



Начнем с пары слов о том, что такое Postman. Это инструмент для работы с API, который позволяет тестировщику посылать запросы к сервисам и работать с их ответами. С его помощью можно протестировать бекэнд и убедиться, что он корректно работает.

Инструментов с аналогичным функционалом существует много. Я выбрала Postman, поскольку он самый популярный. Но у него есть и другие преимущества. Postman:

- интуитивно-понятен и простой в использовании, не требует какой-то сложной настройки или знания языков программирования;
- бесплатный;
- поддерживает разные API (REST, SOAP, GraphQL);
- расширяется под любые нужды с помощью Postman API;
- легко интегрируется в CI/CD с помощью Newman - консольной утилиты для запуска тестов;
- запускается на любых ОС;
- поддерживает ручное и автоматизированное тестирование;
- собрал вокруг себя большое комьюнити, где можно найти ответы на любые вопросы.

Тестировщику этот инструмент позволяет:

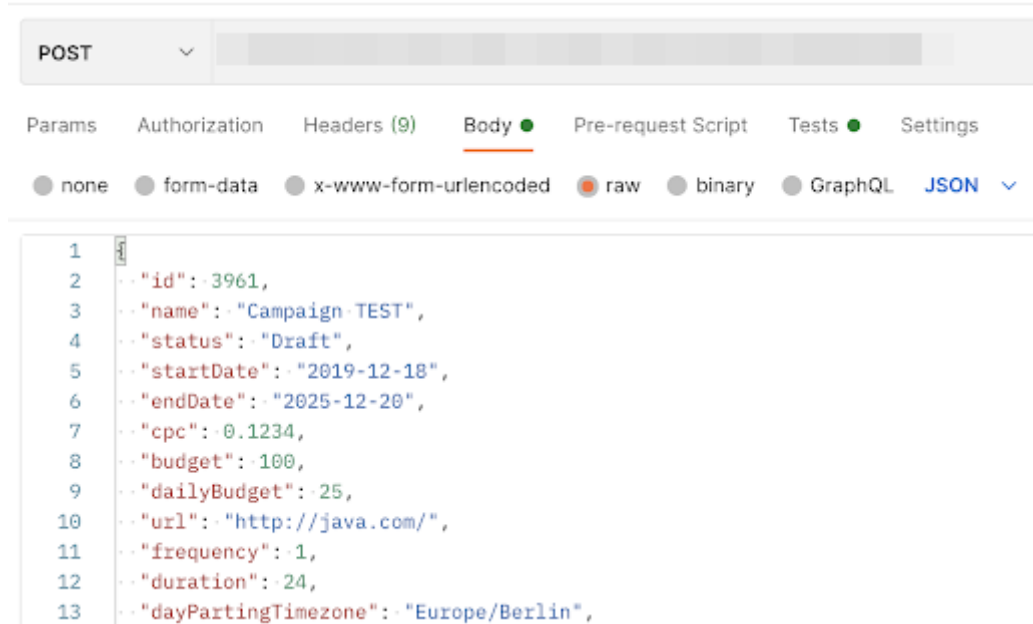
- отправлять запросы и получать ответы;
- сохранять запросы в папки и коллекции;
- изменять параметры запросов;
- изменять окружения (dev, test, production);
- выполнять автотесты, используя Collections runner, в том числе по расписанию;
- импортировать и экспортировать коллекции запросов и наборы тестов, чтобы обмениваться данными с коллегами.

Перейдем к сути.

На нашем проекте мы разрабатываем менеджер рекламных кампаний. У каждой кампании в нашей системе есть ряд полей - имя, описание, ID и креатив (реклама, которую видит пользователь). Для демонстрации возможностей Postman буду использовать запросы на создание и обновление кампании и креатива из “боевого” проекта.

Экспериментируем с запросом на обновление

Создадим самый простой запрос на обновление кампании.



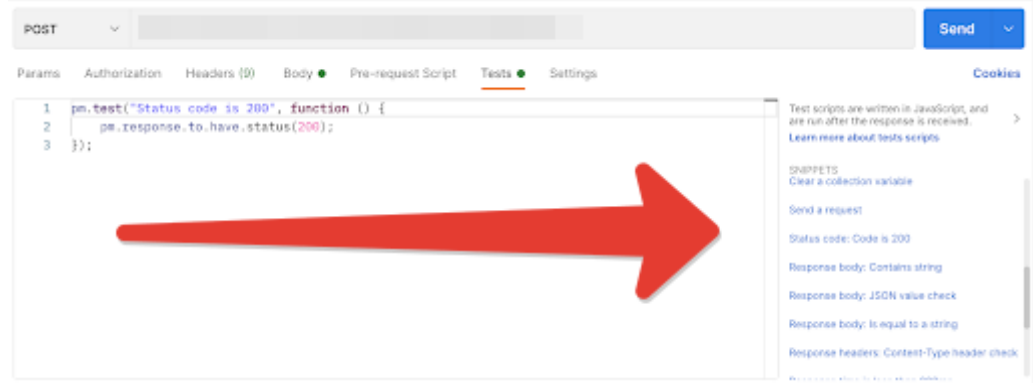
Простейший запрос на обновление кампании

При его успешном выполнении мы получим ответ 200 OK.

Напишем самый простой автотест, который будет это проверять. Для этого в интерфейсе Postman переходим на вкладку Tests. Код с этой вкладки будет 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.



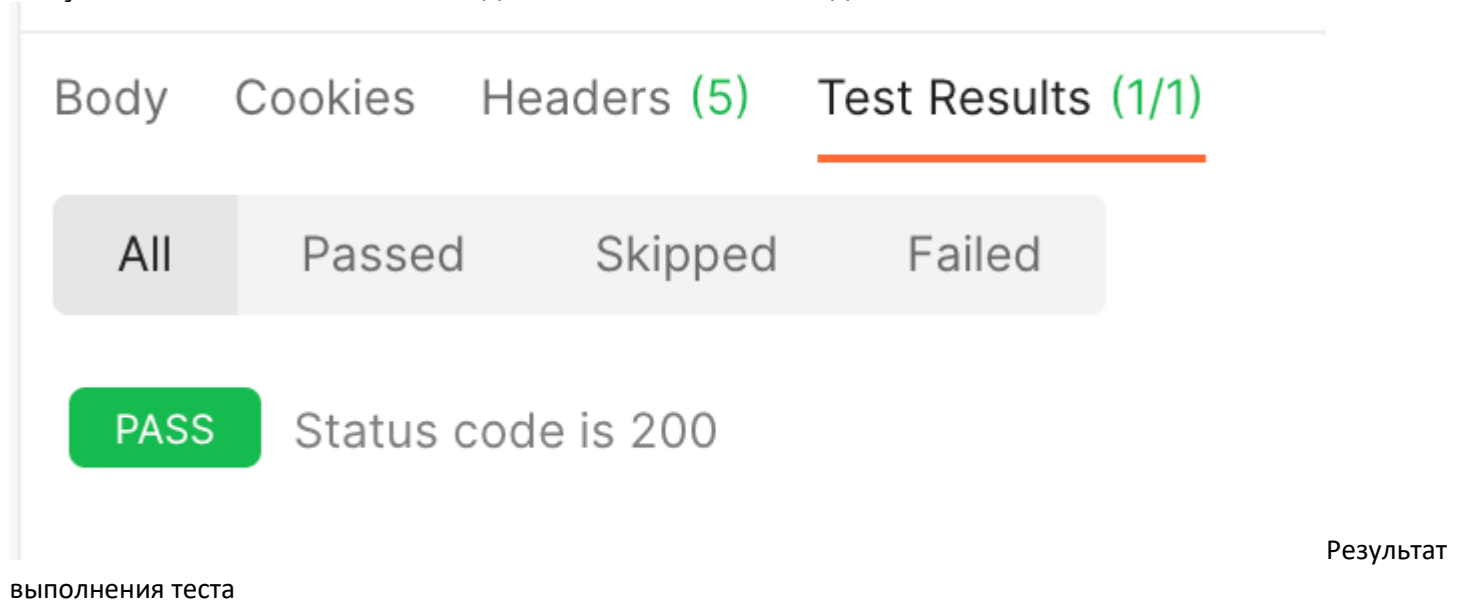
Список готовых скриптов справа

Выбираем сниппет, он добавляется на вкладку Tests. Этот код можно отредактировать - задать другое имя теста или ответ.

```
pm.test("Status code is 200", function () {  
  
    pm.response.to.have.status(200);  
  
});
```

Сохраняем код и отправляем запрос.

Результат можно найти на вкладке Test Results. Мы видим:



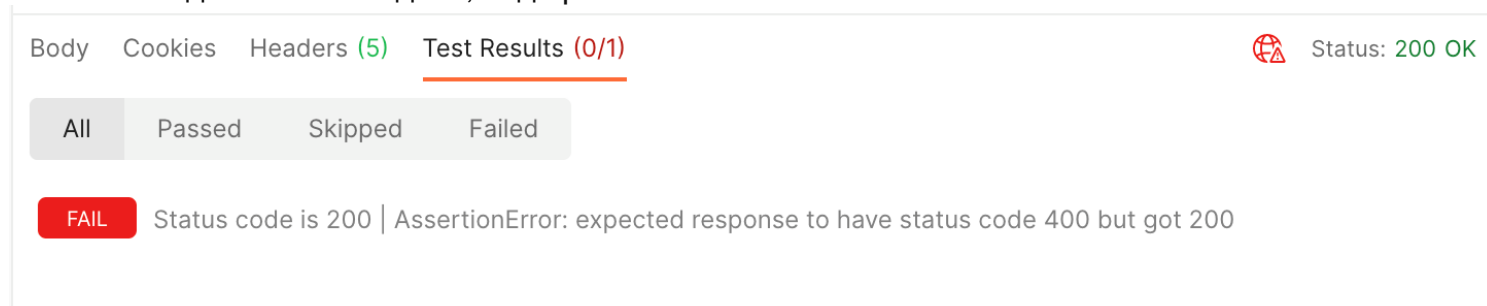
The screenshot shows the 'Test Results' tab selected in Postman. It displays a green 'PASS' button and the text 'Status code is 200'. The tabs 'Body', 'Cookies', 'Headers (5)', and 'Test Results (1/1)' are visible at the top, with 'Test Results (1/1)' being the active tab. Below the tabs are filters: 'All', 'Passed', 'Skipped', and 'Failed'. The 'All' filter is selected. The result is a green 'PASS' button followed by the text 'Status code is 200'. The text 'Результат выполнения теста' is written below the screenshot.

Давайте проверим, что тест действительно работает. Изменим код ответа на 400.

```
pm.test("Status code is 200", function () {  
  
    pm.response.to.have.status(400);  
  
});
```

Сохраним запрос еще раз запустим тест.

Вполне ожидаемо тест падает, ведь реальный ответ - 200 OK:



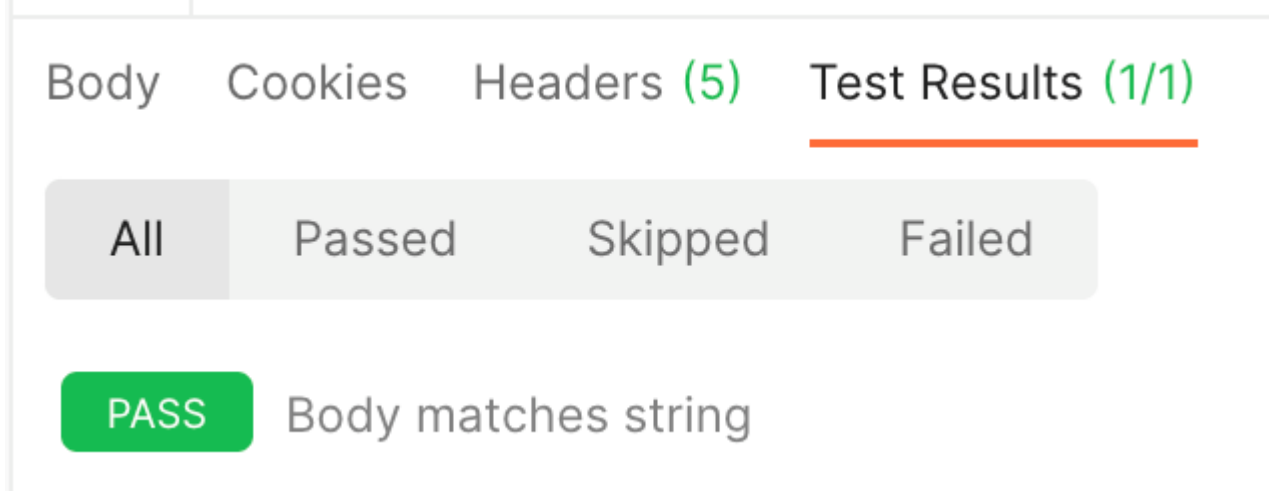
The screenshot shows the 'Test Results' tab selected in Postman. It displays a red 'FAIL' button and the text 'Status code is 200 | AssertionError: expected response to have status code 400 but got 200'. The tabs 'Body', 'Cookies', 'Headers (5)', and 'Test Results (0/1)' are visible at the top, with 'Test Results (0/1)' being the active tab. Below the tabs are filters: 'All', 'Passed', 'Skipped', and 'Failed'. The 'All' filter is selected. The result is a red 'FAIL' button followed by the text 'Status code is 200 | AssertionError: expected response to have status code 400 but got 200'. The text 'Результат выполнения теста' is written below the screenshot.

Аналогично можно проверить, что в теле ответа содержится определенная строка. В нашем случае посмотрим, есть ли там название рекламной кампании, которое мы передавали в запросе на обновление данных.

Ищем соответствующий сниппет и правим его под свою задачу:

```
pm.test("Body matches string", function () {  
  
    pm.expect(pm.response.text()).to.include("Campaign TEST");  
  
});
```

Сохраняем и отправляем запрос. Видим, что тест выполнен успешно.



Как и в прошлом примере, мы можем проверить работоспособность теста, исправив искомую строку:

```
pm.test("Body matches string", function () {  
  
    pm.expect(pm.response.text()).to.include("Campaign TEST1");  
  
});
```

Искать строку можно не во всем теле ответа, а в конкретном поле. Для проверки, что название кампании присутствует именно в поле Name, отредактируем другой сниппет:

```
pm.test("Your test name", function () {  
  
    var jsonData = pm.response.json();  
  
    pm.expect(jsonData.name).to.eql("Campaign TEST");  
});
```

```
});
```

Для одного запроса можно создать несколько тестов. Например, так:

```
pm.test("Status code is 200", function () {  
  
    pm.response.to.have.status(200);  
  
});
```

```
pm.test("Body matches string", function () {  
  
    pm.expect(pm.response.text()).to.include("Campaign TEST");  
  
});
```

```
pm.test("Campaign status check", function () {  
  
    var jsonData = pm.response.json();  
  
    pm.expect(jsonData.status).to.eql("Draft");  
  
});
```

Здесь мы проверяем status code, содержание в теле ответа названия кампании и тот факт, что после обновления статус кампании "Draft".

На вкладке Test Results мы увидим, что все три теста выполнены успешно:

Body Cookies Headers (5) Test Results (3/3)

All

Passed

Skipped

Failed

PASS

Status code is 200

PASS

Body matches string

PASS

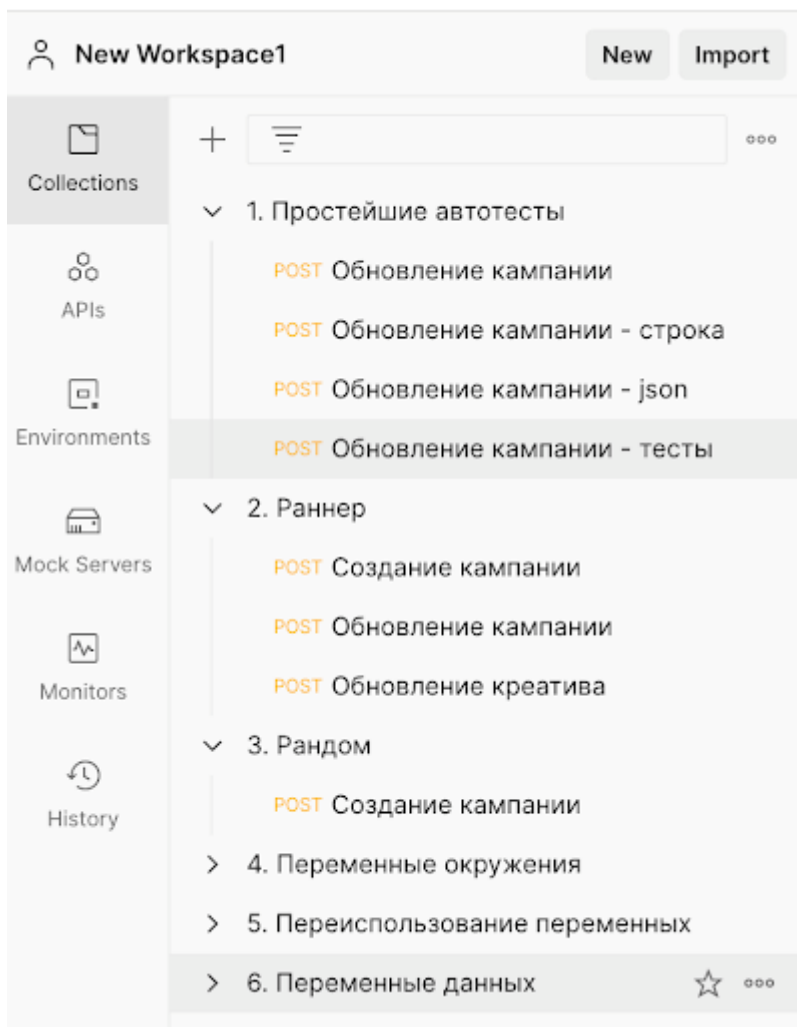
Campaign status check

Запуск автотестов с Collection runner

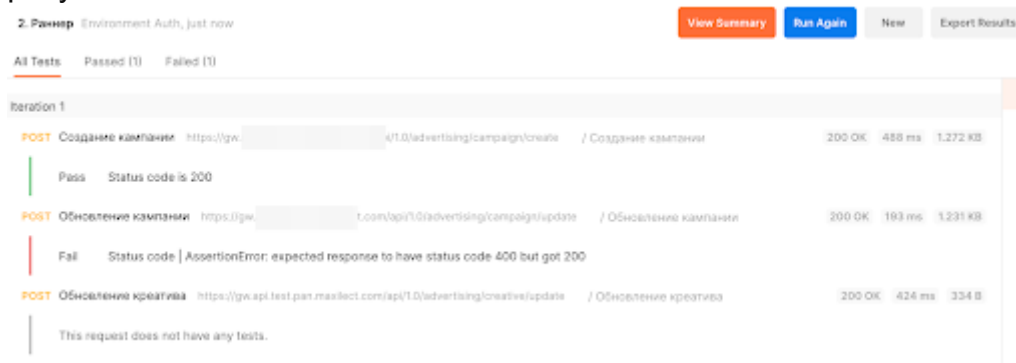
Collection runner запускает не отдельные тесты, а их коллекции.

Новую коллекцию можно создать с помощью значка + на закладке Collections (в каждой такой коллекции можно создать папку с тестами с помощью Add Folder).

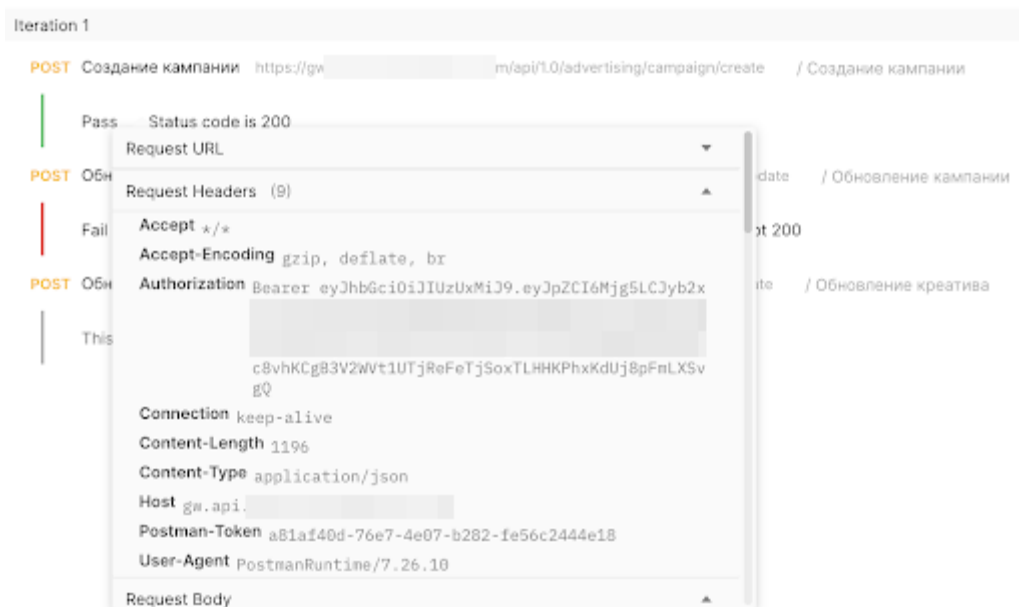
Если создать несколько коллекций, выглядеть это будет примерно так:



Чтобы запустить Collection runner, нужно выбрать коллекцию и на открывшейся вкладке нажать Run. Запросы будут выполняться поочередно. После окончания выполнения можно увидеть все результаты:



По каждому запросу в выпадающем меню можно посмотреть подробные данные - на какой URL запрос был отправлен, какие данные в header и т.п.



Аналогично можно посмотреть тело и заголовок ответа. Но для этого необходимо включить логирование (включить галочку Save response перед запуском коллекции).

При запуске коллекции Collection runner позволяет задать количество итераций. Это очень удобно, особенно когда запросов много - тесты не придется запускать вручную.

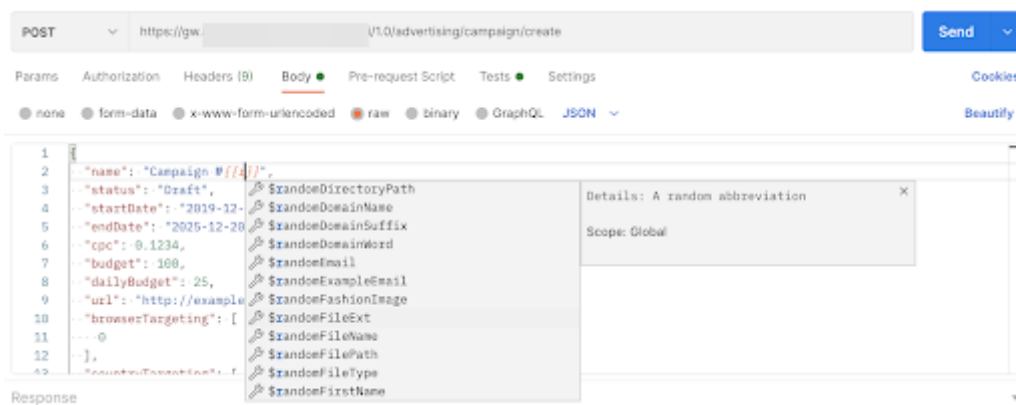
Также Postman умеет запускать коллекции по расписанию. Нажимаем на название коллекции, открываем меню Action и задаем расписание на вкладке Monitor collection.

Переменные в Postman

В ходе тестирования удобно использовать переменные.

Допустим, недавно почистили базу и для тестирования нам нужно ее заполнить - создать несколько рекламных кампаний с разными именами. Чтобы не делать это вручную, можно использовать динамические рандомные переменные.

Рандомных переменных в Postman много. Если при написании кода начать вводить парные фигурные скобки, Postman сам подскажет, какие из них доступны.



Подробнее про переменные можно почитать в [документации к Postman](#).

Для нашего теста выберем \$randomInt. Сохраним и отправим запрос.

Он выполняется успешно, кампания получает случайное название.

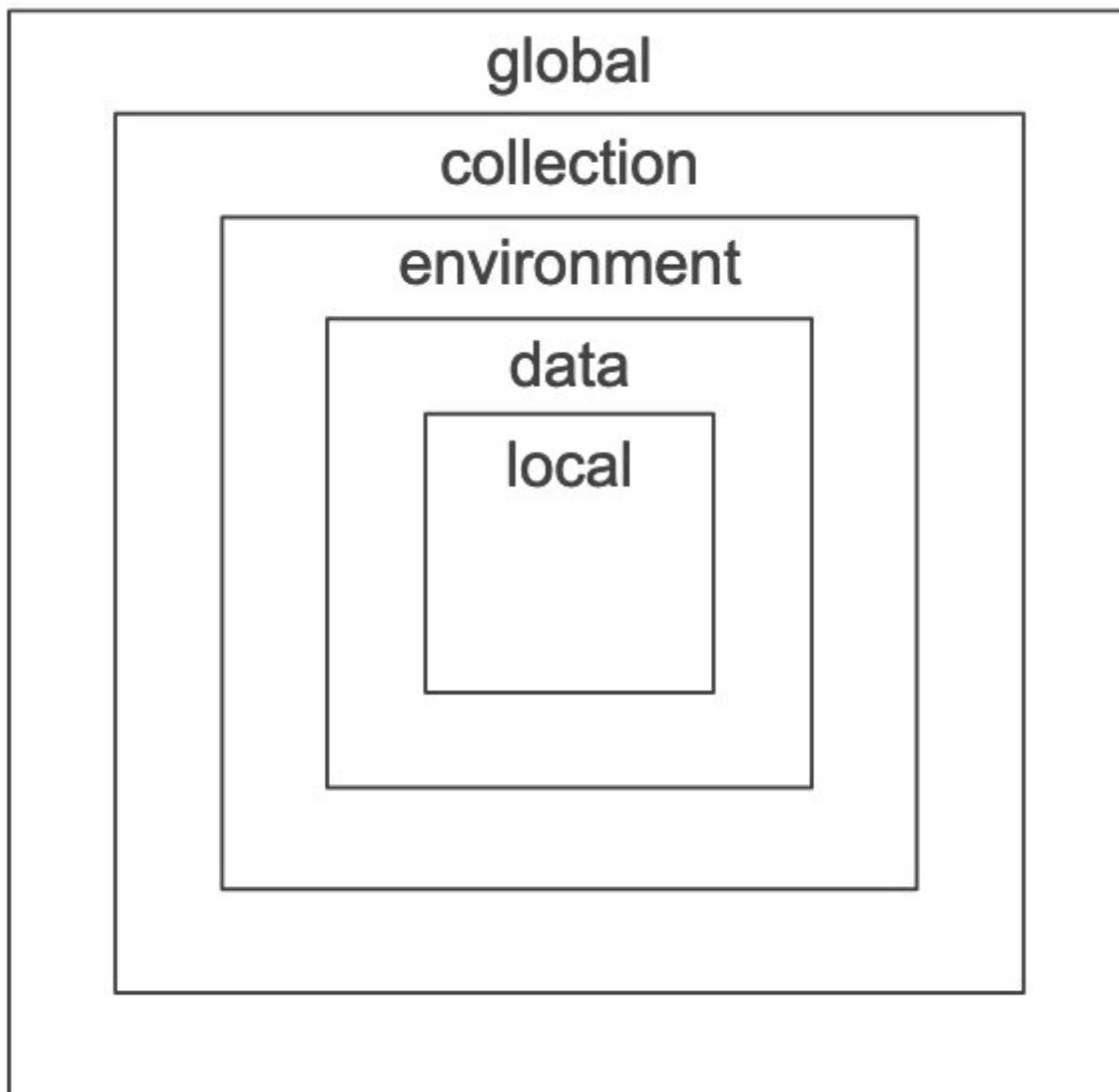
Вместо \$randomInt можно выбрать, допустим, случайный месяц - \$randomMonth.



Чтобы получать уникальные данные, точно так же можно вставлять переменные в header и URL.

Области видимости

Postman поддерживает несколько видов переменных, в зависимости от пространств и областей видимости. Идею хорошо иллюстрирует картинка из документации:



Области

видимости переменных в Postman

Поддерживаются следующие типы переменных:

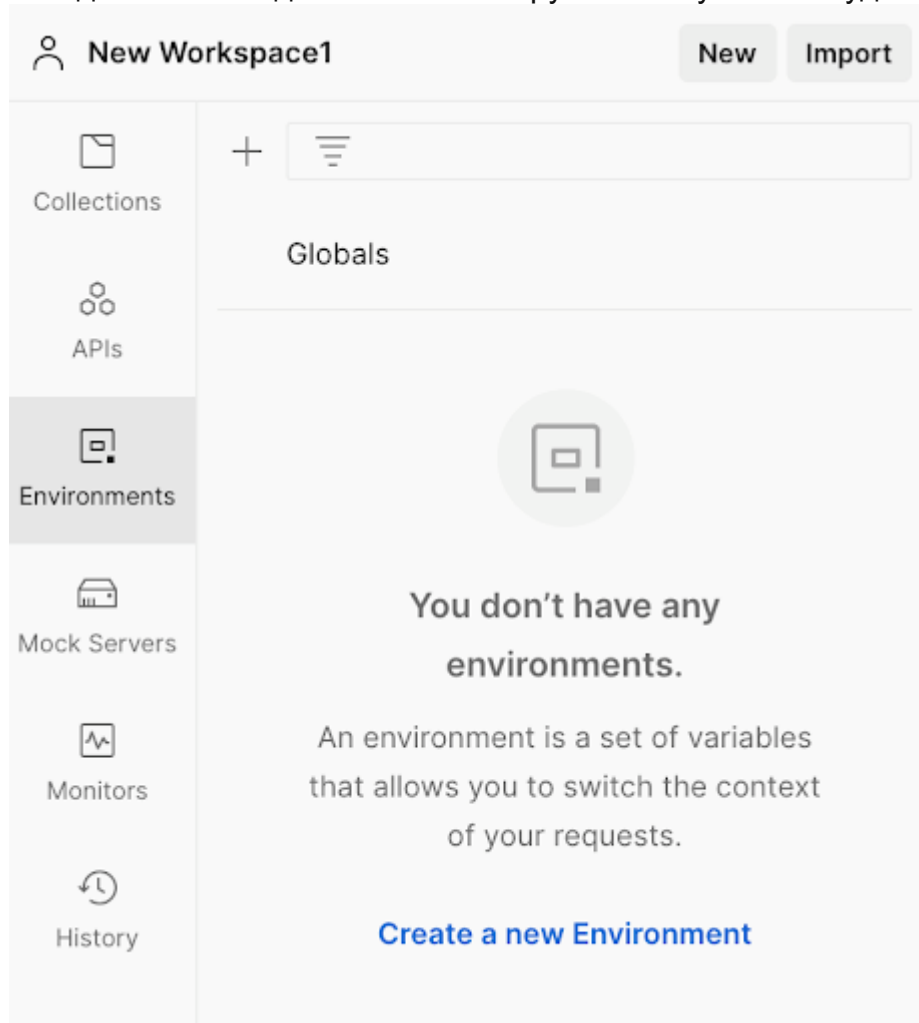
- Глобальные - переменные, которые не относятся ни к какому окружению, они доступны во всем рабочем пространстве, из всех окружений. Глобальные переменные могут использоваться для передачи данных между коллекциями, запросами и окружениями.
- Переменные коллекции доступны во всех запросах внутри одной коллекции.
- Переменные окружения изменяются в зависимости от выбранного окружения.
- Локальные переменные являются временными. Они доступны только внутри запроса и используются, когда нужно переписать переменные коллекции или какие-то глобальные значения.
- Переменные данных - файловые переменные. Я остановлюсь на них подробно далее.

Глобальные переменные не могут иметь дубликаты. А вот локальные переменные могут иметь одни и те же имена, но только если они находятся в разных окружениях. Если Postman встречается с двумя переменными с одинаковыми именами, высший приоритет будет у локальной переменной (она затрет глобальную).

Переменные окружения

Поговорим о переменных окружения. Они позволяют передавать данные из запроса в запрос внутри этого окружения.

Перед началом экспериментов создадим окружение. Для этого в левом меню надо выбрать Environments и нажать либо на плюсики, либо на Create a new environment. На открывшейся вкладке можно задать название окружения. Пусть это будет Environment token.



Отличный пример использования переменных окружения - передача токена авторизации. Если токен истекает, то тест падает с ошибкой 401. Во всех запросах, которые я отправляла до этого, свежие токены авторизации я получала и вставляла вручную ("за кадром" моего рассказа). Когда запросов всего несколько, это не сложно. Но когда запросов много, проще реализовать это через переменную окружения. Для этого мы зададим переменную и присвоим ей соответствующее значение, а потом используем переменную во всех запросах.

Зададим в окружении Environment token переменную varBearerToken. В качестве начального значения подставим наш токен. Остается в header запросов заменить значение токена на переменную.

Как и с случайными переменными, можно начать вводить двойные фигурные скобки. Поскольку Postman знает эту переменную (в данном окружении), он подскажет значение.

ParamsAuthHeaders (9)Body ●Pre-req. TestsSettings

☒

Accept ⓘ

☒

Accept-Encoding ⓘ

☒

Connection ⓘ

☒

Authorization

Bearer {{VarBearerToken}} ⓘ

KeyValue

E VarBearerToken

INITIALeyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcyI6IiJPTEVfQ01fVVNFUiIsImZlZXh0ZXJuYWxBcGkiOiJmYWxzZSIsImhhdCI6MTYzNjU0...

CURRENTeyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcyI6IiJPTEVfQ01fVVNFUiIsImZlZXh0ZXJuYWxBcGkiOiJmYWxzZSIsImhhdCI6MTYzNjk1...

SCOPEEnvironment

Response



Если мы уберем Environment, Postman подсказок уже не предложит, поскольку вне своего окружения переменная недоступна. Нет окружения - нет и переменных окружения.

Точно также можно выполнять наши запросы на разных стендах. Чтобы в каждом запросе вручную не изменять URL, можно прописать стенд-попеременную. Пусть это будет varStage со значением по умолчанию test.

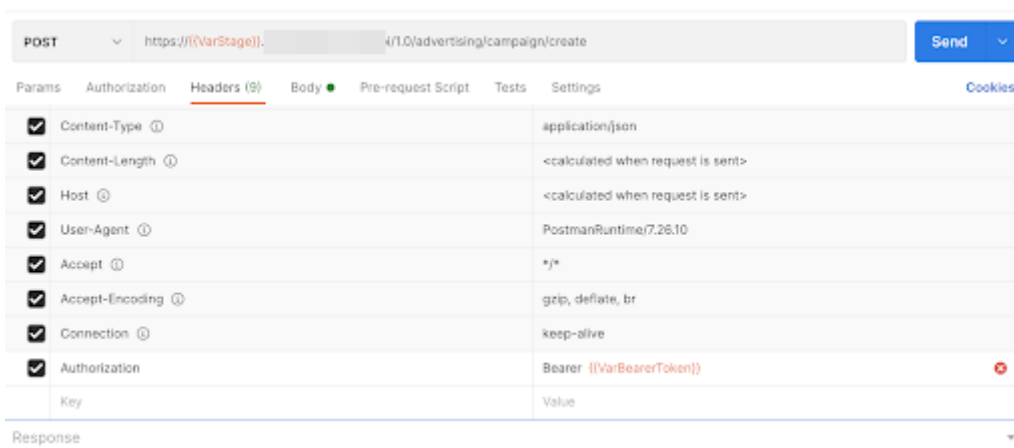
Environment token

SaveShare

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	VarBearerToken	eyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcyI6IiJPTEVfQ01fVVNFUiIsImZlZXh0ZXJuYWxBcGkiOiJmYWxzZSIsImhhdCI6MTYzNjU0...	eyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcyI6IiJPTEVfQ01fVVNFUiIsImZlZXh0ZXJuYWxBcGkiOiJmYWxzZSIsImhhdCI6MTYzNjk1...			
<input checked="" type="checkbox"/>	VarStage	gw.api.test	gw.api.test			
	Add a new variable					

Аналогично можно было бы прописать dev и другие стенды.

Остается добавить эту переменную в URL. Как и в случае с токеном, Postman будет ее подсвечивать, только если выбрано то окружение, в котором она задана.



Окружение со всеми переменными можно пошарить коллегам через экспорт в файл json. Они смогут открыть его на своем рабочем месте в Postman.

Передача данных между запросами

Очень важная возможность для написания тестов - передача значений из запроса в запрос с помощью переменных окружения.

В предыдущем примере я создала переменную окружения varToken и в неё вручную приписала токен. Но вместо копирования токен можно получить в ответе на запрос авторизации и передать его в другие запросы.

Создадим новое окружение Environment Auth. Не будем на старте прописывать никаких переменных. Просто выберем это окружение и возьмем из списка сниппетов соответствующий скрипт. Прямо в нем объявим переменную varToken и присвоим ей значение токена (для этого парсим JSON ответа):

```
var jsonData = JSON.parse(responseBody);

pm.environment.set("varToken", jsonData.data.token);

console.log(jsonData);

console.log(jsonData.data.token);
```

Для наглядности я добавила еще вывод переменных в консоль, чтобы видеть их значения.

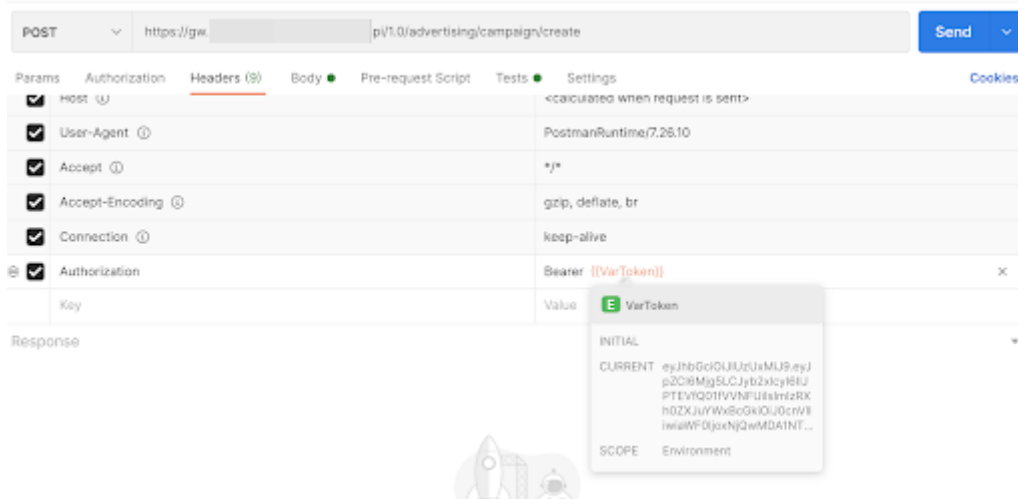
Чистим консоль, запускаем. И видим результат - переменную jsonData и значение токена, которое присваиваем varToken.

POST https://gw. /api/1.0/auth/login

```
{data: {...}, message: "Ok", status: "200"}
data: {...}
  token: "eyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcYI6IjJPTeVfQ01fVWVfUjIzZXZi73Sk0CyH2TFaRmYy0teuZspw"
  refreshToken: "0b04d4f7-450e-45b3-af40-f1e04af0746a"
  expired: "2021-12-20T14:05:47.388427"
  durationSeconds: 3600
  scheme: "Bearer"
  message: "Ok"
  status: "200"

"eyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcYI6IjJPTeVfQ01fVWVfUjIzZXZi73Sk0CyH2TFaRmYy0teuZspw"
```

Остается прописать переменную varToken в Header наших запросов на создание кампании.



Поскольку работаем мы внутри окружения Environment Auth, Postman о ней знает.

А теперь усложним пример. Допустим, мы хотим создать рекламную кампанию, выяснить ее ID, а затем обновить кампанию с этим ID.

Объявим еще одну переменную - varID. Для этого название переменной задаем в скрипте, там же присваиваем ей значение из ответа на запрос:

```
var jsonData = JSON.parse(responseBody);
```

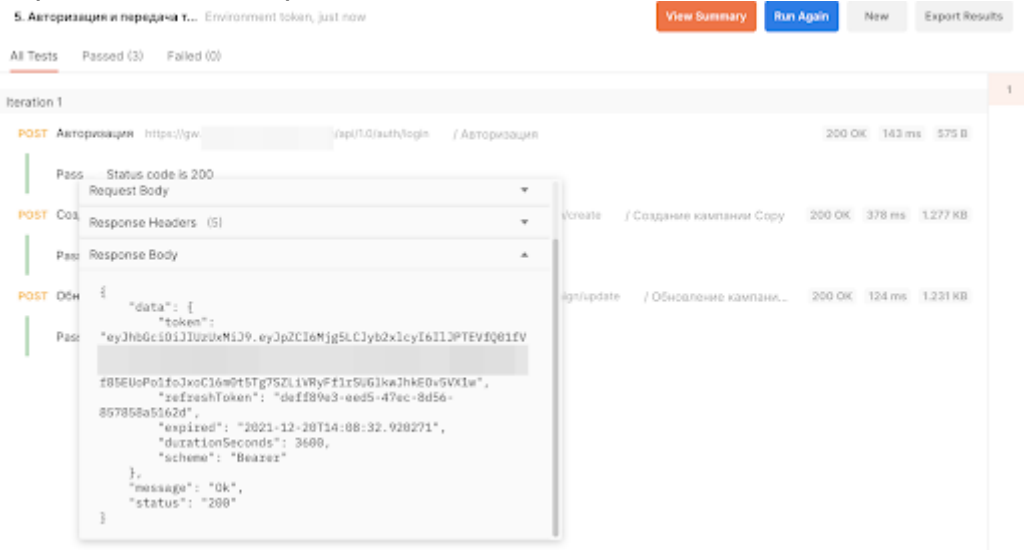
```
pm.environment.set("varID", jsonData.id);
```

```
console.log(jsonData);
```

```
console.log(jsonData.data.id);
```

Запускаем Collection runner и видим, что запросы работают.

Мы получили токен, передали его во все последующие запросы, получили ID кампании и передали его в запрос на обновление. После этого обновили.



Кстати, в окружении, которое мы создавали, все эти переменные будут указаны, вместе с текущими значениями:

Environment Auth						
			Save	Share		
	VARIABLE	INITIAL VALUE	CURRENT VALUE		Persist All	Reset All
<input checked="" type="checkbox"/>	varToken		eyJhbGciOiJIUzUxMiJ9.eyJpZCI6Mjg5LCJyb2xlcyl6IjE1PTEVfQ01FV...			
<input checked="" type="checkbox"/>	varID		5438			
	Add a new variable					

Используя возможности Postman, обновление кампании можно выполнять по расписанию.

Переменные данных

В Collection runner можно использовать переменные данных.

Чтобы продемонстрировать, как это работает, создам несколько рекламных кампаний с параметрами, заданными через файл. Предположим, нам не подходят рандомные значения и нужны строго определенные имена кампаний для проверки сортировки или фильтрации.

Создаем файл с расширением csv или json. Postman поддерживает оба типа файлов, вопрос лишь в формате.

В файле csv в первой строке указывается название переменной или нескольких переменных через запятую. Далее на отдельных строках следуют значения (или несколько значений через запятую).



varCompanyName.csv

```
varCompanyName  
Новая кампания  
New campaign  
Νέα καμπάνια  
Nouvelle campagne  
新しいキャンペーン
```




```
varCompanyName, varCreativeTitle, varCreativeDescription
Новая кампания, Новый креатив, Новое описание креатива
Новая кампания 123, Новый креатив 123, Новое описание креатива 123
New campaign, New creative, New creative description
Νέα καμπάνια, Νέο δημιουργικό, Νέα περιγραφή δημιουργικού
Nouvelle campagne, Nouvelle création, Nouvelle description de création
新しいキャンペーン, 新しいクリエイティブ, 新しいクリエイティブの説明
```

В файле json можно прописать то же самое, но в JSON-формате “ключ-значение”.



```
[{
  "varCompanyName": "Новая кампания",
  "varCreativeTitle": "Новый креатив",
  "varCreativeDescription": "Новое описание креатива"
},
{
  "varCompanyName": "New campaign",
  "varCreativeTitle": "New creative",
  "varCreativeDescription": "New creative description"
},
{
  "varCompanyName": "Νέα καμπάνια",
  "varCreativeTitle": "Νέο δημιουργικό",
  "varCreativeDescription": "Νέα περιγραφή δημιουργικού"
},
{
  "varCompanyName": "Nouvelle campagne",
  "varCreativeTitle": "Nouvelle création",
  "varCreativeDescription": "Nouvelle description de création"
},
{
  "varCompanyName": "新しいキャンペーン",
  "varCreativeTitle": "新しいクリエイティブ",
  "varCreativeDescription": "新しいクリエイティブの説明"
}]
```

Чтобы добавить переменные в Collection runner, нужно нажать кнопку Select File и загрузить любой из этих файлов. Collection runner автоматически посчитает количество значений (и соответственно итераций тестов). Там же можно посмотреть названия и значения переменных, нажав на кнопку Preview Data.

PREVIEW DATA

×

Iteration	varCompanyName	varCreativeTitle	varCreativeDescription
1	"Новая кампания"	"Новый креатив"	"Новое описание креатива"
2	"New campaign"	"New creative"	"New creative description"
3	"Νέα καμπάνια"	"Νέο δημιουργικό"	"Νέα περιγραφή δημιουργικού"
4	"Nouvelle campagne"	"Nouvelle création"	"Nouvelle description de création"
5	"新しいキャンペーン"	"新しいクリエイティブ"	"新しいクリエイティブの説明"

Если запустить Collection runner, а потом проверить названия кампаний, мы увидим, что использованы значения из файлов.