Лабораторная работа №4

«Тестирование REST API и знакомство с Postman»

Цель работы:

Получение знаний об особенностях тестирования REST API, а также базовых умений и навыков использования инструмента Postman.

Теоретическая часть

Терминология

API (Application Programming Interface, программный интерфейс приложения) — набор правил, протоколов и инструментов для взаимодействия между приложениями. Другими словами, API — это интерфейс, определяющий, как одна программа должна взаимодействовать с другой. Как правило, API представляет собой набор функций, которые могут быть вызваны другой программой.

REST (Representational State Transfer, передача состояния представления) — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Архитектурный стиль — это набор согласованных ограничений и принципов проектирования, позволяющий добиться определённых свойств системы.

Назначение REST состоит в том, чтобы придать проектируемой системе такие свойства, как:

- Производительность;
- Масштабируемость;
- Гибкость к изменениям;
- Отказоустойчивость;
- Простота поддержки.

Перечисленные выше свойства являются нефункциональными требованиями к разрабатываемому приложению. Существует 6 принципов REST, которые помогают добиться выполнения этих требований:

1. Клиент-серверная архитектура (Client-Server);
2. Отсутствие сохранения состояния (Stateless);

3. Кеширование (Cache);

4. Единообразный интерфейс (Uniform Interface);

5. «Слоистая» архитектура (Layered System);

6. Код по требованию (Code-On-Demand) — опциональный принцип.

Таким образом, REST — способ организации программной системы, построенный на вышеуказанных принципах. Приложения, которые полностью придерживаются этих принципов, называются RESTful. Для приложений, которые частично соблюдают принципы REST, используют термин REST-like.

Принципы REST

Клиент-серверная архитектура (Client-Server)

Концепция клиент-серверной архитектуры заключается в разделении некоторых зон ответственности: в разделении функций клиента и сервера.

Сервер — программа, в которой хранятся и обрабатываются ресурсы. Ресурсы представляют собой любые данные: текст, изображение, видео, аудио и т. д. Сервер может располагаться на одном или нескольких компьютерах; но даже в одном компьютере может быть несколько виртуальных серверов.

Клиент — программа, которая запрашивает у сервера доступ к ресурсам. Для этого она использует API.

Можно разделить систему так, что клиент реализует только функциональное взаимодействие с сервером, при этом сервер реализует в себе логику хранения данных, сложные взаимодействия со смежными системами и т. д. (рисунок 1).

Рисунок 1 — Клиент-серверная архитектура

Также сервер может иметь базу данных (рисунок 2). В данном случае пара «сервер и БД» тоже будет парой «клиент-сервер» соответственно.

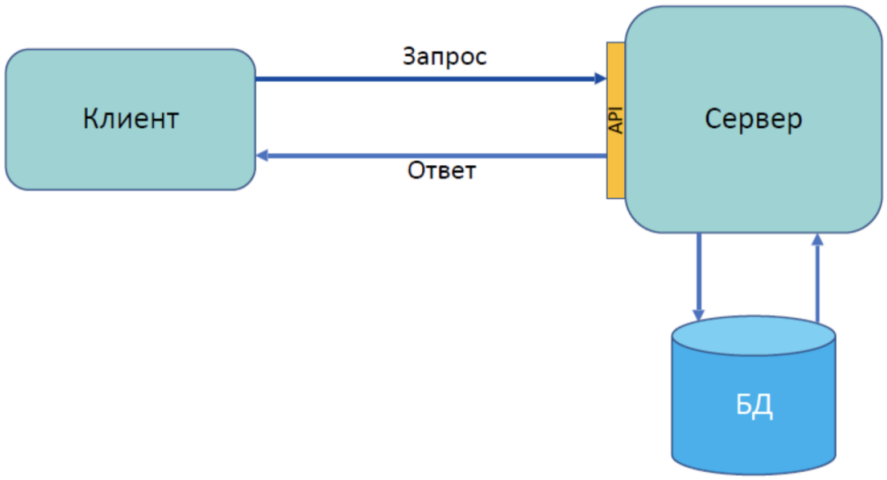

Рисунок 2 — Трехуровневая архитектура

Преимущества:

+ Масштабируемость;

+ Определенная простота поддержки.

Недостатки:

– Увеличение нагрузки в связи с переносом части логики с клиента на сервер (рост количества запросов между клиентом и сервером, так как у клиента меньше доступных возможностей)

– Сервер является единой точкой отказа: при отказе единственного сервера система становится неработоспособной.

Отсутствие сохранения состояния (Stateless)

Сервер не должен хранить какую-либо информацию о сессии с клиентом. В запросе должна храниться вся необходимая информация для его обработки и, если необходимо, идентификации клиента.

Преимущества:

+ Масштабируемость сервера;

+ Уменьшение времени обработки запроса,

+ Простота поддержки,

+ Возможность использовать кэширование.

Недостатки:

– Усложнение логики клиента (именно на стороне клиента нужно хранить всю информацию о состоянии, о допустимых и недопустимых

действиях и т. д.);

– Увеличение нагрузки на сеть за счет передачи большого объема информации (всего контекста).

Кеширование (Cache)

Клиенты и промежуточные узлы могут кешировать ответы сервера. Это нужно и полезно, если у сервера часто запрашивают одинаковую информацию. Например, кэширование активно используется на новостных сайтах или в соцсетях (на веб-ресурсах, к которым происходит много обращений).

Преимущества:

+ Уменьшение количества сетевых взаимодействий;

+ Уменьшение нагрузки на системы (не грузим их дополнительными запросами).

Недостатки:

– При сравнительно малом количестве одинаковых обращений реализация кеширования неуместна;

– Сохраненные ранее данные могут устареть.

Единообразный интерфейс (Uniform Interface)

Должен быть единый способ обращения к каждому ресурсу.

Файлы обычно передаются клиенту не в том виде, в котором хранятся на сервере. В вебе их часто преобразуют в JSON или XML и только потом отправляют клиенту. Ответ на запросы к новому ресурсу должен приходить в том же формате, что и к старым, и сразу же содержать дополнительную информацию: что разрешается делать с ресурсом, можно ли его изменять и удалять на сервере и так далее.

Для реализации единообразного интерфейса в REST API используется принцип HATEOAS.

HATEOAS (Hypermedia as the Engine of Application State) — одно из ограничений REST, согласно которому сервер возвращает не только ресурс, но и его связи с другими ресурсами и действия, которые можно с ним

совершить.

Главное преимущество: клиент становится очень гибким в плане изменений на сервере с точки зрения изменения допустимых действий, изменения модели данных и т.д.

Недостаток: сильное усложнение логики, в первую очередь, клиента. Это может потянуть за собой и усложнение логики на сервере, потому что такие ответы нужно правильно формировать. Фактически ответственность за действия, которые совершает клиент, передаются на его же сторону.

«Слоистая» архитектура (Layered System)

Между сервером и клиентом есть несколько промежуточных узлов, выполняющих вспомогательные функции, — прокси-серверы. Они используются для кэширования, обеспечения безопасности, дополнительной обработки данных. Если основных серверов несколько, то дополнительные серверы-балансировщики могут распределять нагрузку между ними и решать, в какой из них направлять запрос (рисунок 3).

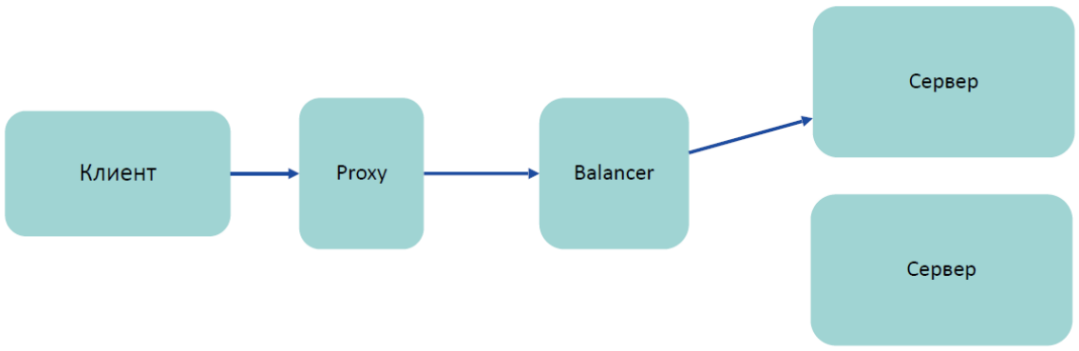

Рисунок 3 — Модель слоистой архитектуры

Концепция слоистой архитектуры заключается в том, что ни клиент, ни сервер не должны знать о том, как происходит цепочка вызовов дальше своих прямых соседей.

Преимущества:

+ Возможность изменять общую архитектуру без доработок на стороне клиента или сервера.

Недостатки:

– Увеличение нагрузки на сеть (больше участников и больше вызовов);

– Увеличение времени получения ответа.

Код по требованию (Code-On-Demand)

Этот принцип означает, что сервер в ответ на запрос может отправить исходный код, который выполняется уже на стороне клиента. Благодаря этому можно передавать целые сценарии. Например, динамические элементы пользовательского интерфейса, написанные на JavaScript.

В REST API требование необязательно, потому что не всем сайтам и сервисам нужно умение работать с готовыми скриптами.

Вызов операций REST API

Так как REST — архитектурный подход, а не протокол, в нём не заложено никаких конкретных методов. Клиентские приложения обращаются к REST API по протоколу HTTP.

HTTP (HyperText Transfer Protocol) — широко распространённый протокол передачи данных, изначально предназначенный для передачи документов. Протокол означает четко специфицированные правила передачи информации, которые клиент и сервер должны соблюдать и выполнять.

HTTP-запрос состоит из следующих частей:

1. HTTP-метод (GET, PUT, POST, DELETE и т.п.). Подробнее обо всех методах будет рассказано далее.

2. Uniform Resource Identifier (URI) — описание ресурса;

3. Версия HTTP;

4. Заголовки запроса;

5. Payload — тело запроса, содержащее непосредственно само сообщение серверу.

HTTP-ответ содержит 4 элемента:

1) Код (или номер) ответа, обозначающий статус. Этот код отправляется сервером в ответ на запрос клиентской программы. Существует список кодов состояния HTTP. Код представляет собой целое трёхразрядное десятичное

число, первая цифра которого указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

Классы состояния:

1 — Информационные

2 — Успех

3 — Перенаправление

4 — Ошибка клиента

5 — Ошибка сервера

2) Версия HTTP

3) Заголовки: тип контента (Content-type), его длина (Content-length), дата, текущее состояние сервера и его тип.

4) Тело ответа — данные, запрошенные клиентским приложением.

Метод HTTP — это команда HTTP, указывающая на основную операцию над ресурсом, с которой начинается первая строка клиентского запроса. Методы HTTP представлены в таблице 1.

Таблица 1 — Методы HTTP

Метод	Описание
GET	Используется для запроса содержимого указанного ресурса
HEAD	Применяется для извлечения метаданных, проверки наличия ресурса
PUT	Применяется для загрузки содержимого запроса на указанный в запросе URI. Используется, когда надо не создать ресурс, а поменять его значения (атрибуты)
DELETE	Удаляет указанный ресурс
POST	Применяется для передачи пользовательских данных заданному ресурсу.
PATCH	Позволяет только создание или полную замену документа

Существует несколько способов описания объектов в теле запроса, то

есть, несколько форматов представления данных. REST API могут использовать любой формат сообщений, включая XML, JSON, Atom, RSS, CSV, HTML и другие.

Несмотря на разнообразие параметров формата сообщений, большинство REST API используют JSON (нотацию объектов JavaScript) в качестве формата сообщений по умолчанию. Они используют JSON, потому что он обеспечивает легкий, простой и более гибкий формат сообщений, который увеличивает скорость связи.

JSON (JavaScript Object Notation) — текстовый формат представления данных в нотации объекта JavaScript.

В качестве значений в JSON могут быть использованы:

- запись — это неупорядоченное множество пар ключ: значение, заключённое в «{ }».
- массив — это упорядоченное множество значений. Массив заключается в «[ ]».
- число (целое или вещественное).
- литералы true («истина»), false («ложь») и null.
- строка — это упорядоченное множество символов юникода, заключённое в двойные кавычки.

Знакомство с Postman

Postman — это сервис для создания, тестирования, документирования, публикации и обслуживания API. Он позволяет создавать коллекции запросов к любому API, применять к ним разные окружения, настраивать мок-серверы, писать автотесты на JavaScript, анализировать и визуализировать результаты запросов.

Скачать Postman можно бесплатно на официальном сайте https://www.postman.com/downloads/. Есть версии под Linux, Windows и macOS. Для использования платформы понадобится регистрация.

Для изучения основ тестирования распределенных приложений можно использовать открытые API, расположенные на серверах в Интернете,

например, The Star Wars API (SWAPI) или REQ | RES API и другие.

Основная сущность в Postman — **рабочее пространство** (workspace). В бесплатной версии доступно три вида рабочих пространств:

- личное — видно только владельцу;
- командное — видно только членам команды, которые в нём работают;
- публичное — доступно всем желающим.

Рабочее пространство Postman представлено на рисунке 4.

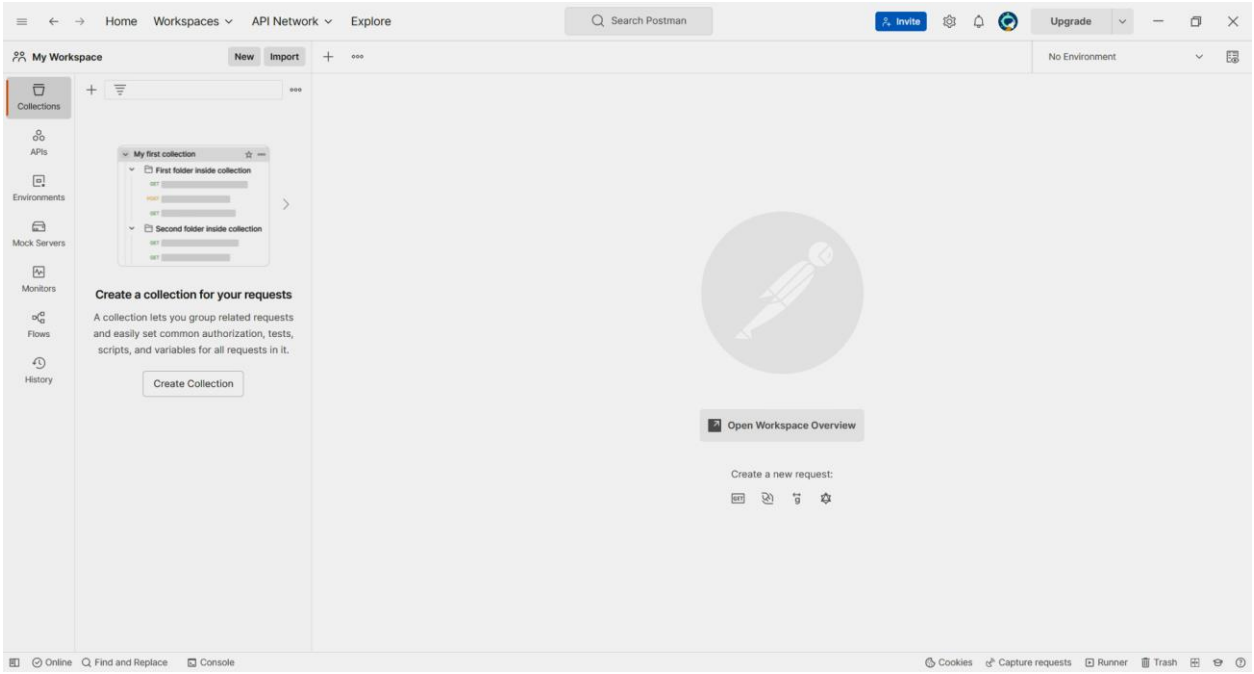

Рисунок 4 — Рабочее пространство Postman

Семь вкладок боковой панели слева соответствуют основным инструментам Postman:

1. Collections. Здесь хранятся коллекции запросов к API. Несколько запросов внутри одной коллекции можно объединять в папки.

2. APIs. В этой вкладке можно хранить целые API: их определения и коллекции.

3. Environments. Здесь создаются и хранятся окружения, в которых лежат переменные. Активное окружение можно менять из любой вкладки в правом верхнем углу. Также тут можно создавать глобальные переменные, с которыми можно работать везде вне зависимости от окружения.

4. Mock Servers. Здесь можно создавать фиктивные серверы, которые имитируют поведение реальных. Это полезно при тестировании.

5. Monitors. Мониторы позволяют визуализировать и отслеживать работу API: следить за производительностью, проверять доступность и корректность работы сервера по расписанию, отправлять уведомления о сбоях.

6. Flows. Это инструмент, с помощью которого можно настроить логику API-приложений в визуальном редакторе. На март 2023 года он всё ещё доступен только в тестовом режиме, чтобы его использовать, придётся отправить заявку.

7. History. Здесь хранится история всех отправленных запросов. Их можно сохранять, объединять в коллекции, создавать для них документацию, мониторы и мок-серверы.

Создание коллекции

Для создания коллекции нужно перейти во вкладку Collections и нажать плюс. В левой части приложения появится пункт New Collection. Для изменения названия необходимо нажать на три точки справа от New Collection и во всплывающем окне выбрать пункт Rename. В рамках данной лабораторной работы предлагается тестировать открытый The Star Wars API (SWAPI), поэтому назовем новую коллекцию StarWarsCollection (рисунок 5).

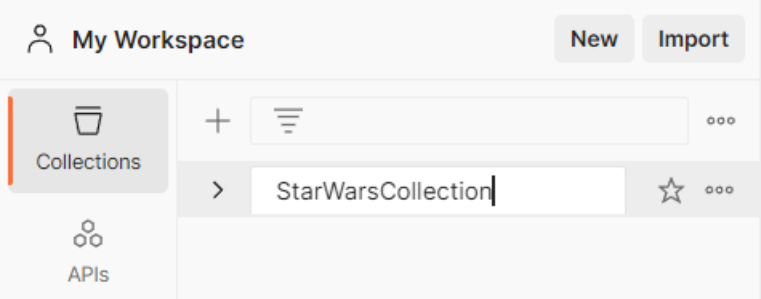

Рисунок 5 — Созданная коллекция StarWarsCollection

В рабочей области коллекции есть пять вкладок:

1. Authorization. Здесь можно настроить метод и параметры авторизации, которые будут использоваться в каждом запросе внутри

коллекции.

2. Pre-request Script. В данной вкладке можно написать программу на JavaScript, которая будет выполняться перед каждой отправкой запроса внутри коллекции. Для наиболее распространённых алгоритмов Postman предлагает готовые сниппеты. Их можно использовать, чтобы не писать код с нуля.

3. Tests. Работает как Pre-request Script, но выполняет код после выполнения запроса. Именно этот раздел используют тестировщики для проверки API. Здесь тоже есть готовые сниппеты.

4. Variables. Здесь можно создать переменную и присвоить ей значение. Потом эту переменную можно использовать, указав её название в двойных фигурных скобках {{имя переменной}}.

5. Runs. Postman позволяет запускать запросы не по отдельности, а все сразу — внутри одной коллекции или папки. В разделе Runs хранится информация о таких прогонах и результатах их тестов.

Создание переменной

Виды переменных:

- Глобальные переменные применяются ко всему рабочему пространству. Написать их можно во вкладке Environments в разделе Globals.
- Переменные коллекции создаются внутри конкретной коллекции и работают только внутри неё.
- Переменные окружения задаются на уровне окружения во вкладке Environments. Чтобы применить их к запросу, нужно напрямую связать его с окружением.
- Локальные переменные существуют на уровне скриптов, которые выполняются при отправке запросов.
- Переменные данных возникают, когда используется Collection Runner — инструмент для запуска сразу всех скриптов внутри коллекции или папки.

Добавим переменную ранее созданной коллекции StarWarsCollection. Необходимо нажать на название коллекции, выбрать вкладку Variables и задать ей имя в поле «VARIABLE» и начальное значение в поле «INITIAL

VALUE». Создадим переменную коллекции base_url со значением https://swapi.dev/api/. После создания переменной необходимо сохранить изменения (рисунок 6).

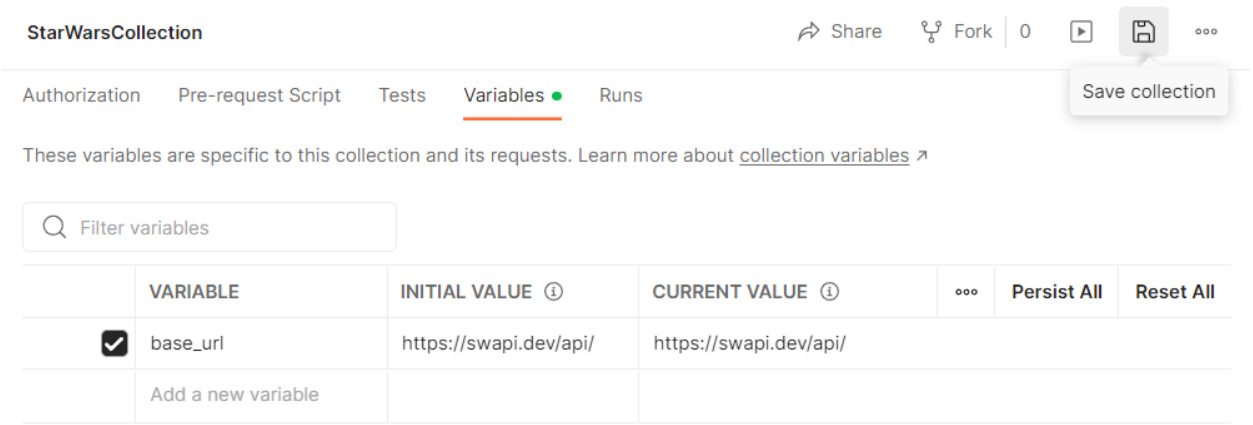

Рисунок 6 — Созданная переменная base_url

Создание запроса

Существует несколько способов для создания запроса:

- нажать в левом верхнем углу кнопку New и выбрать нужный тип запроса;
- нажать на плюс на панели вкладок в верхней части приложения;
- нажать на три точки возле названия коллекции и во всплывающем меню выбрать пункт Add request.

В первых двух случаях при первом сохранении запроса нужно указать коллекцию, в которой тот будет храниться.

Создадим запрос, содержащий только ранее созданную переменную. В результате выполнения запроса получим список URL, доступных для использования (рисунок 7).

SWAPI содержит информацию о следующих ресурсах: people (персонажи), planets (планеты), films (фильмы), species (расы и существа), vehicles (транспортные средства, не передвигающиеся в гиперпространстве), starships (звездолеты).

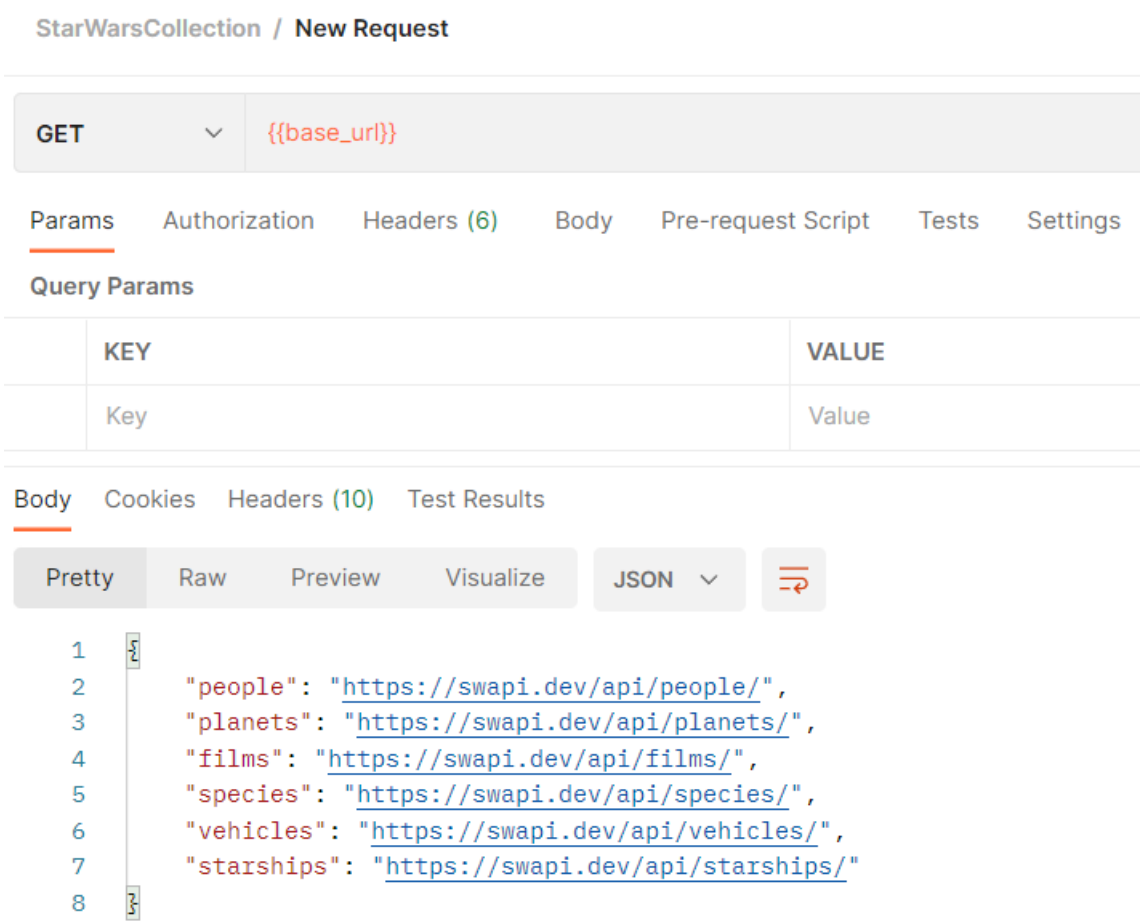

Рисунок 7 — Результат запроса {{base_url}}

Параметры запроса

Параметры можно добавлять к запросу, заполнив значения полей «KEY» и «VALUE» в «Query Params» на вкладке «Params».

Для SWAPI доступны следующие параметры:

• page указывает номер нужной страницы ответа от сервера (если в ответе много информации, он делится на несколько страниц);

• format указывает кодировку отображения данных (в SWAPI доступны кодировки JSON и wookiee);

• search используется для нахождения ресурса с конкретным названием или поиска множества объектов с некоторой подстрокой в названии.

Создание тестов

Для проверки ответа сервера на запрос с помощью тестов нужно перейти

на вкладку «Tests». Скрипт тестов разрабатывается на языке JavaScript, но также можно использовать готовые фрагменты кода — **сниппеты**. Сниппеты располагаются справа от окна кода тестов под названием «SNIPPETS» (рисунок 8).

Некоторые из доступных сниппетов:

– Status code is 200 (запрос выполнен успешно);

– Response time is less than 200ms (время ответа меньше 200 мс);

– Response body: Contains string (ответ содержит строку).

При нажатии на название сниппета в окне формируется фрагмент кода, который можно менять по своему усмотрению, например, изменить время на 1000 мс или задать искомую строку.

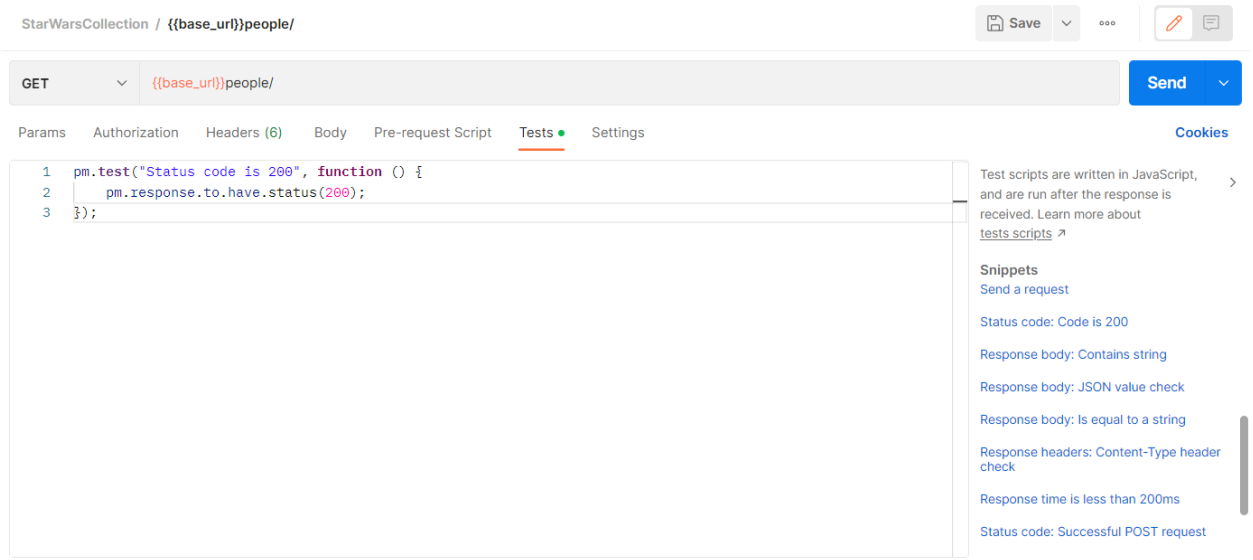

Рисунок 8 — Использование сниппетта для написания теста

После выполнения запроса во вкладке «Test Results» можно увидеть результат теста (рисунок 9).

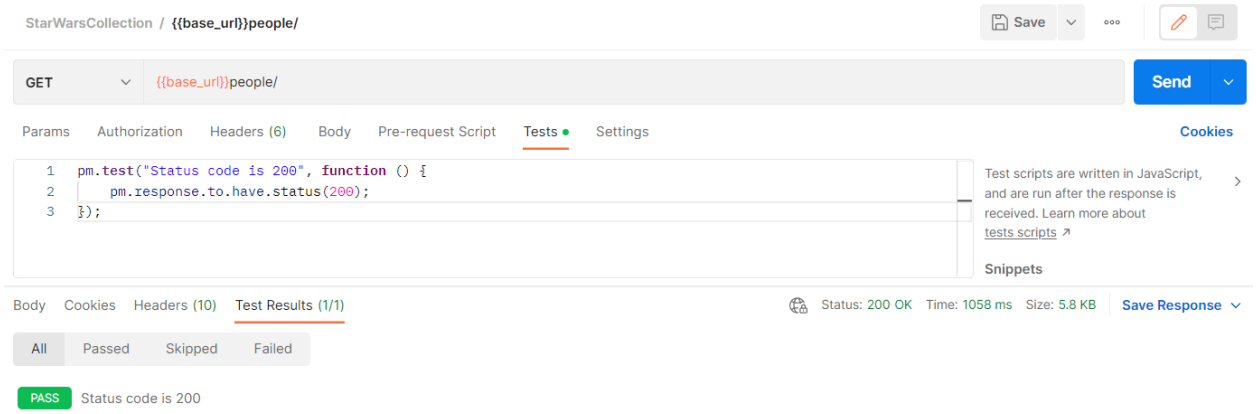

Рисунок 9 — Результат теста

Запуск тестирования коллекции запросов

Для запуска набора запросов вместе с тестами в определенной последовательности можно использовать Collection Runner. Есть два способа запуска набора запросов:

- выбрать коллекцию и нажать на кнопку «Run»;
- нажать на кнопку «Runner» в нижнем правом углу, после чего перетащить в появившееся окно название выбранной коллекции.

После этого нужно оставить галочки на выбранных для тестирования запросах и поменять их местами в случае необходимости. Затем необходимо нажать на кнопку запуска для коллекции.

Практическая часть

Для выполнения работы необходим доступ к сети Интернет.

1. Изучите теоретическую часть лабораторной работы.

2. Запустите Postman.

3. Создайте коллекцию для запросов к SWAPI.

4. Создайте переменную коллекции с именем base_url и значением https://swapi.dev/api/. Далее используйте эту переменную в запросах к SWAPI.

5. Запросите информацию о планетах. Выясните, что означают атрибуты rotation_period и orbital_period.

6. Запросите подробную информацию обо всех звездолетах.

7. Теперь выполните тот же запрос, но с параметром, позволяющим

получить вторую страницу ответа от сервера. Проверьте с помощью теста, что запрос выполнен успешно.

8. Запросите все звездолеты, в названии или модели которых есть подстрока Star или star. Проверьте с помощью теста, пришел ли ответ за 300 мс.

9. Запросите подробную информацию обо всех звездолетах на языке wookiee.

10. Проверьте с помощью теста, что ответ от сервера содержит подстроку warcworawawhoohurracao.

11. Запустите вместе все три последних запроса с тестами.

12. Выберите персонажа из таблицы 2 согласно своему номеру варианта задания. Дальнейшие пункты выполняются для заданного персонажа. По согласованию с преподавателем возможен выбор какого-либо другого персонажа, представленного в SWAPI.

Таблица 2 — Варианты персонажей

Вариант	**Персонаж**	**Вариант**	**Персонаж**
1	Luke Skywalker	11	Jar Jar Binks
2	Lando Calrissian	12	Padmé Amidala
3	Leia Organa	13	Gasgano
4	Wedge Antilles	14	Shmi Skywalker
5	Yoda	15	Han Solo
6	Darth Vader	16	Qui-Gon Jinn
7	Obi-Wan Kenobi	17	Arvel Crynyd
8	Ayla Secura	18	Palpatine
9	Chewbacca	19	Anakin Skywalker
10	C-3PO	20	R2-D2

13. Найдите персонажа с помощью запроса с параметром. Узнайте рост, вес и цвет глаз своего персонажа.

14. Определите с помощью запроса название родной планеты своего персонажа. Определите следующие сведения об этой планете: длительность суток в часах, длительность года в сутках, климат.

15. Узнайте, в скольких фильмах участвовал персонаж.

16. Определите с помощью запроса родной язык у персонажа, если о нем есть сведения.

17. Определите с помощью запроса максимальную скорость в атмосфере для одного из транспортных средств персонажа.

18. Создайте с помощью сниппетов по 3 теста для каждого запроса о персонаже. Запустите все тесты для запросов о персонаже вместе и убедитесь в их успешном завершении.

Контрольные вопросы

1. Для чего предназначена платформа Postman?
2. Что представляет собой REST?
3. Какие HTTP-методы наиболее часто используются в REST-запросах?
4. В чем заключается отличительная особенность метода GET?
5. Что такое API?
6. Что объединяют в себе коллекции в Postman?
7. Как можно использовать переменную коллекции в Postman?
8. Какими способами можно создать запрос GET параметрами?
9. Зачем нужны сниппеты?
10. Как запустить сразу несколько запросов вместе с тестами?
11. Перечислите информацию, необходимую тестировщику для начала тестирования API.
12. В чем состоит разница методов POST и PUT?