GeekBrains

Тестирование Backend на Java

Автоматизированное тестирование REST API с использованием rest-assured

На этом уроке

1. Научимся писать первые тесты с библиотекой rest-assured.
2. Вспомним основные паттерны проектирования тестов.
3. Настроим логирование и отчётность.

Оглавление

Введение

В прошлых уроках мы рассмотрели, как тестировать SOAP и RESTful-сервисы, используя графические клиенты. Это хороший способ, если автоматизацию требуется сделать быстро. Однако приходится жертвовать гибкостью тестов и часто тратить время на их обновление. В этом уроке мы начнём автоматизировать проверки для REST API на языке Java с использованием rest-assured: соберём проект, используя Maven. Вспомним, как писать тесты на JUnit 5 и познакомимся с проверками rest-assured во fluent-стиле.

Автоматизация с использованием rest-assured

На этом уроке мы продолжим работать с сервисом хранения изображений и видео. Коллекция запросов для Postman с документацией лежит по этой ссылке.

Загрузка библиотеки

Воспользуемся репозиторием Apache Maven для подключения библиотеки и добавим зависимости в наш pom.xml:

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>4.3.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.7.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.6</version>
    </dependency>
</dependencies>
```

Кроме библиотеки rest-assured, добавим тестовый фреймворк JUnit 5 и библиотеку Commons IO, которая содержит полезные методы для работы с файлами.

О структуре тестов

Любой автотест включает в себя три компонента:

1. Подготовка данных для тестирования.
2. Сама проверка (тест).
3. Удаление тестовых данных.

В курсе мы будем использовать JUnit 5, но сама библиотека rest-assured работает с любыми тестовыми фреймворками.

Разберём базовые особенности API-автотеста на примере GET-запроса.

```
public class RecipeTests {
    @Test
    void getRecipePositiveTest() {
        given()
                .when()
                .get("https://api.spoonacular.com/recipes/716429/information?" +
"includeNutrition=false&apiKey=2e6cc9c807c84d0ba3518045b86e6687")
                .then()
                .statusCode(200);
    }
```

Теперь посмотрим, как можно сделать запрос с авторизацией (сейчас она является частью URI). В документации указано, что для авторизации необходимо использовать query параметр apiKey, так и напишем:

```
// выделяем query parameters

@Test
void getRecipeWithQueryParametersPositiveTest() {
   given()
           .queryParam("apiKey", "2e6cc9c807c84d0ba3518045b86e6687")
           .queryParam("includeNutrition", "false")
           .when()
           .get("https://api.spoonacular.com/recipes/716429/information")
           .then()
           .statusCode(200);
}
```

Также мы выделили в query параметр includeNutrition со значением false.

Рассмотрим тест подробнее. Он состоит из трёх блоков, аналогичным шагам в BDD-подходе:

1. Given, в котором пишутся предусловия.

2. When, в котором пишутся требуемые шаги.

3. Then, в котором пишутся проверки (assertions).

Библиотека rest-assured поддерживает fluent-стиль написания тестов, что делает их ещё более читаемыми.

Если тесты требуют создания каких-либо данных, после выполнения теста данные надо удалить. Рассмотрим этот подход на примере добавления блюда в MealPlan:

```
  String id;

@Test
void addMealTest() {
   id = given()
           .queryParam("hash", "a3da66460bfb7e62ea1c96cfa0b7a634a346ccbf")
           .queryParam("apiKey", "3e6cc9c807c84d0ba3518045b86e6687")
           .body("{\n"
                   + "    \"date\": 1644881179,\n"
                   + "    \"slot\": 1,\n"
                   + "    \"position\": 0,\n"
                   + "    \"type\": \"INGREDIENTS\",\n"
                   + "    \"value\": {\n"
                   + "        \"ingredients\": [\n"
                   + "            {\n"
                   + "                \"name\": \"1 banana\"\n"
                   + "            }\n"
                   + "        ]\n"
                   + "    }\n"
                   + "}")
           .when()
           .post("https://api.spoonacular.com/mealplanner/geekbrains/items")
           .then()
           .statusCode(200)
           .extract()
           .jsonPath()
           .get("id")
           .toString();
}

@AfterEach
void tearDown() {
   given()
           .queryParam("hash", "a3da66460bfb7e62ea1c96cfa0b7a634a346ccbf")
           .queryParam("apiKey", "3e6cc9c807c84d0ba3518045b86e6687")
           .delete("https://api.spoonacular.com/mealplanner/geekbrains/items/" + id)
           .then()
           .statusCode(200);
}
```

Теперь разберём сами тесты.

Основные методы

Любой тест начинается со слов given(), если есть какие-либо предусловия, или when(). Затем через точку идёт перечисление методов в цепочке.

Рассмотрим первый тест на получение информации о рецепте:

```
    @Test
    void getRecipePositiveTest() {
        given()
                .when()
                .get("https://api.spoonacular.com/recipes/716429/information" +
"?includeNutrition=false&apiKey=3e6cc9c807c84d0ba3518045b86e6687")
                .then()
                .statusCode(200);
    }
```

1. given() позволяет сформировать запрос с конкретными заголовками, телом, query параметрами.
2. when().get(url) позволяет отправить GET-запрос по указанному URL. Кроме get(url), используются методы, которые называются так же, как и методы HTTP — put(), post(), delete() и прочие.

В нашем сервисе используются query параметры для передачи токена авторизации и дополнительных параметров. Мы передаем их в given(), как было описано выше:

```
// выделяем query parameters

@Test
void getRecipeWithQueryParametersPositiveTest() {
   given()
           .queryParam("apiKey", "2e6cc9c807c84d0ba3518045b86e6687")
           .queryParam("includeNutrition", "false")
           .when()
           .get("https://api.spoonacular.com/recipes/716429/information")
           .then()
           .statusCode(200);
}
```

Проверки в тестах

Обычно проверки в тестах пишутся после выполнения всех шагов, rest-assured позволяет делать это, используя встроенные методы, например, как проверка выше then().statusCode(200)), а также

встроенную библиотеку ассертов Hamcrest. Для её использования требуется сохранить какое-либо значение из ответа на запрос в переменную, а затем использовать метод assertThat:

```
 @Test
    void getRecipeWithBodyChecksAfterRequestPositiveTest() {
        JsonPath response = given()
                .queryParam("apiKey", apiKey)
                .queryParam("includeNutrition", "false")
                .when()
                .get("https://api.spoonacular.com/recipes/716429/information")
                .body()
                .jsonPath();
        assertThat(response.get("vegetarian"), is(false));
        assertThat(response.get("vegan"), is(false));
        assertThat(response.get("license"), equalTo("CC BY-SA 3.0"));
        assertThat(response.get("pricePerServing"), equalTo(163.15F));
        assertThat(response.get("extendedIngredients[0].aisle"), equalTo("Milk, Eggs,
Other Dairy"));
    }
```

Здесь через метод body().jsonPath() мы распарсили JSON-ответ сервера, а затем взяли значение url, которое лежит внутри обёртки data. Путь пишется аналогично тестам в Postman.

Сохранив в переменную значение response, мы сравниваем его с ожидаемым значением в строке assertThat(response.get(jsonPath), equalTo(значение)));. Для булевых значений можно использовать is вместо equalTo, например,

```
  assertThat(response.get("vegetarian"), is(false));
```

Подробнее о библиотеке Hamcrest — здесь (на русском).

Ещё один способ, который предоставляет библиотека rest-assured — писать проверки в блоке given(), где мы пишем «ожидания» от выполнения запроса:

```
// тесты в проверками в Expect
@Test
void getRecipeWithBodyChecksInGivenPositiveTest() {
   given()
           .queryParam("apiKey", apiKey)
           .queryParam("includeNutrition", "false")
           .expect()
           .body("vegetarian", is(false))
           .body("vegan", is(false))
           .body("license", equalTo("CC BY-SA 3.0"))
           .body("pricePerServing", equalTo(163.15F))
           .body("extendedIngredients[0].aisle", equalTo("Milk, Eggs, Other Dairy"))
           .when()
           .get("https://api.spoonacular.com/recipes/716429/information")
```

```
            .then()
            .statusCode(200);
}
```

Здесь мы указываем, что требуется проверить поля vegetarian и vegan и другие в теле (body) ответа на равенство определённым значениям.

В обоих примерах мы делали проверки после then() и проверяли еще и statusCode.

Логирование в тестах

Rest-assured предоставляет настройку для логирования «упавших» тестов:

```
@BeforeAll
static void setUp() {
   RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
}
```

Если надо логировать все ответы, что полезно, то лучше это делать в самом запросе через методы log() и prettyPeek(). Особенность метода log() — можно выбрать, что лучше логировать:

```
    //добавляем логирование
    @Test
    void getRecipeWithLoggingPositiveTest() {
        given()
                .log()
                .all()
                .when()
                .get("https://api.spoonacular.com/recipes/716429/information")
                .prettyPeek()
                .then()
                .statusCode(200);
    }
```

log().all() логирует всё связанное с реквестом, а prettyPeek() — то, что связано с ответом. Для логирования ответа ещё используются print() и prettyPrint(), но они возвращают строку, соответственно, дальнейшие действия с объектом невозможны.

Логирование запросов и ответов в Allure-отчётах

Для интеграции с Allure используется специальная библиотека allure-rest-assured. Её последнюю версию можно найти на странице библиотеки в Maven-репозитории.

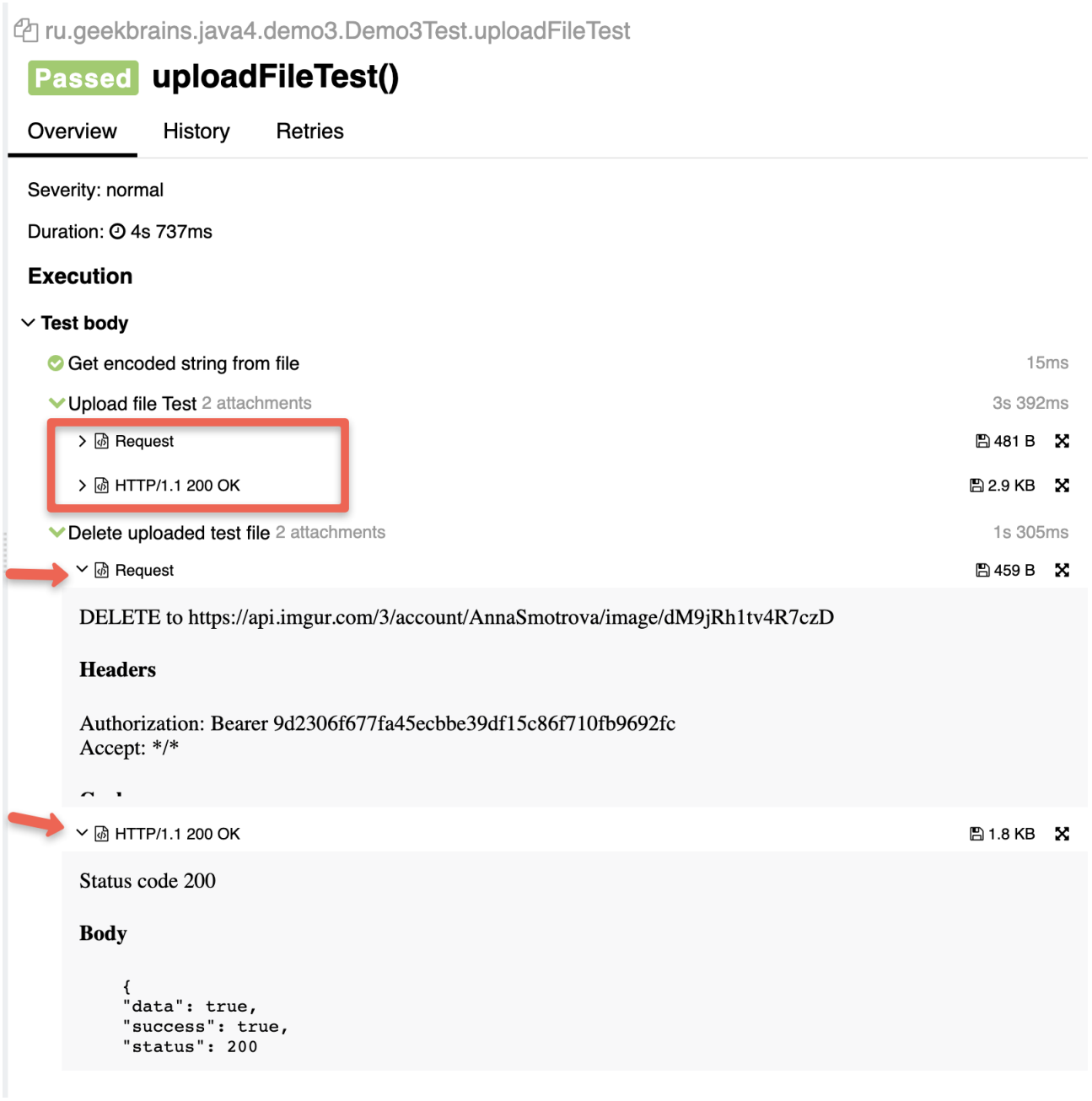
```
<dependency>
```

```
    <groupId>io.qameta.allure</groupId>
    <artifactId>allure-rest-assured</artifactId>
    <version>2.13.8</version>
</dependency>
```

В методе с аннотацией @BeforeAll прописываем специальную строчку, которая включает логирование:

```
    @BeforeAll
    static void beforeAll() {
//for logging request and responses in Allure reporting
        RestAssured.filters(new AllureRestAssured());
    }
```

После настройки Allure-отчётов и прогона тестов через mvn clean test allure:serve можно увидеть результаты прогона в следующем виде (вид со страницы Suites):



Практическое задание

Работа со Spoonacular API

1. Автоматизируйте GET /recepies/complexSearch (минимум 5 кейсов) и POST /recipes/cuisine (минимум 5 кейсов), используя rest-assured.
2. Сделать автоматизацию цепочки (хотя бы 1 тест со всеми эндпоинтами) для создания и удаления блюда в MealPlan). Подумайте, как использовать tearDown при тестировании POST /mealplanner/:username/shopping-list/items

Воспользуйтесь кейсами, которые вы написали в ДЗ №2, перенеся всю логику из постман-коллекции в код.

Сдайте ссылку на репозиторий, указав ветку с кодом.

Дополнительные материалы

1. Официальная документация.
2. Статья A Guide to REST-assured.
3. Ссылка на автоматизированную коллекцию Postman с цепочкой Shopping list

Используемые источники

1. Alan Richardson. Automating and Testing a REST API.
2. Официальная документация.
3. Статья REST Assured Tutorial: How to test API with Example.