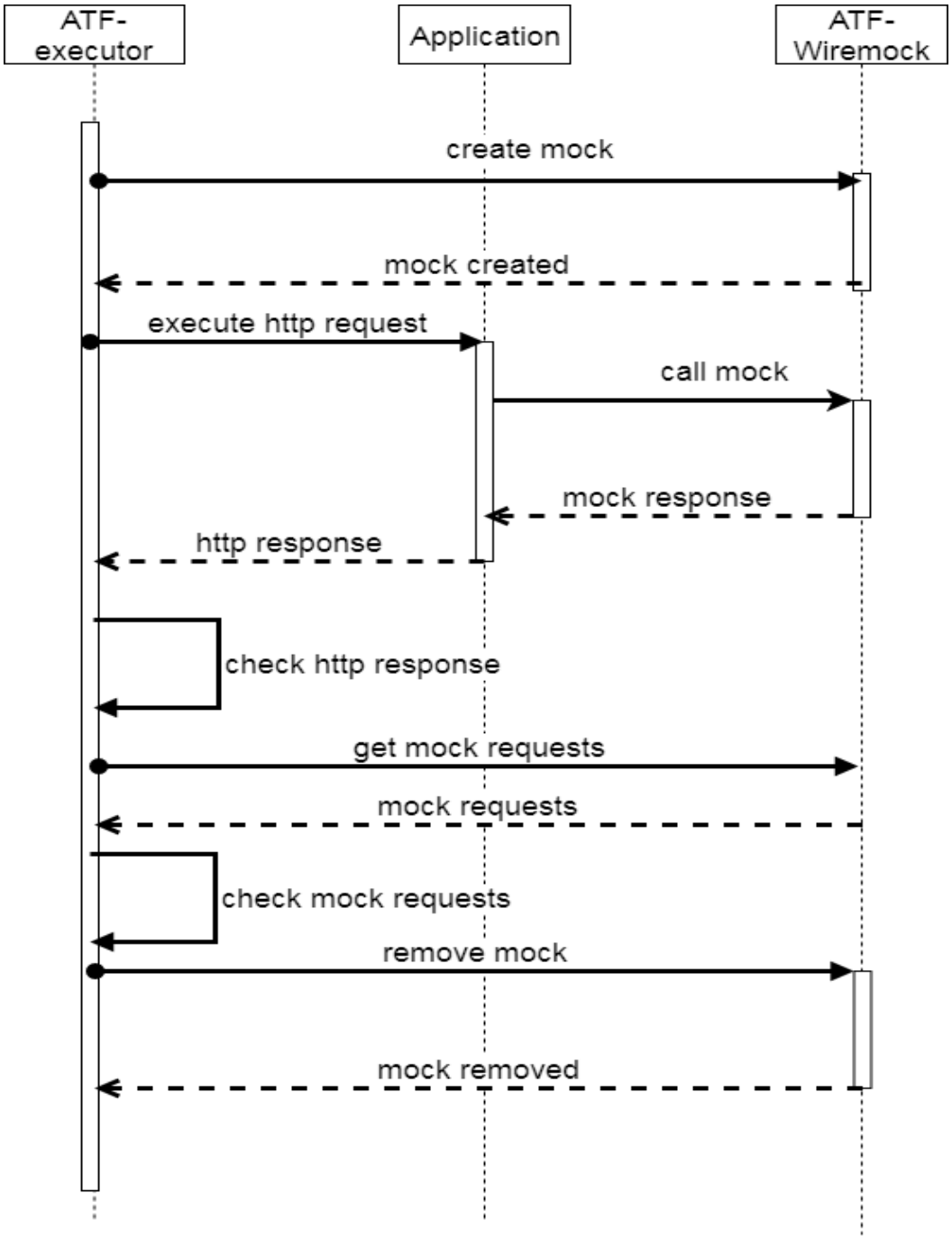


При запуске тестов через Jenkins отчет формируется в консоли после успешного запуска сборки.

6.Описание механизмов мокирования

6.1. Механизм мокирования HTTP-запросов

Принцип мокирования http-запросов представлен в виде диаграммы последовательностей, на рисунке ниже.



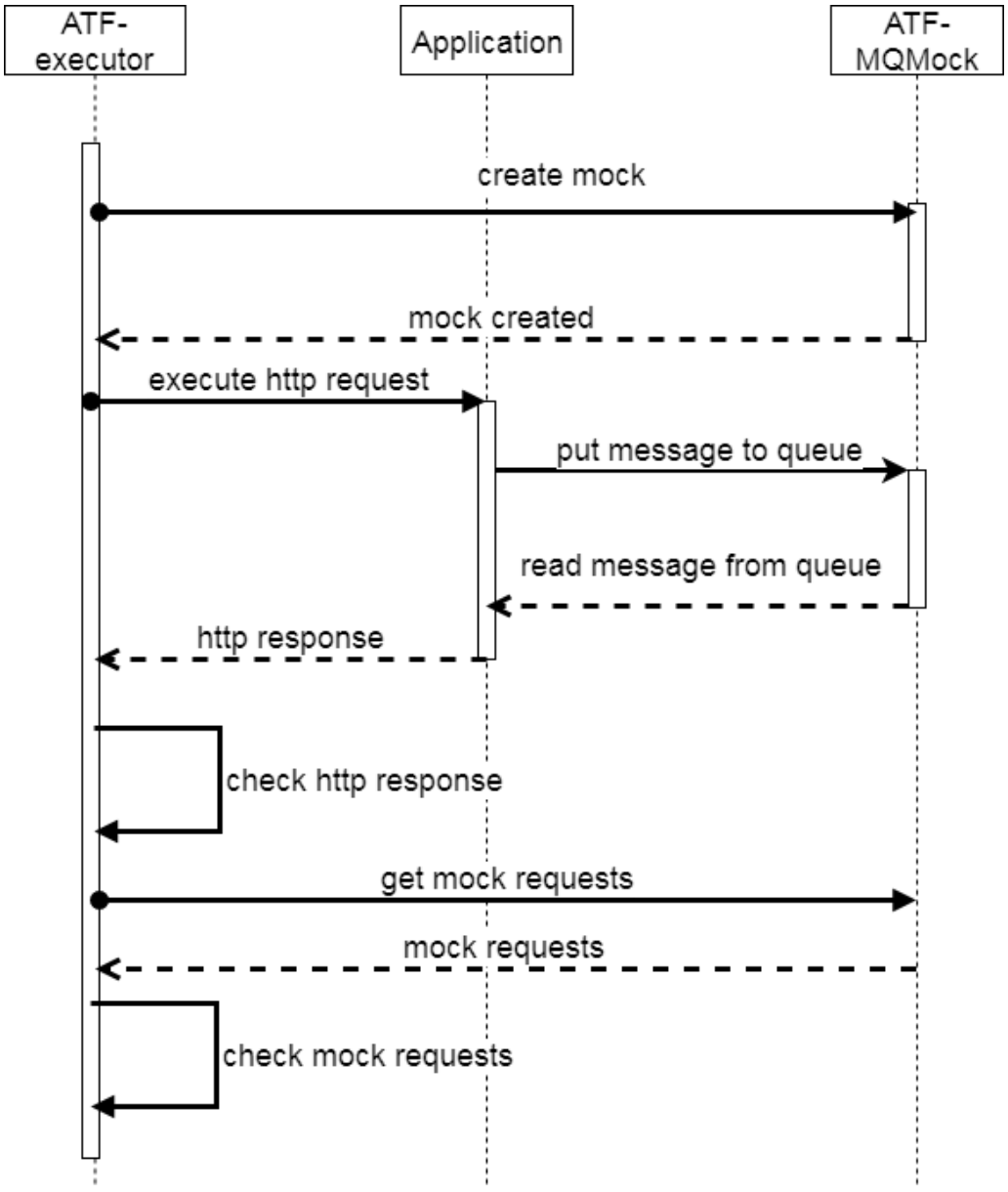
Легенда:

- **create mock** - создание уникальной заглушки для выполняемого теста;
- **call mock** - вызов заглушки, созданной на шаге 1 по http. Возврат Mock-ответа;

- **get mock requests** - получение запросов к заглушкам;
- **check mock requests** - проверка запросов к заглушкам (проверка того, что сформированный тестируемым приложением запрос соответствует ожидаемому);
- **remove mock** - удаление заглушки по завершению теста.

6.2. Механизм мокирования MQ-вызовов

Принцип мокирования MQ-вызовов представлен в виде диаграммы последовательностей, на рисунке ниже.



Легенда:

- **create mock -** создание уникальной заглушки для выполняемого теста;
- **put message to queue -** передача JMS-сообщения в очередь заглушки;
- **read message from queue -** чтение JMS-сообщения из очереди заглушки;
- **check mock requests -** проверка запросов к заглушкам (проверка того, что сформированный тестируемым приложением запрос соответствует ожидаемому).