

# Тестовое задание:

Срок выполнения - 7 дней (само задание делается за 3 дня, но мы отталкиваемся от того что кандидат может иметь стороннюю загрузку, либо немного иной стек).

## Цель

Напишите проект автотестов для сервиса <https://petstore.swagger.io/>

1. валидационные тесты на контроллеры (объем определяется исполнителем)
2. пользовательские сценарии (объем определяется исполнителем)

## Условия к заданию:

1. базовый сборщик - gradle (Использую maven он мне роднее)
2. базовый фреймворк запуска тестов -TestNg (использую junit5 он круче и не сложнее)
3. взаимодействие с тестируемым сервисом происходит с помощью RestAssured
4. в проекте используется Allure Framework для генерации отчетов
5. автотесты упакованы в Docker с использованием Docker Compose и переменных окружения, которые должны быть описаны в отдельном файле
6. файл README должен содержать подробное описание по работе с проектом (смотреть PDF или лучше через IDE с плагином asciidoc позволяет легко найти все файлы из README.adoc)
7. все создаваемые записи в сервисе должны удаляться после прогона тестов (в работе, но тесты пока работают последовательно и последний тест чистит работу предыдущих)
8. передача проекта осуществляется через ссылку на git репозиторий с проектом
9. будет плюсом разработка понятного консольного логирования (считаю это не комельфо, все должно быть понятно из отчета, тесты должны быть простыми и консольное логирование нужно для сложных моментов)

## Описание пакетов и классов

1. endpoints - содержит классы с методами настройки запросов (заполнение значениями тел запросов) и хранит данные запроса
2. src/test/java/tests - пакет содержит функциональные тесты по проверке endpoint
3. src/test/java/userscases - пакет содержит функциональные тесты по проверке пользовательских сценариев

4. src/test/java/endpoints - пакет содержит классы по работе с endpoint (PageObject на сервис)
5. Sender - класс который занимается отправкой запросов согласно методу протокола HTTP/HTTPS (GET, POST, PUT, DELETE)
6. Endpoint - класс который хранит необходимую информацию для отправки проверки и настройки запросов (Путь endpoint, заголовки, пути к файлам с телами запросов и схем валидации ответов)

## Класс Sender

Класс содержит статические методы для отправки CRUD запроса с разной конфигурацией

```
public class Sender {
    public static RequestSpecification sessionAndContentTypeJson(String sessionCookie){
        return new RequestSpecBuilder()
            .addCookie("api_key", sessionCookie)
            .setContentType(ContentType.JSON)
            .build();
    }
    @Step("GET запрос - {path} ожидаем статус ответа - {statusCode}")
    public static Response step_Get(RequestSpecification requestSpecification, Map
    <String, ?> header, String path, String responseSchema, int statusCode) {
        return given()
            .redirects().follow(false)
            .spec(requestSpecification)
            .when()
            .log().all()
            .headers(header)
            .get(path)
            .then()
            .statusCode(statusCode)
            .body(matchesJsonSchema(new File(responseSchema)))
            .extract().response();
    }
    @Step("GET запрос - {path} ожидаем статус ответа - {statusCode}")
    public static Response step_Get(RequestSpecification requestSpecification, Map
    <String, ?> header, String path, String nameParam, Object valueParam, String
    responseSchema, int statusCode) {
        return given()
            .redirects().follow(false)
            .spec(requestSpecification)
            .queryParams(nameParam, valueParam)
            .when()
            .log().all()
            .headers(header)
            .get(path)
            .then()
            .statusCode(statusCode)
            .body(matchesJsonSchema(new File(responseSchema)))
            .extract().response();
    }
}
```

```

}
@Step("POST запрос - {path} ожидаем статус ответа - {statusCode}")
public static Response step_Post(RequestSpecification requestSpecification, Map
<String, ?> header, String path, Object queryJson, String responseSchema, int
statusCode) {
    return given()
        .redirects().follow(false)
        .spec(requestSpecification)
        .when()
        .log().all()
        .headers(header)
        .body(queryJson)
        .post(path)
        .then()
        .log().all()
        .statusCode(statusCode)
        .body(matchesJsonSchema(new File(responseSchema)))
        .extract().response();
}

@Step("POST запрос - {path} с файлом и параметрами загрузки, ожидаем статус ответа -
{statusCode}")
public static Response step_Post(RequestSpecification requestSpecification, Map
<String, ?> header, Map<String, ?> formParam, String path, String responseSchema, int
statusCode) {
    return given()
        .redirects().follow(false)
        .spec(requestSpecification)
        .when()
        .log().all()
        .headers(header)
        .formParams(formParam)
        .post(path)
        .then()
        .log().all()
        .statusCode(statusCode)
        .body(matchesJsonSchema(new File(responseSchema)))
        .extract().response();
}

@Step("POST запрос - {path} с файлом и параметрами загрузки, ожидаем статус ответа -
{statusCode}")
public static Response step_Post(RequestSpecification requestSpecification, Map
<String, ?> header, String path, Map<String, ?> formParam, String nameParamFile, File
file, String filetype, String responseSchema, int statusCode) {
    return given()
        .redirects().follow(false)
        .spec(requestSpecification)
        .when()
        .log().all()
        .headers(header)
        .formParams(formParam)
        .multiPart(nameParamFile, file, filetype)

```

```

        .post(path)
        .then()
        .log().all()
        .statusCode(statusCode)
        .body(matchesJsonSchema(new File(responseSchema)))
        .extract().response();
    }
    @Step("PUT запрос - {path} ожидаем статус ответа - {statusCode}")
    public static Response step_Put(RequestSpecification requestSpecification, String
path, Object queryJson, String responseSchema, int statusCode) {
        return given()
            .redirects().follow(false)
            .spec(requestSpecification)
            .when()
            .log().all()
            .body(queryJson)
            .put(path)
            .then()
            .log().all()
            .statusCode(statusCode)
            .body(matchesJsonSchema(new File(responseSchema)))
            .extract().response();
    }
    @Step("DELETE запрос - {path} с параметром {nameParam} = {valueParam} ожидаем
статус ответа - {statusCode}")
    public static Response step_Delete(RequestSpecification requestSpecification, String
path, String nameParam, Object valueParam, int statusCode) {
        return given()
            .redirects().follow(false)
            .spec(requestSpecification)
            .queryParam(nameParam, valueParam)
            .when()
            .log().all()
            .delete(path)
            .then()
            .log().all()
            .statusCode(statusCode)
            .extract().response();
    }
    @Step("DELETE запрос - {path} ожидаем статус ответа - {statusCode}")
    public static Response step_Delete(RequestSpecification requestSpecification, String
path, int statusCode) {
        return given()
            .redirects().follow(false)
            .spec(requestSpecification)
            .when()
            .log().all()
            .delete(path)
            .then()
            .log().all()
            .statusCode(statusCode)

```

```

        .extract().response();
    }
}

```

## Класс Endpoint

Содержит информацию о точке сервиса, собирается из XML файлов в которых описываем путь к rest методу Сервиса, параметры заголовка, так же содержит пути к шаблонам тел запросов и пути к json схемам валидации ответов

```

@Getter
@Setter
public class Endpoint {
    private Map<String, String> urls;
    private Map<String, String> headerParams;
    private Map<String, String> request;
    private Map<String, String> response;
}

```

## Класс Pet

Содержит свойства и методы для управления CRUD питомцами

Свойство в котором храниться информация о endpoint (путь, параметры заголовка, пути к шаблонам запросов и схем валидации ответа)

```

public class Pet {
    @Qualifier("endpointPet")
    @Autowired
    protected Endpoint endpoint;
}

```

Свойства для заполнения тела запроса:

```

private int status_code;
private int id;
private String name;
private String status;
private List<String> photoUrls;
private HashMap<String, String> category;
private HashMap<String, String> tags;
private String image;
private String additionalMetadata;

```

Методы:

```

@Step("Добавить нового питомца в хранилище")
public Response add() {
    Map<String, Object> queryJson = request("create");
    return step_Post(requestSpecification, endpoint.getHeaderParams(), endpoint
.getUrls().get("base"), queryJson, endpoint.getResponse().get("create_pass"),
status_code);
}
@Step("Обновить данные питомца")
public Response update() {
    Map<String, Object> queryJson = request("create");
    return step_Put(requestSpecification, endpoint.getUrls().get("base"), queryJson,
endpoint.getResponse().get("create_pass"), status_code);
}
@Step("Удалить питомца из хранилища")
public Response delete() {
    return step_Delete(requestSpecification, endpoint.getUrls().get("base") + "/" +
this.id, status_code);
}
@Step("Обновляем картинку питомца через загрузку файла")
public Response uploadImage() {
    Map<String, String> headers = endpoint.getHeaderParams();
    headers.replace("Content-Type", "multipart/form-data");
    Map<String, String> formParams = new HashMap<>();
    formParams.put("additionalMetadata", this.additionalMetadata);
    return step_Post(requestSpecification, headers, format(endpoint.getUrls().get(
"uploadAnImages"), this.id), formParams, "file", new File(this.image), "image/jpeg",
endpoint.getResponse().get("uploadAnImages"), status_code);
}
@Step("Найти всех питомцев по полю {string} с значением {string}")
public Response find(String name, String value) {
    return step_Get(requestSpecification, endpoint.getHeaderParams(), endpoint.
getUrls().get("findsByStatus"), name, value, endpoint.getResponse().get("
findsByStatus"),
status_code);
}
@Step("Найти питомца по id {string}")
public Response getPetById(String petId) {
    return step_Get(requestSpecification, endpoint.getHeaderParams(), format(endpoint
.getUrls().get("byPetId"), petId), endpoint.getResponse().get("create_pass"),
status_code);
}
@Step("Найти питомца по id {string}")
public Response updateWithFormData(String name, String status) {
    Map<String, String> headers = endpoint.getHeaderParams();
    headers.replace("Content-Type", "application/x-www-form-urlencoded");
    Map<String, String> formParams = new HashMap<>();
    formParams.put("name", name);
    formParams.put("status", status);
    return step_Post(requestSpecification, headers, formParams, format(endpoint
.getUrls().get("byPetId"), this.id), endpoint.getResponse().get("updateWithFormParam"

```

```

), status_code);
}
@NotNull
@Step("Заполняем запрос {request}")
private Map<String, Object> request(String request) {
    JsonPath jsonPath = new JsonPath(new File(endpoint.getRequest().get(request)));
    Map<String, Object> queryJson = jsonPath.getMap("$");
    queryJson.replace("id", this.id);
    queryJson.replace("name", this.name);
    queryJson.replace("status", this.status);
    queryJson.replace("photoUrls", this.photoUrls);
    queryJson.replace("category", this.category);
    queryJson.replace("tag", this.tags);
    return queryJson;
}
}

```

Xml описание bean endpoint свойства endpoint в классе Pet

```

<property name="urls">
    <map>
        <entry key="base" value="/v2/pet"/>
        <entry key="findsByStatus" value="/v2/pet/findByStatus"/>
        <entry key="byPetId" value="/v2/pet/%s"/>
        <entry key="uploadAnImages" value="v2/pet/%s/uploadImage"/>
    </map>
</property>
<property name="headerParams">
    <map>
        <entry key="accept" value="application/json"/>
        <entry key="Content-Type" value="application/json"/>
    </map>
</property>
<property name="request">
    <map>
        <entry key="create" value=
"src/test/resources/requests/add_a_new_pet_to_thestore.json"/>
    </map>
</property>
<property name="response">
    <map>
        <entry key="create_pass" value=
"src/test/resources/response/add_a_new_pet_to_thestore.json"/>
        <entry key="uploadAnImages" value=
"src/test/resources/response/upload_images.json"/>
        <entry key="findsByStatus" value=
"src/test/resources/response/findsByStatus.json"/>
        <entry key="updateWithFormParam" value=
"src/test/resources/response/update_with_forparam.json"/>
    </map>
</property>

```

```
</property>
</bean>
```

xml описание bean Pet для теста с положительным результатом

```
<bean name="pass_user" class="endpoints.User">
  <property name="status_code" value="200"/>
  <property name="id" value="100"/>
  <property name="username" value="UserTest1"/>
  <property name="firstName" value="Тестер"/>
  <property name="lastName" value="Тестовый"/>
  <property name="email" value="UserTest1@mal0.com"/>
  <property name="password" value="PaSSw0rd"/>
  <property name="phone" value="88008008080"/>
  <property name="userStatus" value="0"/>
</bean>
</beans>
```

## Тесты валидации

```
@Link("https://petstore.swagger.io/#/pet")
@Epic("Зоомагазин->Домашний питомец->валидационные тесты")
@Feature("Питомец")
@ExtendWith(SpringExtension.class)
@ContextConfiguration({"datapools/pet.xml", "endpoints/petstore.xml"})
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
//TODO: реализовать для генерации данных для валид тестов негативного характера
//@ExtendWith(SetupPet.class)
//@ExtendWith(CleanupPet.class)
public class PetTest {
  @Qualifier("pass")
  @Autowired
  Pet dog;
  static {
    RestAssured.baseURI = System.getenv("BASE_URL");
    RestAssured.requestSpecification = sessionAndContentTypeJson(System.getenv(
"API_KEY"));
  }
  @Test
  @Order(0)
  @DisplayName("Add a new pet to the store")
  void should_SuccessAddaNewPetToTheStore_CompareIdAndNameNewDogFrenki(){
    Response response = dog.add();
    JsonPath jPath = response.jsonPath();
    int id = jPath.getInt("id");
    String name = jPath.getString("name");
    Assertions.assertEquals(dog.getId(), id);
    Assertions.assertEquals(dog.getName(), name);
  }
}
```



```

@Test
@Order(1)
@DisplayName("Update an existing pet")
void should_SuccessUpdateAnExistingPet_CompareIdAndNameUpdateNameDogFrenk(){
    dog.setName("Frenk");
    Response response = dog.update();
    JsonPath jPath = response.jsonPath();
    int id = jPath.getInt("id");
    String name = jPath.getString("name");
    Assertions.assertEquals(dog.getId(), id);
    Assertions.assertEquals(dog.getName(), name);
}

@Test
@Order(2)
@DisplayName("Uploads an image")
void should_UploadImagesForPet_ReturnCode200AndMessageUploadFileBytes(){
    Path filePath = Paths.get(dog.getImage());
    String fileName = filePath.getFileName().toString();
    Response response = dog.uploadImage();
    JsonPath jPath = response.jsonPath();
    int code = jPath.getInt("code");
    String type = jPath.getString("type");
    String message = jPath.getString("message");
    Assertions.assertAll(
        () -> Assertions.assertEquals(200, code),
        () -> Assertions.assertEquals("unknown", type),
        () -> Assertions.assertEquals(format("additionalMetadata: %s\nFile uploaded to
./%s, 116435 bytes", dog.getAdditionalMetadata(), fileName), message));
}

@Test
@Order(2)
@DisplayName("Finds Pets by status")
void should_FindsPetsByStatus_ReturnListAllPetStatusAvailable(){
    Response response = dog.find("status", "available");
    JsonPath jPath = response.jsonPath();
    List<Map<String, ?>> result = jPath.get("$");
    long countStatus_sold = result.stream().filter(s -> s.get("status").equals("sold"
)).count();
    long countStatus_pending = result.stream().filter(s -> s.get("status").equals(
"pending")).count();
    long countStatus_available = result.stream().filter(s -> s.get("status").equals(
"available")).count();
    Assertions.assertAll(
        () -> Assertions.assertEquals(0, countStatus_sold),
        () -> Assertions.assertEquals(0, countStatus_pending),
        () -> Assertions.assertNotEquals(373, countStatus_available));
}

@Test
@Order(2)
@DisplayName("Find pet by ID")
void should_FindsPetsById_ReturnDataPetThisId100(){

```

```

String petId = "100";
Response response = dog.getPetById(petId);
JsonPath jPath = response.jsonPath();
String id = jPath.getString("id");
Assertions.assertEquals(petId, id);
}
@Test
@Order(2)
@DisplayName("Updates a pet in the store with form data")
void should_UpdateAnExistingPet(){
    Response response = dog.updateWithFormData("name_1", "sold");
    JsonPath jPath = response.jsonPath();
    int code = jPath.getInt("code");
    String type = jPath.getString("type");
    String message = jPath.getString("message");
    Assertions.assertAll(
        () -> Assertions.assertEquals(200, code),
        () -> Assertions.assertEquals("unknown", type),
        () -> Assertions.assertEquals("100", message));
}
@Test
@Order(7)
@DisplayName("Delete a pet")
void should_SuccessDeleteAPet_ReturnCode200And_TypeUnknownMessage100(){
    Response response = dog.delete();
    JsonPath jPath = response.jsonPath();
    int code = jPath.getInt("code");
    String type = jPath.getString("type");
    String message = jPath.getString("message");
    Assertions.assertAll(
        () -> Assertions.assertEquals(200, code),
        () -> Assertions.assertEquals("unknown", type),
        () -> Assertions.assertEquals("100", message));
}
}

```

## Класс User

Содержит свойства и методы для управления CRUD пользователями

Свойство в котором храниться информация о endpoint (путь, параметры заголовка, пути к шаблонам запросов и схем валидации ответа)

```

@Qualifier("endpointUser")
@Autowired
protected Endpoint endpoint;

```

Свойства для заполнения тела запроса:

```
private int status_code;
private int id;
private String username;
private String firstName;
private String lastName;
private String email;
private String password;
private String phone;
private int userStatus;
```

Методы:

```
@Step("Создает массив пользователей с заданным входным массивом")
public Response createWithArray() {
    return null;
}
@Step("Создает список пользователей с заданным входным массивом")
public Response createWithList() {
    return null;
}
@Step("Обновленный пользователь")
public Response upload() {
    return null;
}
@Step("Получить пользователя по имени пользователя")
public Response find() {
    return null;
}
@Step("Удалить пользователя")
public Response delete() {
    return null;
}
@Step("Авторизует пользователя в системе")
public Response login() {
    return null;
}
@Step("Выход из текущего сеанса пользователя, вошедшего в систему")
public Response logout() {
    return null;
}
@Step("Создать пользователя")
public Response create() {
    return null;
}
}
```

Xml описание bean endpoint свойства endpoint в классе Pet

```

<bean name="endpointUser" class="endpoints.Endpoint">
  <property name="urls">
    <map>
      <entry key="createWithArray" value="/v2/user/createWithArray"/>
      <entry key="createWithList" value="/v2/user/createWithList"/>
      <entry key="byUserName" value="/v2/user/%s"/>
      <entry key="base" value="/v2/user"/>
      <entry key="login" value="/v2/user/login"/>
      <entry key="logout" value="/v2/user/logout"/>
    </map>
  </property>
  <property name="headerParams">
    <map>
      <entry key="accept" value="application/json"/>
      <entry key="Content-Type" value="application/json"/>
    </map>
  </property>
  <property name="request">
    <map>
      <entry key="create" value=
"src/test/resources/requests/add_a_new_pet_to_thestore.json"/>
    </map>
  </property>
  <property name="response">
    <map>
      <entry key="create_pass" value=
"src/test/resources/response/add_a_new_pet_to_thestore.json"/>
    </map>
  </property>
</bean>
</beans>

```

xml описание bean Pet для теста с положительным результатом

```

<bean name="pass" class="endpoints.Pet">
  <property name="status_code" value="200"/>
  <property name="id" value="100"/>
  <property name="name" value="Frenki"/>
  <property name="status" value="available"/>
  <property name="photoUrls">
    <list>
      <value>http://cairnterrier.ru/wp-content/uploads/2017/02/IMG-Weimaraner-1-
459x574.jpg</value>
    </list>
  </property>
  <property name="category">
    <map>
      <entry key="id" value="0"/>
      <entry key="name" value="dog"/>
    </map>
  </property>
</bean>

```

```

        </map>
    </property>
    <property name="tags">
        <map>
            <entry key="id" value="0"/>
            <entry key="name" value="веймар"/>
        </map>
    </property>
    <property name="image" value="src/test/resources/dog_veimar.jpg"/>
    <property name="additionalMetadata" value="jpgFile"/>
</bean>
</beans>

```

## Тесты валидации

```

@Link("https://petstore.swagger.io/#/pet")
@Epic("Зоомагазин->Домашний питомец->валидационные тесты")
@Feature("Пользователь")
@ExtendWith(SpringExtension.class)
@ContextConfiguration({"datapools/user.xml", "endpoints/petstore.xml"})
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
//TODO: реализовать для генерации данных для валид тестов негативного характера
//@ExtendWith(SetupUser.class)
//@ExtendWith(CleanupUser.class)
public class UserTest {
    @Qualifier("pass_user")
    @Autowired
    User user;
    @BeforeEach
    public void setUp() {
        RestAssured.baseURI = System.getenv("BASE_URL");
        user.setRequestSpecification(sessionAndContentTypeJson(System.getenv("API_KEY")));
    }
    @Test
    @Order(0)
    @DisplayName("Logs user into the system")
    void should_LogsUserIntoTheSystem_ReturnLis4t(){
        Response response = user.login();
    }
    @Test
    @Order(1)
    @DisplayName("Create user")
    void should_CreateUser_ReturnList6(){
        Response response = user.create();
    }
    @Test
    @Order(2)
    @DisplayName("Creates array of users with given input array")
    void should_CreatesArrayOfUsersWithGivenInputArray_ReturnArray(){
        Response response = user.createWithArray();
    }
}

```

```

}
@Test
@Order(2)
@DisplayName("Creates list of users with given input array")
void should_CreatesListOfUsersWithGivenInputArray_ReturnList(){
    Response response = user.createWithList();
}
@Test
@Order(2)
@DisplayName("Get user by user name")
void should_GetUserByUserName_ReturnUserData(){
    Response response = user.find();
}
@Test
@Order(2)
@DisplayName("Updated user")
void should_UpdatedUser_ReturnUserData(){
    Response response = user.upload();
}
@Test
@Order(3)
@DisplayName("Delete user")
void should_DeleteUser_ReturnSuccessMessage(){
    Response response = user.delete();
}
@Test
@Order(4)
@DisplayName("Logs out current logged in user session")
void should_LogsOutCurrentLoggedInUserSession_ReturnList5(){
    Response response = user.logout();
}
}

```

## Запуск тестов

### Собрать docker images

1. Перейти в корень проекта
2. Запустить команду "docker build -t petstore-tz ."

### Запустить тесты из docker-compose

1. Перейти в корень проекта
2. Заменить нужными значениями параметры BASE\_URL и API\_KEY в файле docker-compose.yml

**BASE\_URL:** <https://petstore.swagger.io>

3. Запустить команду "docker-compose up"

## Артефакты проекта

### POM.XML

```
<groupId>ru.ibs.pfadeev</groupId>
<artifactId>petstore-tz</artifactId>
<version>1.0-SNAPSHOT</version>
```

### Свойства проекта

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
```

### Подключённые зависимости к проекту

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.30</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.3.30</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.2.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>3.0.0</version>
  </dependency>
```

```

<dependency>
  <groupId>io.qameta.allure</groupId>
  <artifactId>allure-junit5</artifactId>
  <version>2.24.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.30</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.4.11</version>
  <exclusions>
    <exclusion>
      <artifactId>slf4j-api</artifactId>
      <groupId>org.slf4j</groupId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>2.0.9</version>
</dependency>
</dependencies>

```

## Плагины проекта

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.2.1</version>
      <configuration>
        <testFailureIgnore>>false</testFailureIgnore>
        <argLine>
          -javaagent:"${settings.localRepository}/org/aspectj/aspectjweaver/1.9.19/aspectjweaver-1.9.19.jar"
          </argLine>
        <systemProperties>
          <property>
            <name>allure.results.directory</name>
            <value>${project.build.directory}/allure-results</value>
          </property>
        </systemProperties>
      </configuration>
    </plugin>
  </plugins>
</build>

```



```

    <dependencies>
      <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.9.19</version>
      </dependency>
    </dependencies>
  </plugin>
</plugins>
</build>

```

## DOCKERFILE

```

ARG DOCKER_IMAGE=maven:3.8-jdk-11
FROM ${DOCKER_IMAGE}
ENV MAVEN_CONFIG=/home/maven
ENV JAVA_TOOL_OPTIONS -Duser.home=/home/maven
RUN mkdir -p /home/maven/
WORKDIR /home/maven
COPY pom.xml /home/maven/
COPY src /home/maven/src
CMD mvn -Dmaven.home=/home/maven -e -q -B test -Dmaven.main.skip
RUN mvn -Dmaven.home=/home/maven -e -q -B -U -Dmaven.wagon.http.ssl.insecure=true
-Dmaven.wagon.http.ssl.allowall=true -Dmaven.wagon.http.ssl.ignore.validity.dates=true
test-compile
RUN chmod -R 777 /home/maven/.m2 /home/maven/target

```

## DOCKER-COMPOSE

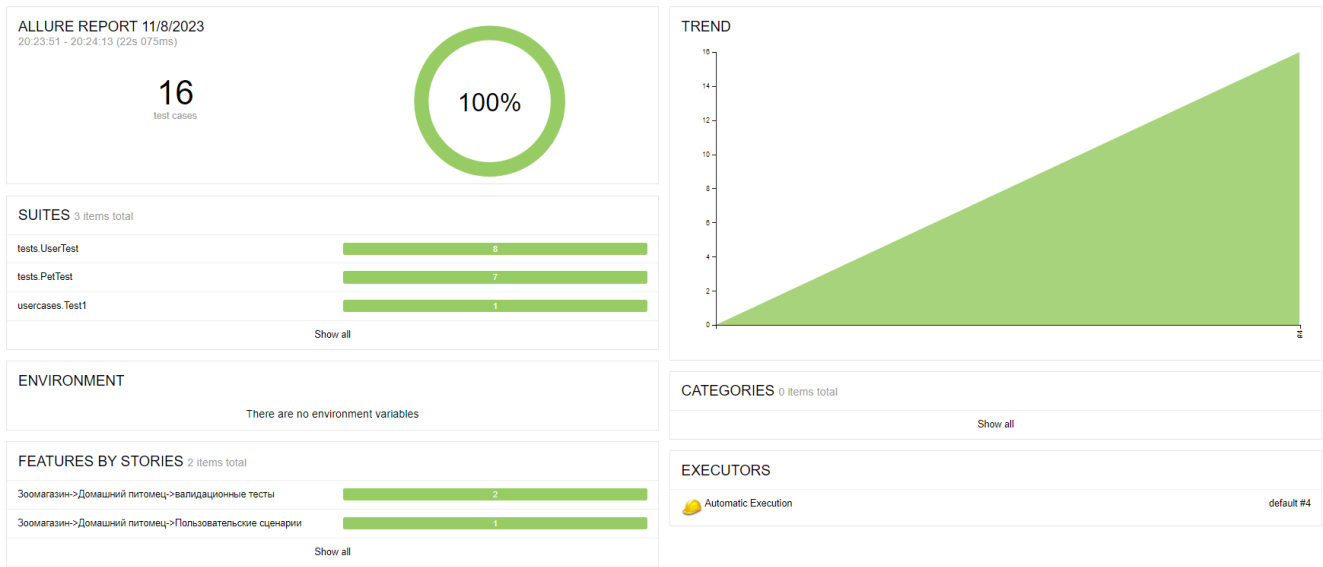
```

version: '3.7'
services:
  petstore-tz:
    container_name: "petstore-tz"
    image: petstore-tz:latest
    volumes:
      - /home/maven/target/allure-results:YOU_PATH
    environment:
      BASE_URL: https://petstore.swagger.io
      API_KEY: special
    restart: 'no'

```

# Отчёт

## Сводный отчет



## Пользовательские сценарии

