Лабораторная работа №6

Тестирование API

Цель: Познакомится с основами написания API тестов, их спецификой, а также инструментами и фреймворками которые помогут в ручном и автоматизированном тестировании API.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание.

1. Теоретические сведения

Перед началом работы необходимо установить следующее:

- Open JDK (https://openjdk.java.net)
- Java IDE (рекомендуемыя IDE: IntelliJ IDEA Community)
- Git (https://git-scm.com/downloads)

Опционально (в случае если используется НЕ IntelliJ IDEA):

- Maven (https://maven.apache.org)
- Git клиент (например, SourceTree https://www.sourcetreeapp.com)
- Postman https://www.postman.com (для отправки тестовых REST запросов)

1.1. Тестирование API

API — это Application Programming Interface, или программный интерфейс приложения, с помощью которого одна программа может взаимодействовать с другой. API позволяет слать информацию напрямую из одной программы в другую, минуя интерфейс взаимодействия с пользователем.

API может быть внутренним, частным — когда программные компоненты связаны между собой и используются внутри системы. А может быть

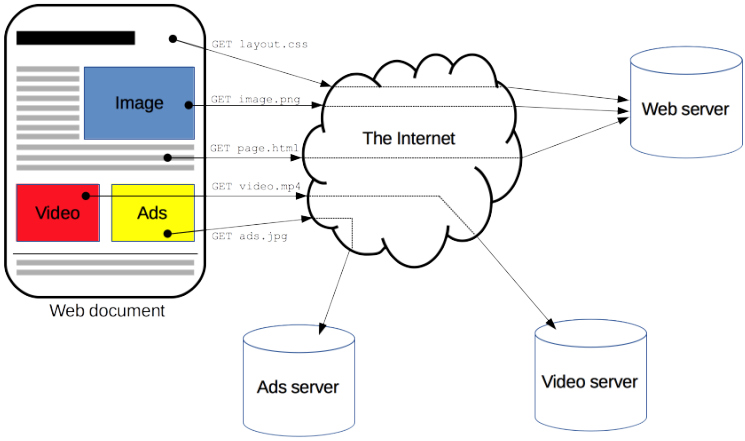
открытым, публичным — в таком случае он позволяет внешним пользователям или другим программам получать информацию, которую можно интегрировать в свои приложения.

Чтобы программам общаться между собой, их API нужно построить по единому стандарту. Одним из них является REST — стандарт архитектуры взаимодействия приложений и сайтов, использующий протокол HTTP. Особенность REST в том, что сервер не запоминает состояние пользователя между запросами. Иными словами, идентификация пользователя (авторизационный токен) и все параметры выполнения операции передаются в каждом запросе. Этот подход настолько прост и удобен, что почти вытеснил все другие.

Тестирование API проводят, основываясь на бизнес-логике программного продукта. Тестирование API относится к интеграционному тестированию, а значит в ходе него можно отловить ошибки взаимодействия между модулями системы или между системами. Для тестирования используют специальные инструменты, где можно отправить входные данные в запросе и проверить точность выходных данных.

1.2. HTTP

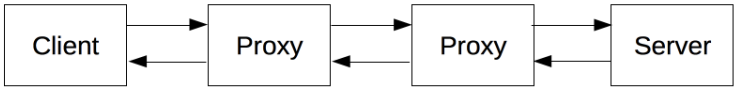
HTTP — это протокол, позволяющий получать различные ресурсы, например HTML-документы. Протокол HTTP лежит в основе обмена данными в Интернете. HTTP является протоколом клиент-серверного взаимодействия, что означает инициирование запросов к серверу самим получателем, обычно веб-браузером (web-browser). Полученный итоговый документ будет (может) состоять из различных поддокументов являющихся частью итогового документа: например, из отдельно полученного текста, описания структуры документа, изображений, видео-файлов, скриптов и многого другого.



Клиенты и серверы взаимодействуют, обмениваясь одиночными сообщениями (а не потоком данных). Сообщения, отправленные клиентом, обычно веб-бруезером, называются запросами, а сообщения, отправленные сервером, называются ответами.

HTTP — это клиент-серверный протокол, то есть запросы отправляются какой-то одной стороной — участником обмена (user-agent) (либо прокси вместо него). Чаще всего в качестве участника выступает веб-браузер, но им может быть кто угодно, например, робот, путешествующий по Сети для пополнения и обновления данных индексации веб-страниц для поисковых систем.

Каждый запрос (англ. request) отправляется серверу, который обрабатывает его и возвращает ответ (англ. response). Между этими запросами и ответами как правило существуют многочисленные посредники, называемые прокси, которые выполняют различные операции и работают как шлюзы или кэш, например.



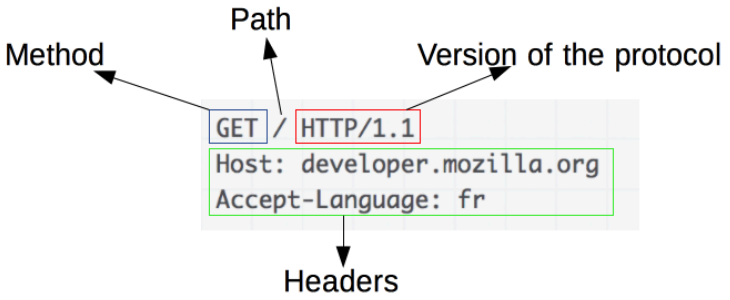
HTTP/1.1 и более ранние HTTP сообщения человеко-читаемы. В версии HTTP/2 эти сообщения встроены в новую бинарную структуру, фрейм, позволяющий оптимизации, такие как компрессия заголовков и

мультиплексирование. Даже если часть оригинального HTTP сообщения отправлена в этой версии HTTP, семантика каждого сообщения не изменяется и клиент воссоздаёт (виртуально) оригинальный HTTP-запрос. Это также полезно для понимания HTTP/2 сообщений в формате HTTP/1.1.

Существует два типа HTTP сообщений, запросы и ответы, каждый в своем формате.

Запросы

Примеры HTTP запросов:

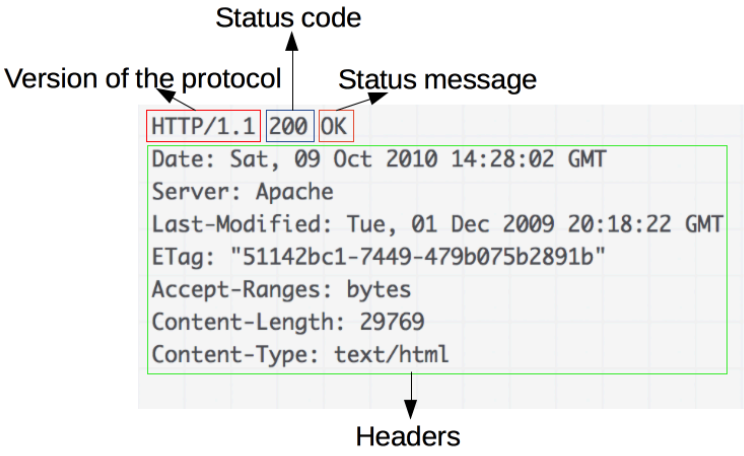


Запросы содержат следующие элементы:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операция могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без protocol (http://), domain (здесь developer.mozilla.org), или TCP port (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляюшие дополнительную информацию для сервера.
- Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Ответы

Примеры ответов:



Ответы содержат следующие элементы:

- Версию HTTP-протокола.
- HTTP код состояния, сообщающий об успешности запроса или причине неудачи.
- Сообщение состояния — краткое описание кода состояния.
- HTTP заголовки, подобно заголовкам в запросах.
- Опционально: тело, содержащее пересылаемый ресурс.

Методы HTTP запросов

HTTP определяет множество методов запроса, которые указывают, какое желаемое действие выполнится для данного ресурса. Несмотря на то, что их названия могут быть существительными, эти методы запроса иногда называются HTTP глаголами. Каждый реализует свою семантику.

Метод	**Описание**
GET	Метод GET запрашивает представление ресурса. Запросы с

Метод	Описание
	использованием этого метода могут только извлекать данные.
HEAD	HEAD запрашивает ресурс так же, как и метод GET, но без тела ответа.
POST	POST используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.
PUT	PUT заменяет все текущие представления ресурса данными запроса.
DELETE	DELETE удаляет указанный ресурс.
CONNECT	CONNECT устанавливает "туннель" к серверу, определённому по ресурсу.
OPTIONS	OPTIONS используется для описания параметров соединения с ресурсом.
TRACE	TRACE выполняет вызов возвращаемого тестового сообщения с ресурса.
PATCH	PATCH используется для частичного изменения ресурса.

Коды состояния HTTP

Код состояния HTTP (HTTP status code) — часть первой строки ответа сервера при запросах по протоколу HTTP. Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом

поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа. Примеры:

- 201 Created.
- 401 Unauthorized.
- 507 Insufficient Storage. Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше. Набор кодов состояния является стандартом. Более детальная информация о наборе поддерживаемых кодов состояния: https://www.restapitutorial.com/httpstatuscodes.html

1.3. REST

REST (от англ. Representational State Transfer – «передача состояния представления») обеспечивает общение между клиентом (как правило, это браузер) и сервером с помощью обычных HTTP-запросов (GET, POST, PUT, DELETE и т. д), передавая информацию от клиента в параметрах самих запросов, информацию от сервера – в теле ответа (который может быть, например, JSON-объектом или XML-документом). REST является архитектурным стилем, а не стандартом.

Принципы REST:

- Ресурсы позволяют легко понять структуру каталогов URI
- Представления передают JSON или XML в качестве представления данных объекта и атрибутов
- Сообщения используют HTTP методы явно(например, GET, POST, PUT и DELETE)
- Отсутствие состояния взаимодействий не сохраняет контекст клиента на сервере между запросами

Используются HTTP методы для определения CRUD (create, retrieve, update, delete) операций HTTP запросов.

GET Получение информации. GET запросы должны быть безопасны и идемпотентны, т.е. независимо от того, как много времени они повторяются с теми же самыми параметрами, результат останется тот же. Они могут

иметь побочные эффекты, но пользователь не ожидает их, поэтому они не могут критичными для функционирования системы. Запросы могут быть также частичными или условными.

Получение адреса по ID, равным 1:

```
GET /addresses/1
```

POST Запрос что-то сделать с ресурсом по URI с предоставлением сущности. Часто POST используется для создания новой сущности, но также возможно использовать и для обновления существующей.

Создание нового адреса:

```
POST /addresses
```

PUT Сохраняет сущность по URI. PUT может создавать новую сущноть или обновлять существующую. PUT запрос идемпотентен. Идемпотентность - главное отличие в поведении между PUT и POST запросом.

Изменение адреса по ID, равным 1:

```
PUT /addresses/1
```

PUT заменяет сущетвующую сущность. Те элементы сущности, которые не представлены в запросе, будут очищены или заменены на null.

PATCH Обновляет только определенные поля сущности по URI. PATCH запрос идемпотентен. Идемпотентность - главное отличие в поведении между PUT и POST запросом.

```
PATCH /addresses/1
```

DELETE Запрос, который удаляет ресурс; кроме того, ресурс не должен быть удален немедленно. Он может быть асинхронным или "долгоиграющим" запросом.

Удаления адреса по ID, равным 1:

```
DELETE /addresses/1
```

HTTP статус коды Статус коды указывают на результат HTTP запроса.

1XX - информационный 2XX - успешное выполнение 3XX - перенаправление 4XX - ошибка клиента 5XX - ошибка сервера

Медиа типы Accept и **Content-Type** HTTP заголовки могут быть использованы для описания содержимого, которое будет отправлено или запрошено HTTP запросом. Клиент может установить Accept в application/json, если он запрашивает ответ в JSON. И наоборот, когда отправляются данные, установленный Content-Type в application/xml говорит клиенту, что данные были отправлены в XML форме.

1.4. REST-assured

REST Assured - это Java DSL для упрощения тестирования REST сервисов. Он поддерживает запросы POST, GET, PUT, DELETE, OPTIONS, PATCH и HEAD и может использоваться для отправки запросов и проверки ответа на эти запросы.

Тест REST-assured состоит из трех основных частей:

- **given()** задает предварительные условия запроса. Например, Content-Type (JSON/XML), параметры логирования и тело запроса. Часть given() необязательна, но чаще всего она используется.
- **when()** – здесь указываются параметры запросы (URI, метод)
- **then()** – тут выполняется проверка ответа от сервера.

Пример:

```
@Test
public void testVerifyResponseBodyForCreatePost() {
    Post newPost = new Post(2, "Title", "Some text");
    given()
            .log().ifValidationFails()
            .contentType(ContentType.JSON)
            .body(newPost)
    .when()
```

```
            .post( "/posts")
    .then()
            .assertThat()
            .body("userId", equalTo(newPost.getUserId()))
            .body("title", equalTo(newPost.getTitle()))
            .body("body", equalTo("Some text"))
            .body("id", notNullValue());
}
```

Более подробная информация доступна в официальной документации REST-assured: https://github.com/rest-assured/rest-assured/wiki/Usage

1.5. Вспомогательные фреймворки

Jackson

Jackson - это очень популярная и эффективная Java библиотека сериализации/десериализации Java-объектов в\из JSON. REST-assured может использовать её для автоматического преобразования тела запроса \ответа в формате JSON при написании тестов.

При совпадении названии класса\полей, представляющих тестовые данные с назвнанием объектов и полей в целевом JSON, используемом в запросах достаточно просто подключить Jackson библиотеку как Maven зависимость. В случае каких-либо несовпадений в классе и формате JSON представления или необходимости тонкой настройки под конкретный случай можно воспользоваться Jackson аннотациями и другими инструментами которые предоставляет библиотека. Но в рамках текущей лабораторной поведения по-умолчанию должно быть достаточно. Подробное описание библиотеки и вариантов её использования можно найти в официальной документации: https://github.com/FasterXML/jackson-docs

В качестве альтернативы Jackson библиотеки можно использовать её аналог Gson: https://github.com/google/gson

Hamcrest

Hamcrest - это фреймворк для написания условий соответствия, позволяющий описывать правила «соответствия» в декларативном стиле. Существует ряд ситуаций, в которых проверки соответствия неоценимы,

такие как проверка пользовательского интерфейса или фильтрация данных, но именно в области написания гибких тестов сопоставления используются наиболее часто.

Hamcrest не является обязательной библиотекой для использования REST-assured, но Hamcrest может помочь сделать ваши тесты максимально удобочитаемыми.

Пример использования Hamcrest:

```
@Test
  public void testEquals() {
    Biscuit theBiscuit = new Biscuit("Ginger");
    Biscuit myBiscuit = new Biscuit("Ginger");
    assertThat(theBiscuit, equalTo(myBiscuit));
  }
```

Подключить Hamcrest можно так же как и все остальные библиотеки используя Maven. Дополнительная ифнормация по использованию: http://hamcrest.org/JavaHamcrest/tutorial

2. Практические задания

2.1. Подготовительная работа

Для выполнения задания необходимо зарегистрироваться на сайте https://gorest.co.in и получить персональный access token. Для успешного выполнения запросов к сервису необходимо добавлять полученный access token для каждого запроса одним из следующих способов:

- добавить access token к URI ресурса. Например:

```
https://gorest.co.in/public-api/users?access-token=xxxxxxxx
```

- Добавить токен в заголовок запроса:

```
Key: Authorization
Value: Bearer <your_access_token>
```

Необходимо реализовать набор REST API тестов согласно описанным в задании условиям и залить созданные тесты на персональный репозиторий на https://github.com

Для первоначальной проверки корректности составления запроса перед тем как переносить его в автоматизированные тесты можно использовать Postman (https://www.postman.com/product/api-client)

2.2. Задания

Задание 1 Напишите позитивные тесты проверяющие следующие ресурсы доступные на https://gorest.co.in:

1. GET /public-api/users : получить список всех пользователей
2. GET /public-api/users?first_name=<user_name> : получить список пользователей с указанным именем
3. POST /public-api/users : создать нового пользователя
4. GET /public-api/users/<user_id> : получить пользователя по его ID
5. PUT /public-api/users/<user_id> : изменить пользователя с указанным ID
6. DELETE /public-api/users/<user_id> : удалить пользователя с указанным ID

Для каждого ресурса тесты должен проверять следующие параметры:

- статус ответа от сервера
- содержимое тела запроса (для ресурсов 2, 3, 4, 5)

опционально можно также проверить заголовки ответа.

Задание 2 Напишите негативные тесты проверяющие следующие сценарии:

1. GET /public-api/users без указания токена авторизации и с некорректным токеном
2. POST /public-api/users с некорректным форматом тела запроса
3. DELETE /public-api/users

Для каждого сценария необходимо проверить следующее:

- статус ответа от сервера (должен отвечать характеру некорректного обращения к API в соответствии с описанием, приведённым в секции "Response Codes": https://gorest.co.in)
- текст ошибки в теле запроса (необходимо извлечь текст ошибки и сравнить с ожидаемым значением)

опционально можно проверить заголовки ответа.