GeekBrains

REST API

Тестирование API

Урок 2

Оглавление

- На этом уроке
- Что такое REST API
  - HTTP
    - Пример взаимодействия:
  - Ресурсы и методы
  - HTTP-коды
  - Headers
  - Request и Response
  - Кеширование данных
    - Различные виды кеширования
      - Приватный (private) кеш браузера
      - Общий (shared) прокси-кеш
      - Цели кеширования
      - Свежесть сохраненной копии
    - JSON
      - Строка
      - Числовые значения
      - Логические значения
      - Нуль
      - Объект
      - Массив
    - Postman
      - Интерфейс
        - Боковая панель
          - Коллекции
          - Создание своей первой коллекции
          - API
          - Окружение
          - Мок-серверы
          - Мониторы
          - История
      - Раздел меню
      - Основное меню
      - Нижний колонтитул
      - Правая боковая панель
  - Автоматизация запросов в Postman на примере готовых сниппетов

Pre-request Script

Tests

Контрольные вопросы

Что можно почитать ещё?

Используемая литература

На этом уроке

1. Изучим REST API, подробнее поговорим про методы, коды ответов, запросы, ответы.
2. Разберем формат общения клиента и сервера — JSON.
3. Научимся тестировать API с помощью Postman.

Что такое REST API

REST API — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful.

Термин состоит из двух аббревиатур, которые расшифровываются следующим образом:

> **API (Application Programming Interface)** — это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой. Это позволяет разработчикам расширять функциональность своего продукта и связывать его с другими.
>
> **REST (Representational State Transfer)** — это архитектурный стиль взаимодействия компонентов распределённой системы в компьютерной сети. Проще говоря, REST определяет стиль взаимодействия (обмена данными) между разными компонентами системы, каждая из которых может физически располагаться в разных местах. Этот архитектурный стиль представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы.

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространённый способ организации API. Он вытеснил ранее пользовавшиеся популярностью способы SOAP и WSDL.

У RESTful нет единого стандарта работы: его называют «архитектурным стилем» для операций по работе с сервером. Разработал этот стиль Рой Филдинг в своей докторской диссертации 2000 года. Основная предпосылка заключается в том, что разработчики используют стандартные методы HTTP, GET, POST, PUT и DELETE для запроса и изменения ресурсов, представленных URI в интернете.

HTTP

Этот протокол нам известен из предыдущей лекции, он описывает взаимодействие между двумя компьютерами (клиентом и сервером), построенное на базе сообщений, называемых «запрос» (Request) и «ответ» (Response). Каждое сообщение состоит из трёх частей: стартовая строка, заголовки и тело. При этом обязательной является только стартовая строка.

Стартовые строки для запроса и ответа имеют различный формат — нам интересна только стартовая строка запроса, которая выглядит так:

```
METHOD URI HTTP/VERSION
```

METHOD — это как раз метод HTTP-запроса,

URI — идентификатор ресурса,

VERSION — версия протокола.

Заголовки — это набор пар «имя-значение», разделённых двоеточием. В заголовках передаётся различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришёл клиент (Referrer) и так далее.

Тело сообщения — это, собственно, передаваемые данные. В ответе передаваемыми данными, как правило, является html-страница, которую запросил браузер, а в запросе (например, в теле сообщения) передаётся содержимое файлов, загружаемых на сервер. Но, как правило, тело сообщения в запросе вообще отсутствует.

HTTP определяет множество методов запроса, которые указывают, какое желаемое действие выполнится для этого ресурса. Несмотря на то что их названия могут быть существительными, эти методы запроса иногда называются HTTP-глаголами. Каждый реализует свою семантику, но каждая группа команд разделяет общие свойства: так, методы могут быть безопасными, идемпотентными или кэшируемыми.

Метод GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

Пример взаимодействия:

Рассмотрим пример.

Запрос:

```
GET /index.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509
Firefox/3.0b5
Accept: text/html
Connection: close
```

Первая строка — это строка запроса, остальные — заголовки; тело сообщения отсутствует

Ответ:

```
HTTP/1.0 200 OK
Server: nginx/0.6.31
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close
```

... САМА HTML-СТРАНИЦА ... // это то, что будет в ответе на запрашиваемую страницу

Ресурсы и методы

Вернёмся к стартовой строке запроса:

```
METHOD URI HTTP/VERSION
```

И вспомним, что в ней присутствует такой параметр, как URI. Это расшифровывается как Uniform Resource Identifier — единообразный идентификатор ресурса. Ресурс — это, как правило, файл на сервере (пример URI в данном случае — '/styles.css'), но вообще ресурсом может являться и какой-либо абстрактный объект ('/blogs/webdev/' — указывает на блок «Веб-разработка», а не на конкретный файл).

Тип HTTP-запроса (также называемый HTTP-методом) указывает серверу на то, какое действие мы хотим произвести с ресурсом. Изначально (в начале 90-х) предполагалось, что клиент может хотеть от ресурса только одного — получить его, однако сейчас по протоколу HTTP можно создавать посты, редактировать профиль, удалять сообщения и многое другое. И эти действия сложно объединить термином «получение».

Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

GET — получение ресурса

POST — создание ресурса

PUT — обновление ресурса

DELETE — удаление ресурса

Обратите внимание на тот факт, что спецификация HTTP не обязывает сервер понимать все методы (которых на самом деле гораздо больше, чем 4) — обязателен только GET. Также эта спецификация не указывает серверу, что он должен делать при получении запроса с тем или иным методом. А это значит, что сервер в ответ на запрос DELETE /index.php HTTP/1.1 не обязан удалять страницу index.php на сервере, так же как на запрос GET /index.php HTTP/1.1 не обязан возвращать вам страницу index.php, он может её удалить, например.

Какие ещё методы существуют:

HEAD — запрашивает ресурс так же, как и метод GET, но без тела ответа.

CONNECT — устанавливает «туннель» к серверу, определённому по ресурсу.

OPTIONS — используется для описания параметров соединения с ресурсом.

TRACE — выполняет вызов возвращаемого тестового сообщения с ресурса.

PATCH — используется для частичного изменения ресурса.

HTTP-коды

Код ответа (состояния) HTTP показывает, был ли успешно выполнен определённый HTTP запрос. Коды сгруппированы в 5 классов:

- Информационные 100 – 199
- Успешные 200 – 299
- Перенаправления 300 – 399
- Клиентские ошибки 400 – 499
- Серверные ошибки 500 – 599

1xx: Information

100: Continue

2xx: Success

200: OK

201: Created

202: Accepted

204: No Content

3xx: Redirect

301: Moved Permanently

307: Temporary Redirect

4xx: Client Error

400: Bad Request

401: Unauthorized

403: Forbidden

404: Not Found

5xx: Server Error

500: Internal Server Error

501: Not Implemented

502: Bad Gateway

503: Service Unavailable

504: Gateway Timeout

Headers

Заголовки HTTP позволяют клиенту и серверу отправлять дополнительную информацию с HTTP запросом или ответом. В HTTP-заголовке содержится не чувствительное к регистру название, а после (:) — значение. Пробелы перед значением игнорируются.

Пользовательские собственные заголовки исторически использовались с префиксом X, но это соглашение было объявлено устаревшим в июне 2012 года из-за неудобств, вызванных тем, что нестандартные поля стали стандартом в RFC 6648; другие перечислены в реестре IANA, исходное содержимое которого было определено в RFC 4229. IANA также поддерживает реестр предлагаемых новых заголовков HTTP.

HTTP-заголовки сопровождают обмен данными по протоколу HTTP. Они могут содержать описание данных и информацию, необходимую для взаимодействия между клиентом и сервером. Заголовки и их статусы перечислены в реестре IANA, который постоянно обновляется.

Заголовки могут быть сгруппированы по следующим контекстам:

- Основные заголовки применяются как к запросам, так и к ответам, но не имеют отношения к данным, передаваемым в теле.
- Заголовки запроса содержат больше информации о ресурсе, который нужно получить, или о клиенте, запрашивающем ресурс.
- Заголовки ответа (en-US) содержат дополнительную информацию об ответе, например, его местонахождение, или о сервере, предоставившем его.
- Заголовки сущности содержат информацию о теле ресурса, например, его длину содержимого или тип MIME.

Существует множество заголовков запроса. Их можно разделить на несколько групп:

- Основные заголовки (General headers), например, Via (en-US), относящиеся к сообщению в целом.
- Заголовки запроса (Request headers), например, User-Agent, Accept-Type, уточняющие запрос (как, например, Accept-Language), придающие контекст (как Referer) или накладывающие ограничения на условия (like If-None).
- Заголовки сущности, например, Content-Length, относящиеся к телу сообщения. Как легко понять, они отсутствуют, если у запроса нет тела.

Request и Response

HTTP-сообщения — это обмен данными между сервером и клиентом. Есть два типа сообщений: запросы, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера, и ответы от сервера.

HTTP-запросы — это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера.

```
GET /api/v1/products HTTP/1.1
```

```
Accept: text/html
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Host: api.youla.io
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/102.0.5005.61 Safari/537.36
```

Выше представлен запрос на сайт. Давайте разберём, из чего же состоит запрос:

Метод HTTP, глагол (например, GET, PUT или POST) или существительное (например, HEAD или OPTIONS), описывающие требуемое действие. Например, GET указывает, что нужно доставить некоторый ресурс, а POST означает отправку данных на сервер (для создания или модификации ресурса или генерации возвращаемого документа).

Цель запроса, обычно **URL**, или абсолютный путь протокола, порт и домен обычно характеризуются контекстом запроса. Формат цели запроса зависит от используемого HTTP-метода.

Версия HTTP, определяющая структуру оставшегося сообщения, указывает, какую версию предполагается использовать для ответа.

Строка ответа HTTP, называемая строкой статуса, содержит следующую информацию:

- Версию протокола, обычно HTTP/1.1.
- Код состояния (status code), показывающий, был ли запрос успешным. Примеры: 200, 404 или 302.
- Пояснение (status text). Краткое текстовое описание кода состояния, помогающее пользователю понять сообщение HTTP.

Например:

```
HTTP/1.1 304 Not Modified
Server: nginx/1.18.0 (Ubuntu)
Date: Sun, 09 Jan 2022 18:45:37 GMT
Last-Modified: Tue, 28 Dec 2021 19:24:52 GMT
Connection: keep-alive
ETag: "61cb6484-7ce"
```

Response:

```
HTTP/1.1 200 OK
Date: Sun, 13 Mar 2022 14:31:19 GMT
X-Powered-By: Express
Content-Length: 113
Content-Type: text/plain; charset=utf-8
Connection: keep-alive

[{"id":"622e0036b9d8210008ee3039","itemId":"510a0d7e-8e83-4193-b483-e27e09ddc34d","qu
antity":1,"unitPrice":15.0}]
```

Кеширование данных

Производительность веб-сайтов и приложений можно значительно повысить за счёт повторного использования ранее полученных ресурсов. Веб-кеши сокращают задержку и снижают сетевой трафик, уменьшая тем самым время, необходимое для отображения ресурсов. Благодаря HTTP-кэшированию сайты становятся более отзывчивыми.

Различные виды кеширования

Техника кэширования заключается в сохранении копии полученного ресурса для возврата этой копии в ответ на дальнейшие запросы. Запрос на ресурс, уже имеющийся в веб-кеше, перехватывается, и вместо обращения к исходному серверу выполняется загрузка копии из кеша. Таким образом, снижается нагрузка на сервер, которому не приходится самому обслуживать всех клиентов, и повышается производительность — кеш ближе к клиенту, и ресурс передаётся быстрее. Кэширование является основным источником повышения производительности веб-сайтов. Однако кеш надо правильно сконфигурировать: ресурсы редко остаются неизменными, так что копию требуется хранить только до того момента, как ресурс изменился, но не дольше.

Существует несколько видов кешей, которые можно разделить на две основные категории:

- приватные кеши;
- кеши совместного использования.

В кешах совместного использования (shared cache) хранятся копии, которые могут направляться разным пользователям. Приватный кеш (private cache) предназначен для отдельного пользователя. Здесь будет говориться в основном о кешах браузеров и прокси, но существуют также кеши шлюзов, CDN, реверсные прокси-кэши и балансировщики нагрузки, разворачиваемые на серверах для повышения надёжности, производительности и масштабируемости веб-сайтов и веб-приложений.

Приватный (private) кеш браузера

Приватный кеш предназначен для отдельного пользователя. Вы, возможно, уже видели параметры кэширования в настройках своего браузера. Кеш браузера содержит все документы, загруженные пользователем по HTTP. Он используется для доступа к ранее загруженным страницам при навигации назад/вперёд, позволяет сохранять страницы или просматривать их код, не обращаясь лишний раз к серверу. Кроме того, кеш полезен при отключении от сети.

Общий (shared) прокси-кеш

Кеш совместного использования — это кеш, который сохраняет ответы, чтобы их потом могли использовать разные пользователи. Например, в локальной сети вашего провайдера или компании может быть установлен прокси, обслуживающий множество пользователей, чтобы можно было повторно использовать популярные ресурсы, сокращая тем самым сетевой трафик и время ожидания.

Цели кеширования

Кэширование в HTTP не является обязательным, однако в большинстве случаев бывает полезно повторно использовать ранее сохранённые ресурсы. Тем не менее стандартные кеши HTTP обычно способны кэшировать только ответы на запросы методом GET, а другие отклоняют.

Первичный ключ состоит из метода запроса и запрашиваемого URI (зачастую используется только URI, поскольку целью кэширования являются только GET-запросы). Вот примеры того, что обычно записывается в кеш:

- Успешно загруженные ресурсы: ответ 200 OK на запрос методом GET HTML-документов, изображений или файлов.
- Постоянные перенаправления: ответ 301 Moved Permanently («перемещено навсегда»).
- Сообщения об ошибках: ответ 404 Not Found («не найдено»).
- Неполные результаты: ответ 206 Partial Content («частичное содержимое»).
- Ответы на запросы, отличные от GET, если есть что-либо, подходящее для использования в качестве ключа кеша.

Свежесть сохраненной копии

Запись в кеше может также состоять из множества ответов, различаемых по вторичному ключу, если при формировании ответа производится согласование данных.

Однажды попав в кеш, ресурс теоретически может храниться там вечно. Однако, поскольку объём хранилища конечен, записи периодически приходится оттуда удалять. Этот процесс называют вытеснением данных из кеша (cache eviction). Кроме того, ресурсы могут изменяться на сервере, поэтому кеш требуется обновлять. Поскольку HTTP является клиент-серверным протоколом, сервера не могут сами обращаться к кешам и клиентам при изменении ресурса — им необходимо договориться о сроке действия сохранённой копии. До его истечения ресурс считается свежим (fresh), после устаревшим (stale).

JSON

JSON (сокр. от англ. JavaScript Object Notation) — простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером.

JSON часто используется для сериализации структурированных данных и обмена ими по сети, обычно между сервером и веб-приложениями.

JSON состоит из следующих типов данных:

- Строка
- Число
- Логическое значение
- Нуль
- Объект
- Массив

Строка

Строка в JSON состоит из символов Юникода, начинающихся с обратной косой черты (\), либо латинские буквы остаются без изменений:

```
{ "name" : "Jones" }
```

Числовые значения

```
{
 "number_1" : 210,
 "number_2" : 215,
 "number_3" : 21.05,
 "number_4" : 10.05
 }
```

Логические значения

Булевы значения обозначаются как true или false. Логические значения не заключаются в кавычки и рассматриваются как строковые значения.

```
{ "AllowPartialShipment" : false }
```

Нуль

Нуль — это пустое значение. Если ключу не присвоено никакого значения, его можно считать значением Null.

```
{ "Special Instructions" : null }
```

Объект

Тип данных объекта JSON — это набор пар имени или значения, вставленных между {} (фигурными скобками). Ключи должны быть строками и должны быть уникальными, разделёнными запятыми.

```
{
 "Influencer" : {
    "name" : "Jaxon" ,
    "age" : "42" ,
    "city" , "New York"
 }
}
```

Массив

Тип данных массива — это упорядоченный набор значений. В JSON значения массива должны быть строкой, числом, объектом, массивом, логическим значением или значением Null.

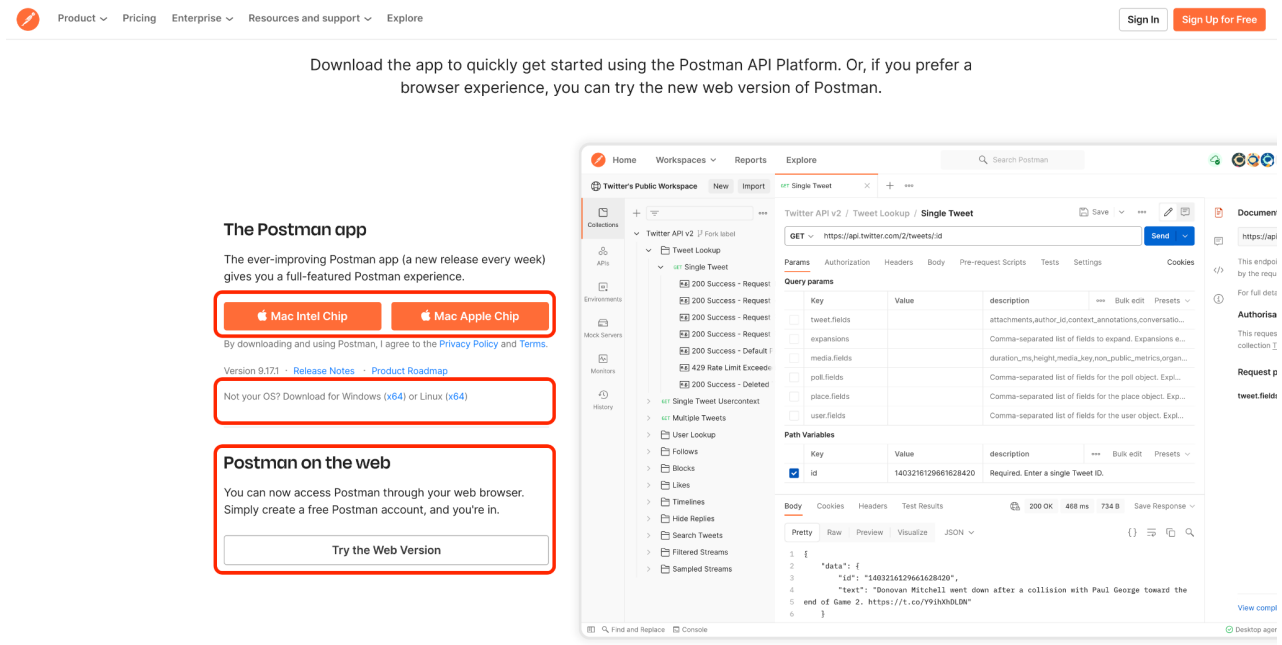
```
{
 "Influencers" :
    [
     {
      "name" : "Jaxon",
      "age" : 42,
      "Works At" : "Tech News"
     },

     {
```

```
      "name" : "Miller",
      "age" : 35
      "Works At" : "IT Day"
    }
  ]
}
```

Postman

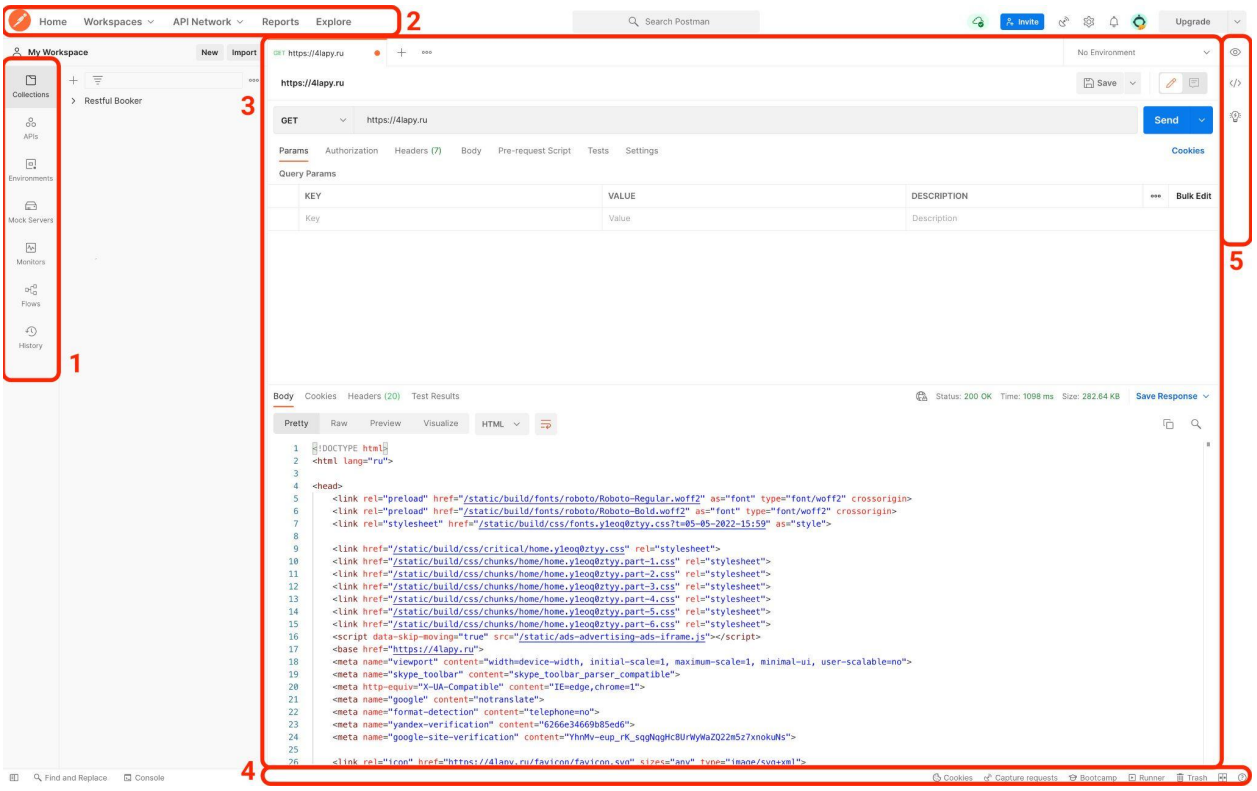
Postman — инструмент с открытым исходным кодом, который используется для тестирования REST API.

Для установки необходимо перейти на [Postman | Get Started for Free] и выбрать нужную платформу среди Mac, Windows или Linux. Далее скачать и установить.



Инструмент также доступ в веб-версии, что довольно удобно. Однако, лучше всего использовать десктопную версию.

Интерфейс



1. Боковая панель
2. Раздел меню
3. Основное меню
4. Нижний колонтитул
5. Правая боковая панель

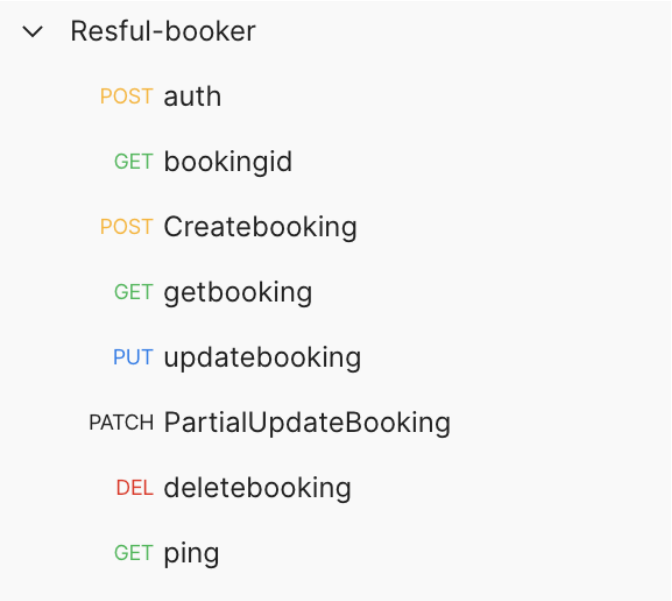
Боковая панель

Боковая панель Postman обеспечивает доступ к основным элементам Postman:

- Коллекции
- API
- Окружение
- Мок-серверы
- Мониторы
- История

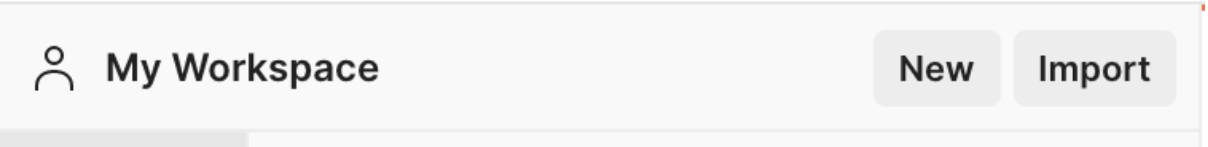
Коллекции

Коллекции — это группа сохранённых запросов. Каждый запрос, который вы отправляете в Postman, отображается на вкладке «История» боковой панели. В небольших масштабах повторное использование запросов через раздел истории удобно. По мере роста вашего использования Postman может потребоваться много времени, чтобы найти конкретный запрос в вашей истории. Вместо того чтобы прокручивать раздел истории, вы можете сохранить все свои запросы в виде группы для более удобного доступа.

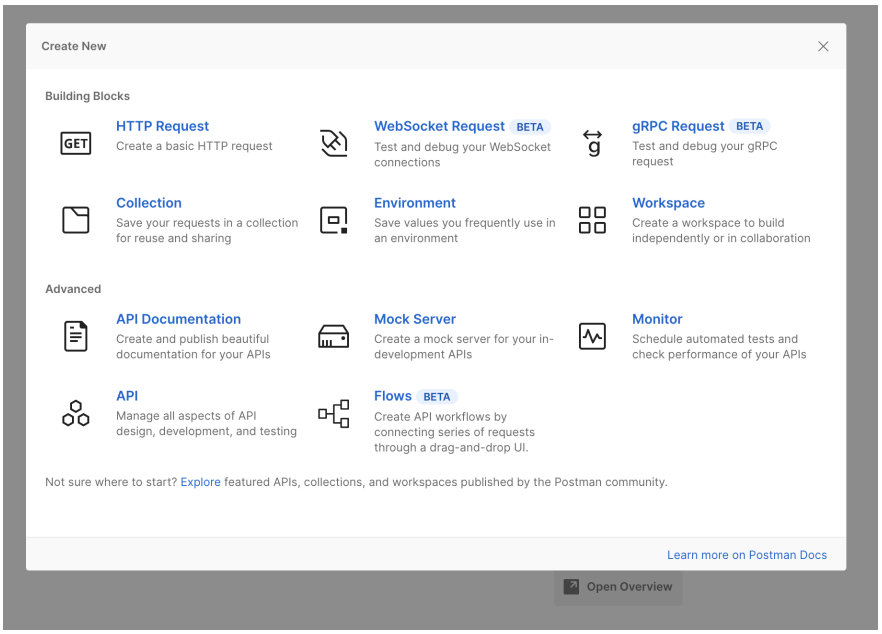


Создание своей первой коллекции

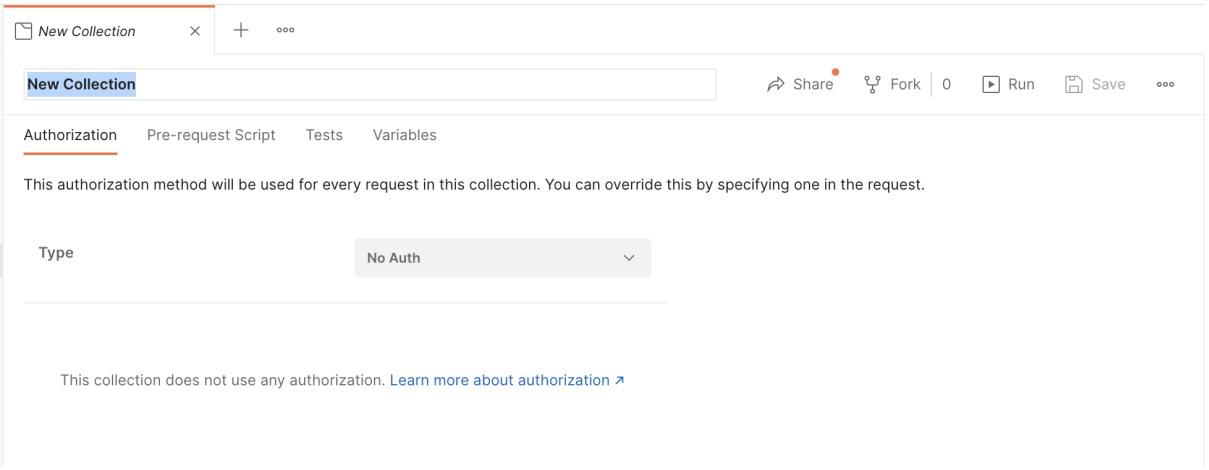
Чтобы создать новую коллекцию, выберите **New**.



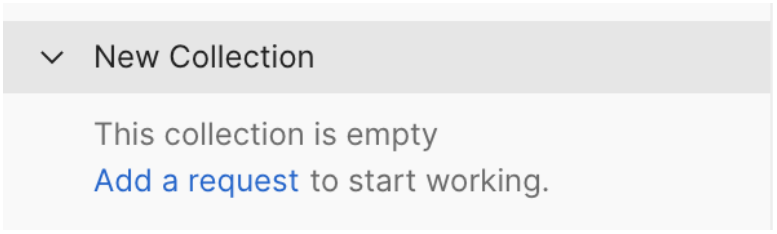
Затем выберите **Collection**



Создайте новую коллекцию, введя название коллекции. Введите имя коллекции, а затем выберите **Save** .



Теперь вы можете увидеть все свои коллекции в разделе «Коллекции» на боковой панели, как показано ниже:

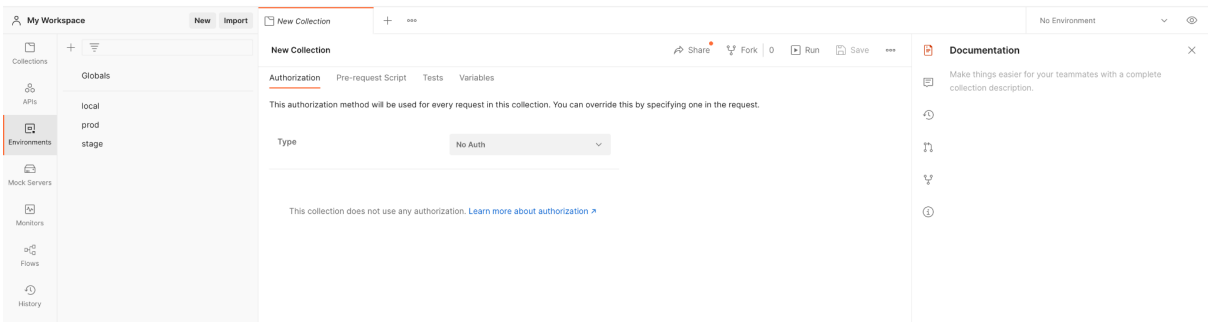


API

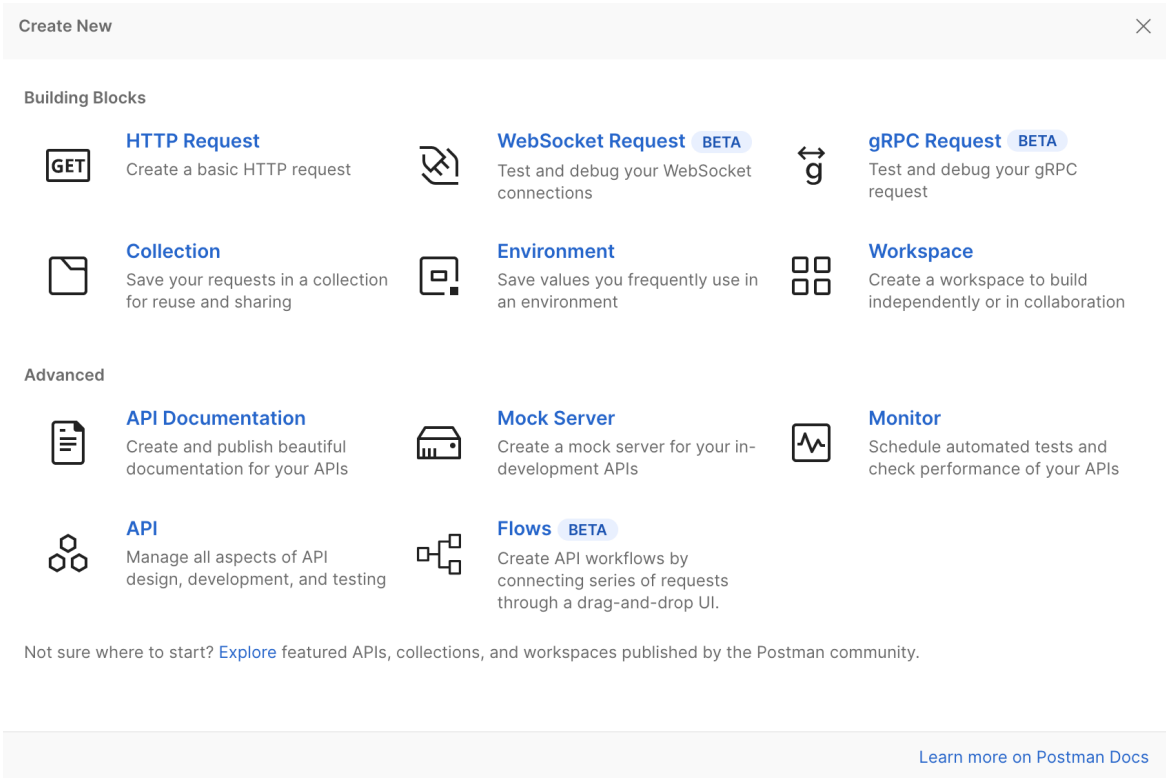
Это API Builder. API Builder поддерживает запуск модульных тестов, они являются скорее инструментом разработчика, а не тестировщика. Чтобы начать использовать API Builder, вы можете создать новый API в своей рабочей области. Вы также можете переименовать или удалить существующие API.

Окружение

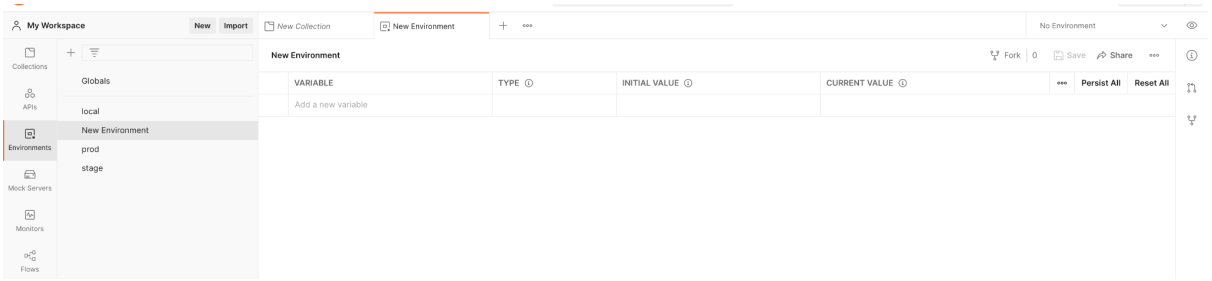
Окружение — это набор переменных, которые вы можете использовать в своих запросах Postman. Вы можете использовать среды для группировки связанных наборов значений и управления доступом к общим данным Postman, если вы работаете в команде.



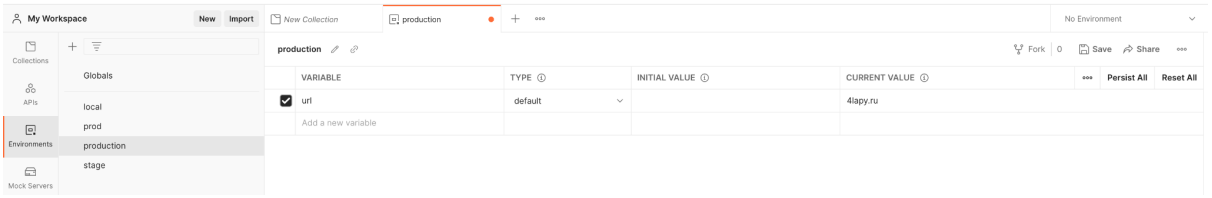
На текущий момент добавлены три переменные local, prod, stage. Добавим ещё одно окружение. Выберем **NEW** → **Environment**.



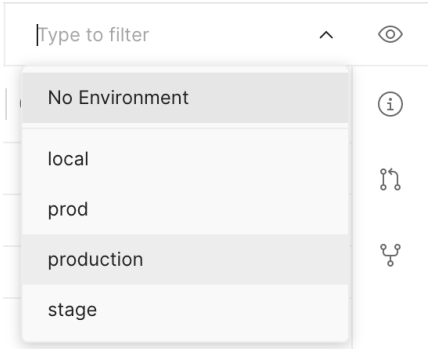
Затем, можно назвать окружение, на текущий момент наше окружение называется New Environment.



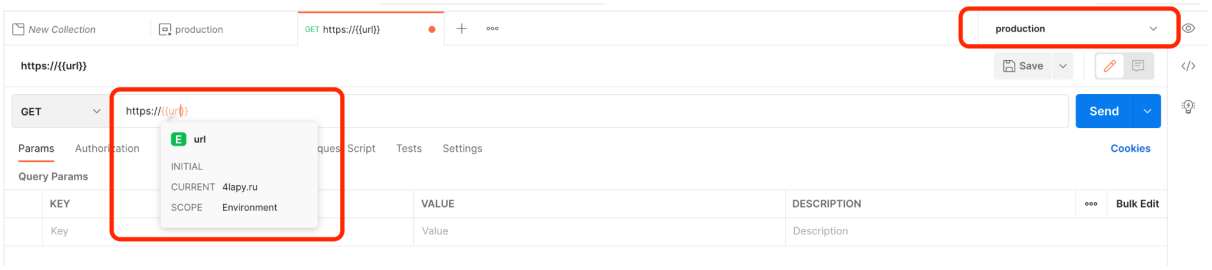
Создаём переменную **url** и зададим её значение 4lapy.ru:



Важно не забыть сохранить окружение, чтобы оно было доступно для использования.



Чтобы использовать значение переменной среды в запросе, укажите его по имени, заключённому в двойные фигурные скобки:



Мок-серверы

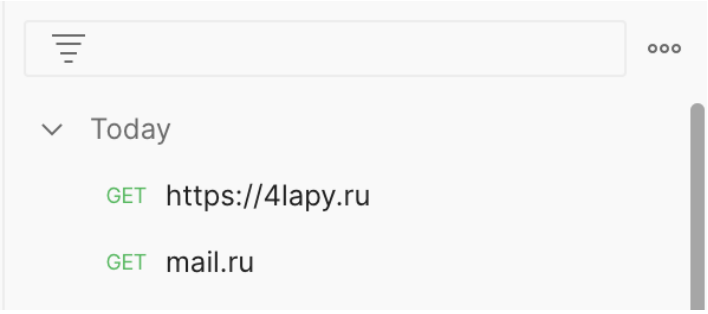
В Postman можно делать запросы, которые возвращают фиктивные данные, определённые в Postman, если у вас нет готового производственного API или вы пока не хотите запускать свои запросы с реальными данными. Добавляя фиктивный сервер в свою коллекцию и добавляя примеры к своим запросам, вы можете имитировать поведение реального API.

Мониторы

Этот раздел необходим для автоматизирования прогонов. Здесь имеется возможность планирования прогонов и получение отчётов о результатах прогонов.

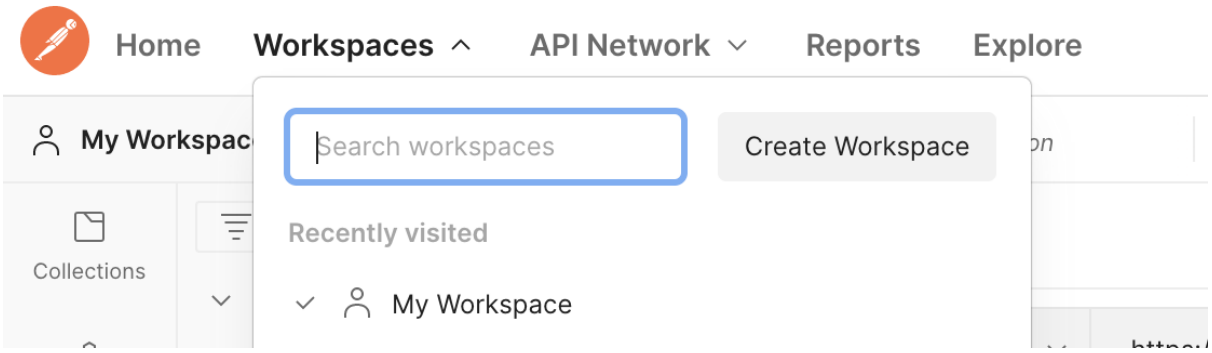
История

В разделе с историей отображаются все выполненные ранее запросы через Postman.



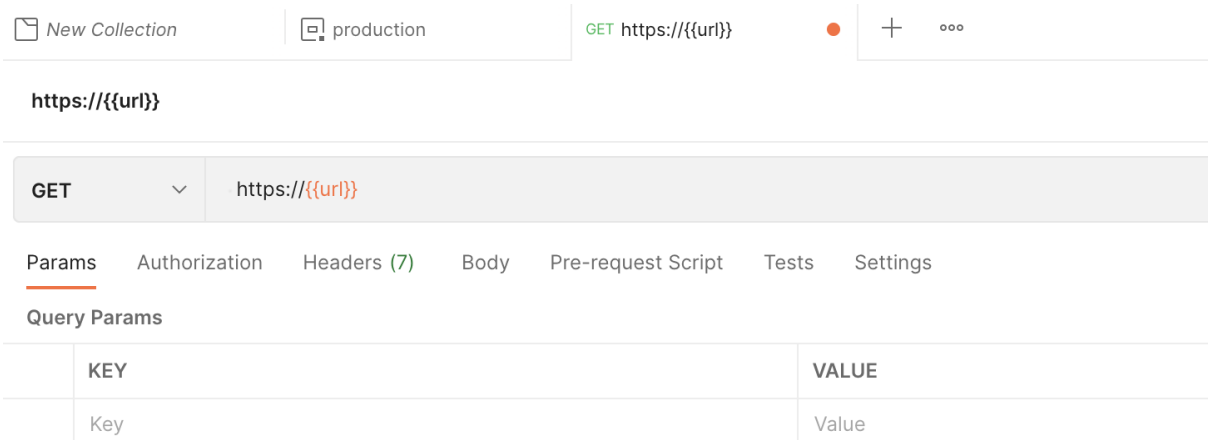
Раздел меню

В меню находятся разделы, которые помогают настраивать рабочее пространство.

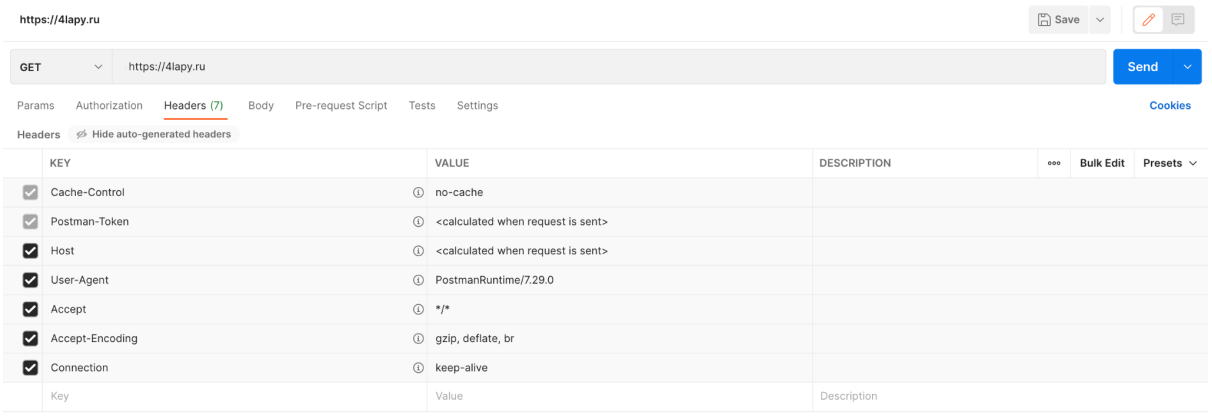


Основное меню

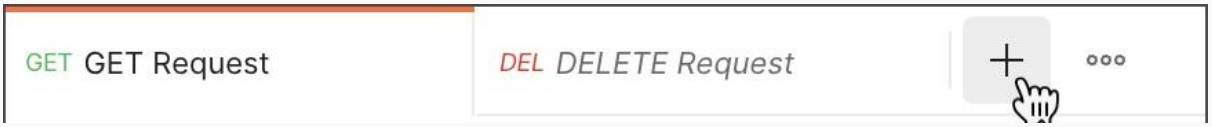
Это основное рабочее пространство. В этом разделе отображаются вкладки. Они позволяют организовывать работу и переключаться между запросами.



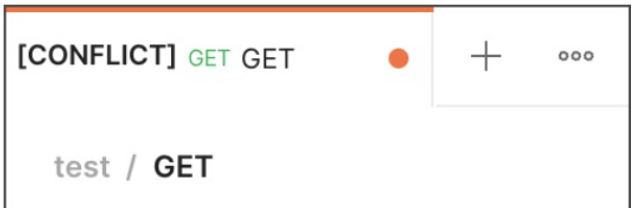
Также в этом разделе предоставляется возможность работать с самими запросами и ответами:



Чтобы открыть новую вкладку, выберите + в середине экрана. Вы также можете нажать ⌘+T или Ctrl+T , чтобы открыть новую вкладку.



Если вы открываете запрос и не редактируете и не отправляете его, а затем открываете другой запрос, вторая вкладка заменяет первую вкладку. Когда вкладка находится в режиме предварительного просмотра, она отображается курсивом.
Вы можете указать, будет ли Postman открывать запросы в новых вкладках. Выберите значок настроек,  чтобы открыть «Настройки». В разделе «Запросы» выберите «Всегда открывать запросы на новой вкладке», чтобы включить или отключить этот параметр.
Если на вкладке есть несохранённые изменения, Postman отображает точку рядом с названием вкладки. Выберите  («Сохранить»), чтобы сохранить изменения. Чтобы закрыть вкладку и отменить изменения, щёлкните значок закрытия  , а затем выберите «Не сохранять».
Также на вкладке отображается информация о конфликте, который необходимо устранить.



Нижний колонтитул

Раздел меню в нижней части экрана позволяет вам находить и заменять текст, открывать консоль, выбирать запросы и файлы cookie, а также многое другое.

Правая боковая панель

Справа располагаются инструменты, включая документацию, комментарии, фрагменты кода и множество другой полезной информации.

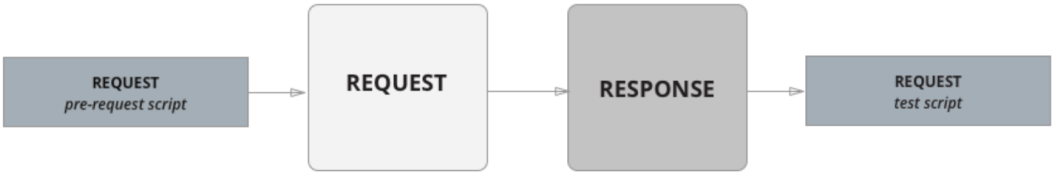
Автоматизация запросов в Postman на примере готовых сниппетов

В Postman имеется возможность писать тесты к API, создавать запросы, которые могут содержать динамические параметры, передавать данные между запросами и многое другое. Вы можете добавить код JavaScript для выполнения во время двух событий в потоке:

- Перед отправкой запроса на сервер в виде сценария предварительного запроса.
- После получения ответа в виде тестового скрипта на вкладке Tests.

В Postman порядок выполнения скрипта для одного запроса выглядит так:

- Сценарий предварительного запроса, связанный с запросом, будет выполняться до отправки запроса.
- Тестовый сценарий, связанный с запросом, будет выполняться после отправки запроса.



Pre-request Script

Pre-request Script — это логика или часть кода, выполнение которого гарантировано до начала выполнения запроса. Это позволяет добавить динамическое поведение для выполнения запроса.

Здесь важно отметить, что сценарии предварительного запроса также могут применяться на уровне коллекции, что косвенно означает, что сценарий предварительного запроса будет применяться ко всем запросам, являющимся частью этой коллекции.

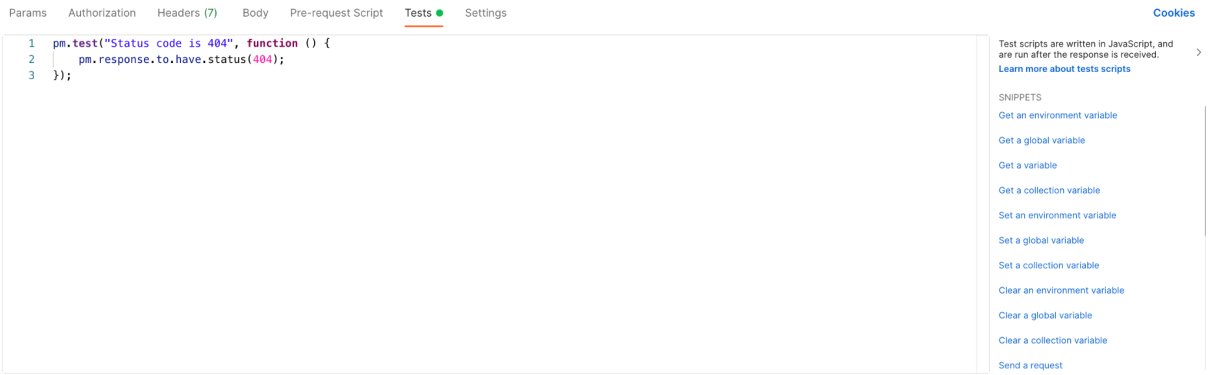
Простыми словами, пользование pre-request script может быть полезно, если вам необходимо передать какое-либо значение для переменной перед выполнением запроса.

Tests

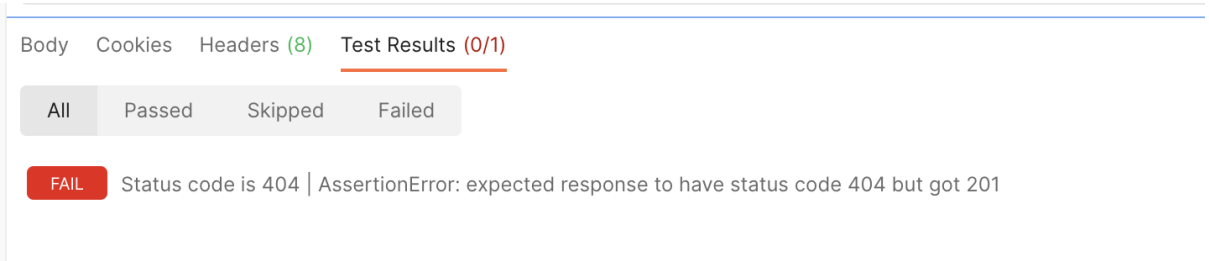
Вкладка Tests необходима для написания тестов, которые будут выполняться после получения ответа на запрос.

Код не обязательно писать с нуля. В Postman есть уже готовый список тестов для проверки API. Любой из них можно отредактировать под свои нужды для экономии времени.

Готовые скрипты (сниппеты) есть в списке справа. Там можно найти код для проверки всего ответа или его части, времени выполнения запроса и множества других вещей.



После отправки запроса мы будем видеть результат выполнения теста:



В нашем примере тест не прошёл проверку, так как ожидался статус 404, а пришёл 201.

Контрольные вопросы

1. Есть ли тело в GET запросе?
2. Какой статус код вернётся в ответе пользователю, если он будет запрашивать несуществующую страницу?

3. Какие базовые возможностями есть в инструменте Postman?

Что можно почитать ещё?

1. Список кодов состояния HTTP — Википедия
2. Understanding And Using REST APIs — Smashing Magazine
3. Introduction | Postman Learning Center

Используемая литература

1. Коды ответа HTTP
2. REST API: Best Practices, Concepts, Structure, and Benefits | AltexSoft
3. JSON