GeekBrains

Тестирование Backend на Java

Ручное тестирование RESTful API-сервисов с использованием Postman

На этом уроке

1. Вспомним основы REST API.
2. Запустим реквесты в Postman.
3. Автоматизируем наши проверки.
4. Познакомимся с Newman и запустим тесты из CLI.

Оглавление

Введение

В курсе «Тестирование веб-приложений» мы вручную тестировали RESTful API-сервис с использованием Postman. На этом уроке автоматизируем рутинные проверки.

Для повторения базовой теории обратимся к методичке курса по тестированию веб-приложений. В этом пособии рассмотрим более детально аспекты тестирования REST API и узнаем, какими инструментами располагает инженер по автоматизированному тестированию.

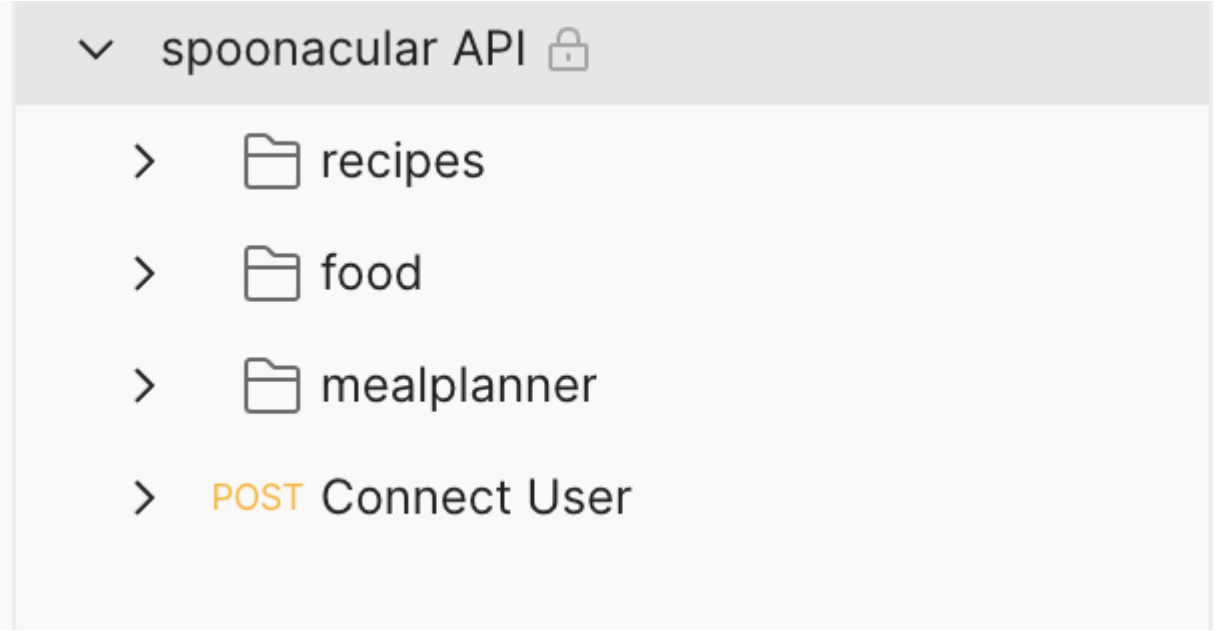
Автоматизация с использованием Postman

На этом уроке мы поработаем с сервисом хранения и поиска рецептов. Этот сервис интересен тем, что для использования большинства эндпоинтов API требуется авторизация, а также он хорошо документирован и даже содержит готовые коллекции запросов.

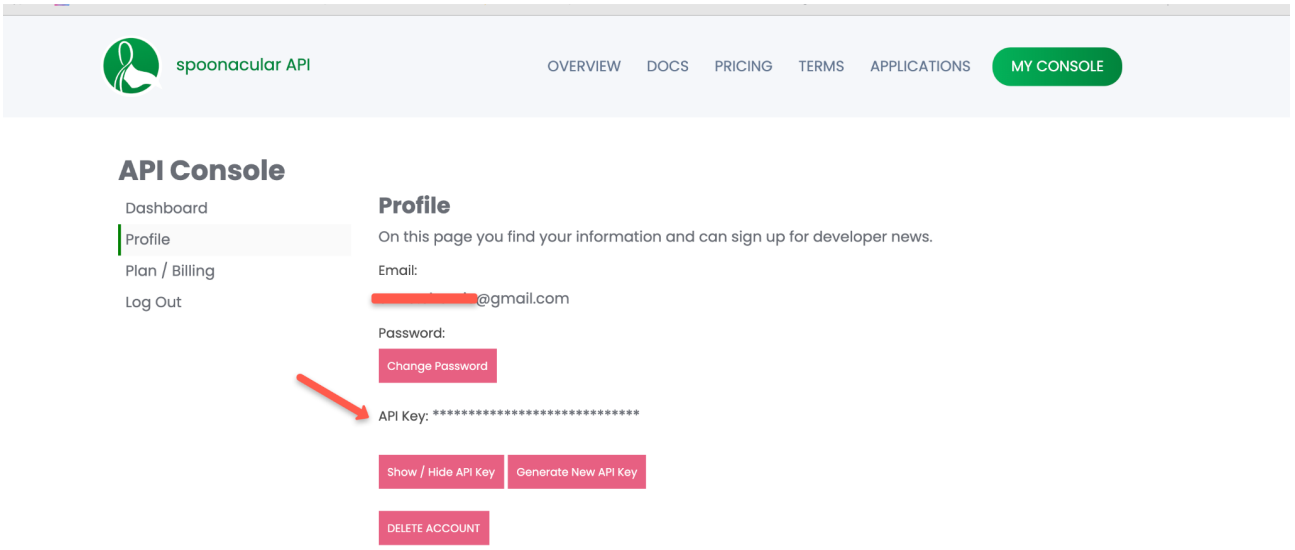
Чтобы получить документированную коллекцию запросов, надо перейти по ссылке выше и нажать Run in Postman.

Авторизация в проекте

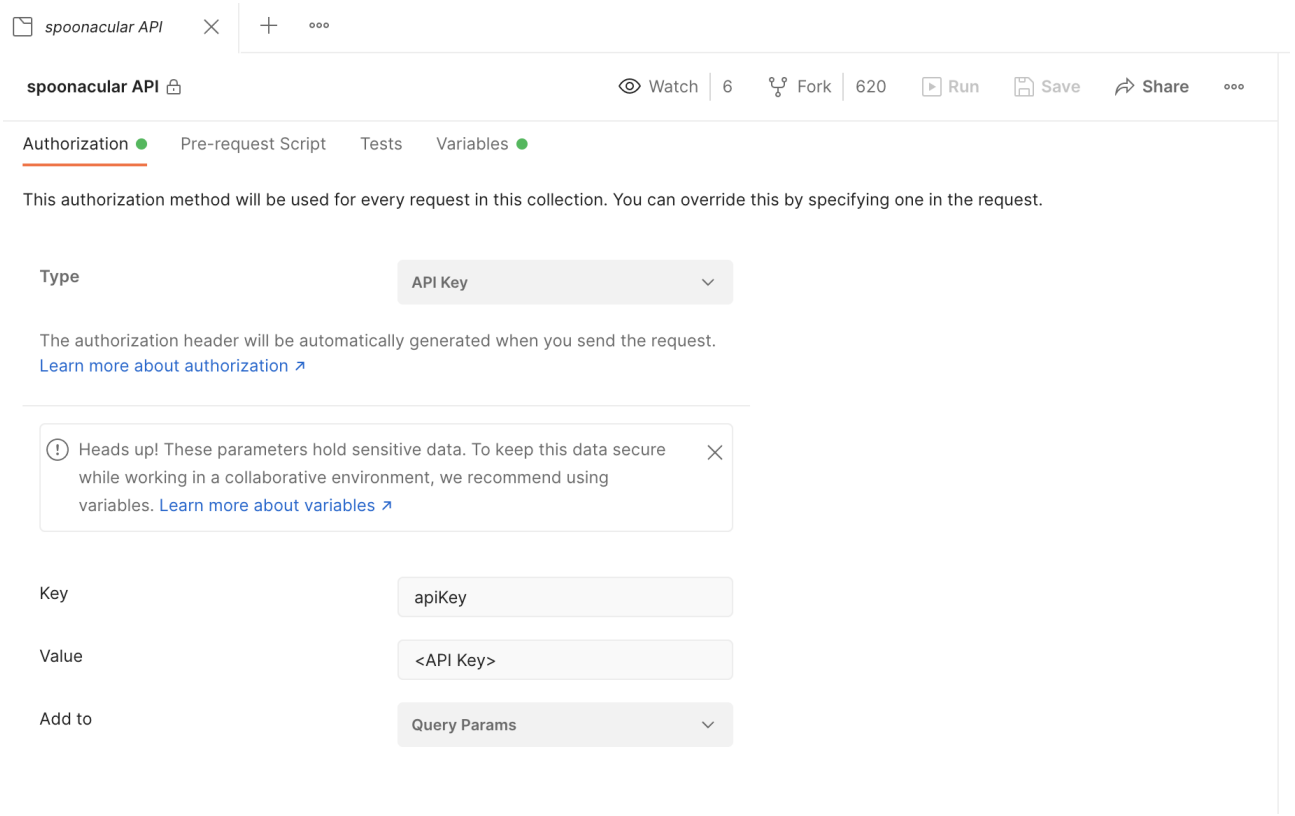
Импортированная коллекция содержит папки по покрытым функциональностям:



Чтобы запустить их, требуется получить токен. Для этого надо зарегистрироваться на сервисе и скопировать токен из раздела API console → Profile:



Теперь нужно перейти в Postman и нажать на вкладку Authorization и выбрать API key:



Вставьте в поле Value ваш ключ и сохраните изменения с помощью комбинации CTRL+ S. Готово!

Переходим к автоматизации.

Запуск запроса и первые проверки

Для нужд автоматизации лучше создать свою коллекцию и скопировать туда запросы. Затем надо настроить авторизацию, выбрав в настройках коллекции уже полученный ранее токен (данные подставляются автоматически).

Начнём с запроса Search Recipes (GET `/recipes/complexSearch`).

Запустим GET-запрос без параметров и увидим результат:


```
{
   "results": [
       {
           "id": 716426,
           "title": "Cauliflower, Brown Rice, and Vegetable Fried Rice",
           "image": "https://spoonacular.com/recipeImages/716426-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 715594,
           "title": "Homemade Garlic and Basil French Fries",
           "image": "https://spoonacular.com/recipeImages/715594-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 715497,
           "title": "Berry Banana Breakfast Smoothie",
           "image": "https://spoonacular.com/recipeImages/715497-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 644387,
           "title": "Garlicky Kale",
           "image": "https://spoonacular.com/recipeImages/644387-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 716268,
           "title": "African Chicken Peanut Stew",
           "image": "https://spoonacular.com/recipeImages/716268-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 716381,
           "title": "Nigerian Snail Stew",
           "image": "https://spoonacular.com/recipeImages/716381-312x231.jpg",
           "imageType": "jpg"
       },
       {
           "id": 782601,
           "title": "Red Kidney Bean Jambalaya",
```

```
            "image": "https://spoonacular.com/recipeImages/782601-312x231.jpg",
            "imageType": "jpg"
        },
        {
            "id": 794349,
            "title": "Broccoli and Chickpea Rice Salad",
            "image": "https://spoonacular.com/recipeImages/794349-312x231.jpg",
            "imageType": "jpg"
        },
        {
            "id": 715446,
            "title": "Slow Cooker Beef Stew",
            "image": "https://spoonacular.com/recipeImages/715446-312x231.jpg",
            "imageType": "jpg"
        },
        {
            "id": 715415,
            "title": "Red Lentil Soup with Chicken and Turnips",
            "image": "https://spoonacular.com/recipeImages/715415-312x231.jpg",
            "imageType": "jpg"
        }
    ],
    "offset": 0,
    "number": 10,
    "totalResults": 5226
}
```


Что можно сделать в этом ответе?

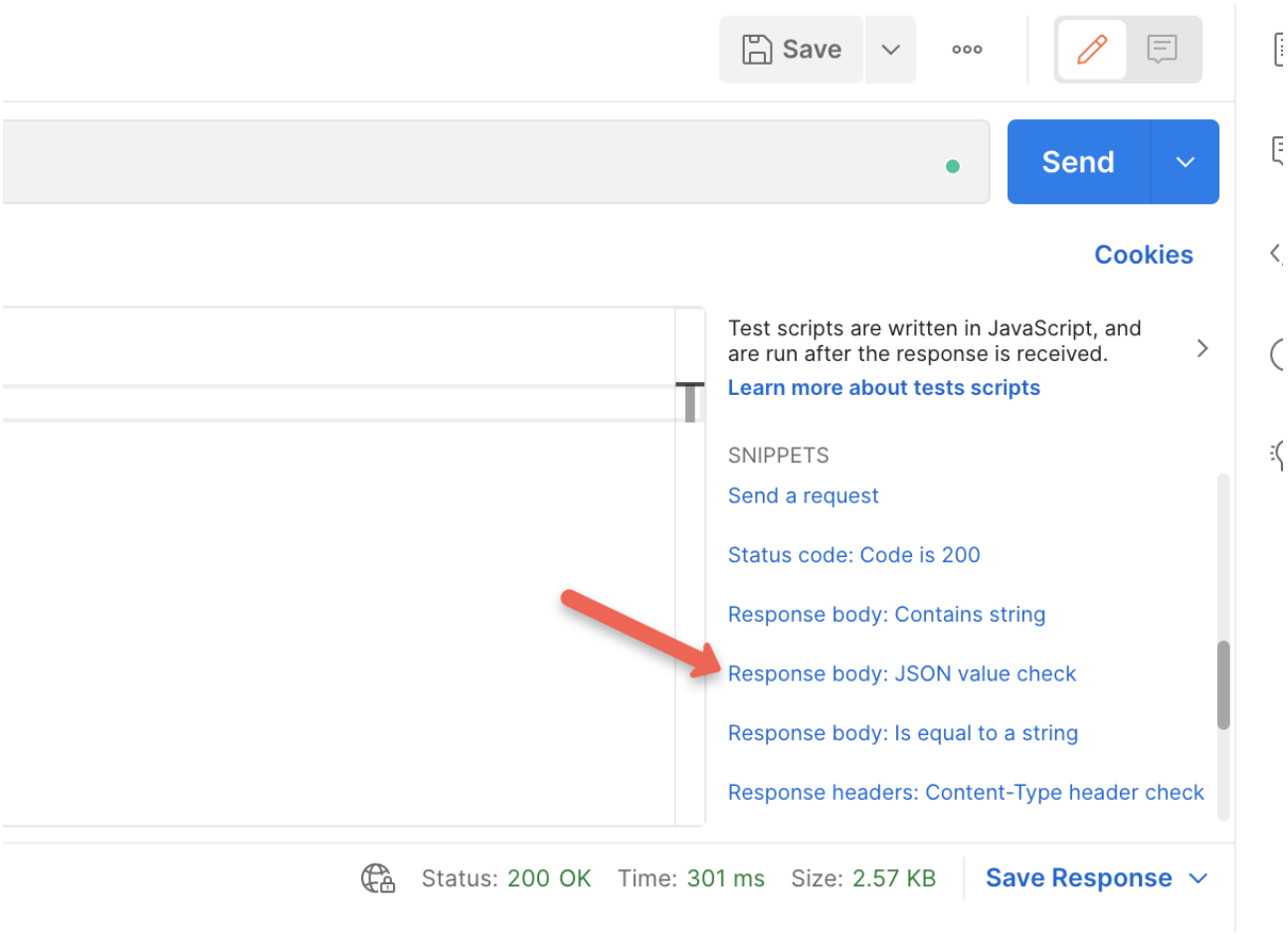
1. Проверить results — он должен быть непустым (так как мы не задавали никаких ограничений для поиска).
2. Поле offset должно быть равно нулю, так как это первая страница результатов поиска
3. Поле number содержит значение по умолчанию

Всё это будем делать на вкладке Tests. Напишем тесты на совпадение значения поля offset и значения 0. Для этого выполним следующие шаги:

1. Распарсить JSON ответа.
2. Взять значение поля offset.
3. Проверить равенство значения поля и числа 10.

Парсим JSON ответа:

Справа на вкладке Tests доступна панель сниппетов - это небольшие заготовленные кусочки кода, которые сильно помогают в тестировании. Возьмем вот этот сниппет:



Он состоит из нескольких строк, для парсинга json нам нужна вот эта строка:

```
var jsonData = pm.response.json();
```

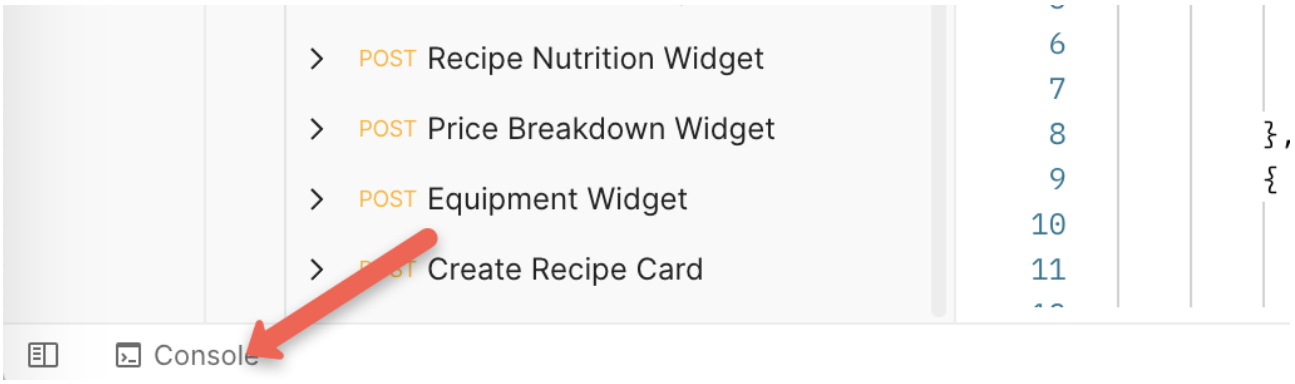
Возьмём значение поля offset следующей строчкой:

```
var offsetData = jsonData.offset;
```

Для проверки выведем это значение на консоль:

```
console.log(offsetData);
```

Консоль, кстати, находится в нижней части экрана:



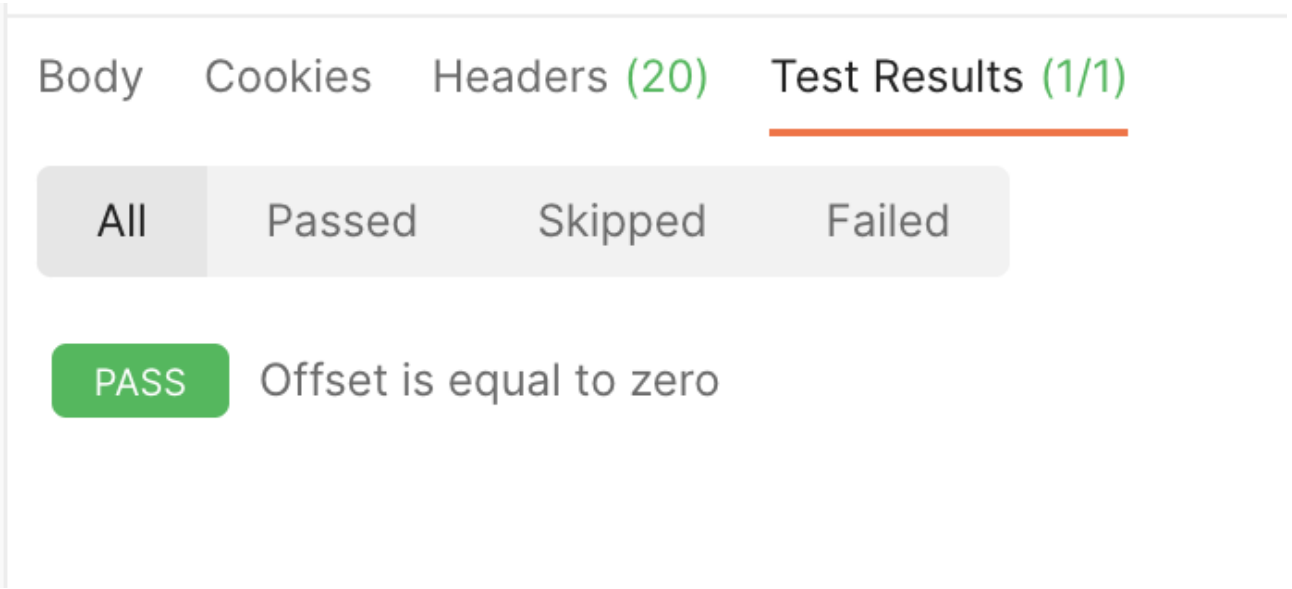
Далее сделаем проверку на равенство:

```
pm.expect(offsetData).to.eql(0);
```

В итоге наш тест будет выглядеть так:

```
pm.test("Offset is equal to zero", function () {
    var jsonData = pm.response.json();
    var offsetData = jsonData.offset;
    pm.expect(offsetData).to.eql(0);
});
```

Запустим запрос снова и увидим, что прошел наш новый тест:



Переменные в Postman

Разберёмся, какие есть переменные, и чем они полезны для нас.

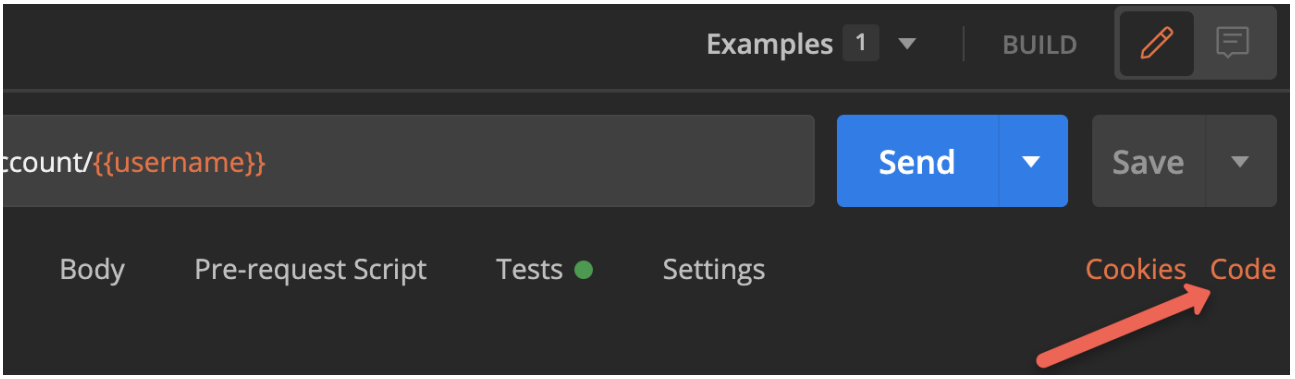
Пользовательские переменные

Переменные отличаются по уровню:

1. Глобальные.
2. Переменные окружения.
3. Переменные коллекции.

Рекомендуется всегда использовать переменные окружения и создавать окружения под каждый новый стенд или продукт. Глобальные переменные распространяются на все коллекции и окружения. Переменные коллекции работают только внутри конкретной коллекции.

Обращаться к любой настроенной переменной можно через {{название_переменной}}. Чтобы увидеть, какая переменная вставится в текущий запрос, воспользуемся кнопкой Code в правом верхнем углу панели табов:



Выберите вид отображения, например, cURL или HTTP:

```
GET /3/account/<значение переменной> HTTP/1.1
Host: api.imgur.com
Authorization: Bearer 9d2306f677fa45ecbbe39df15c86f710fb9692fc
Cookie: __auc=ea4f2c111769328930b677ab678; retina=1;
amplitude_id_f1fc2abcb6d136bd4ef338e7fc0b9d05imgur.com=eyJkZXZpY2VJZCI6IjhjNWNiN
2U2LWIzYzktNGRhZi04OTU4LTZiMzkzYWQ4MzI0NlIiLCJ1c2VySWQiOm51bGwsIm9wdE91dCI6ZmFsc
2UsInNlc3Npb25JZCI6MTYwODc4NjY3ODgwNSwibGFzdEV2ZW50VGltZSI6MTYwODc4NjY3ODgwOCwiZ
XZlbnRJZCI6MCwiaWRlbnRpZnlJZCI6Miwic2VxdWVuY2VOdW1iZXIiOjJ9;
IMGURUIDJAFO=7f5480597754fa931a7a1e290222f42505eb157089ed292d1003dcb080360124;
SESSIONDATA=%7B%22sessionCount%22%3A1%2C%22sessionTime%22%3A1608786678838%7D;
_fbp=fb.1.1608786678869.1120761039; __qca=P0-1506535490-1608786678992;
IMGURSESSION=d4ae6fa98c7199d9cfe2ab4bfad8d45b; _nc=1;
UPSERVERID=upload.i-0700267f09c87e916.production
```

Переменными можно управлять в тестах, используя одну строчку кода по аналогии с SoapUI. Для управления каждым видом переменных создаются специальные сниппеты:

Как и в SoapUI, переменные применяются, чтобы передавать информацию между запросами. В Postman нет механизма Property Transfer, поэтому лучший путь к сохранению и передаче данных — это переменные.

Динамические (встроенные) переменные

Postman предоставляет широкие возможности для использования генераторов конкретных значений. Все встроенные переменные имеют знак $ перед названием. Так, переменная {{$timestamp}} даёт нам текущий таймстамп, а {{$guid}} — случайный UUID.

Полный список встроенных переменных — здесь. Их задача — разнообразить наши тестовые данные и избежать эффекта пестицида, а также сэкономить время на написание подготовительных скриптов для данных.

Tests. Сниппеты

Язык программирования для Postman — это Javascript. Вкладка Tests содержит большое число сниппетов (вспомогательных шаблонов), которые генерируют готовый код, куда подставляются требуемые значения. Например, сниппет Status code: Code is 200 представляет собой следующий фрагмент кода:

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

Главное назначение вкладки Tests — хранить ассерты для тестов. Чтобы выполнить какие-либо действия, например, создать или проверить переменные до выполнения шага, лучше использовать вкладку Pre-request script.

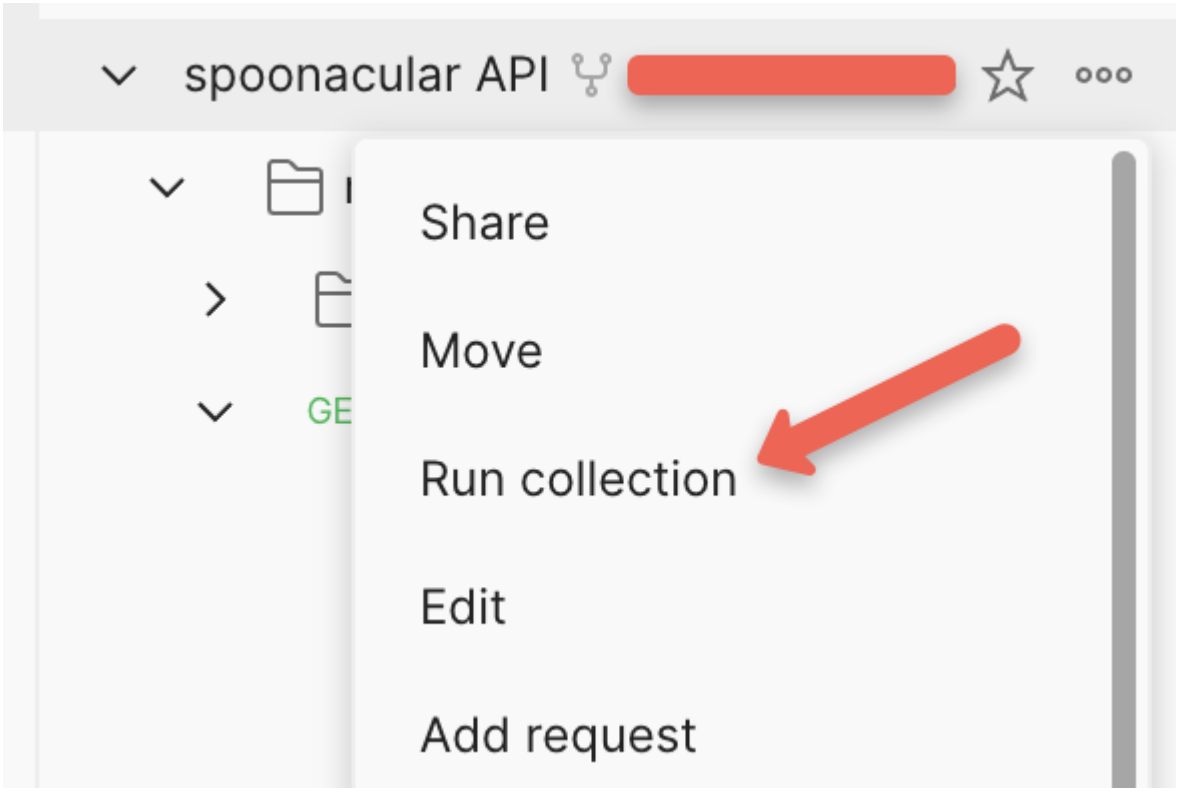
Pre-request scripts

Здесь мы видим те же сниппеты для работы с переменными и сниппет для отправки запроса. В практике часто используются рандомизаторы целых и дробных чисел — и лучшего места для создания переменной с таким значением найти трудно.

```
pm.globals.set('randomNumber', Math.floor(Math.random() *1000));
```

Collection run

После того как мы реализовали связные цепочки тестов, их можно запустить:

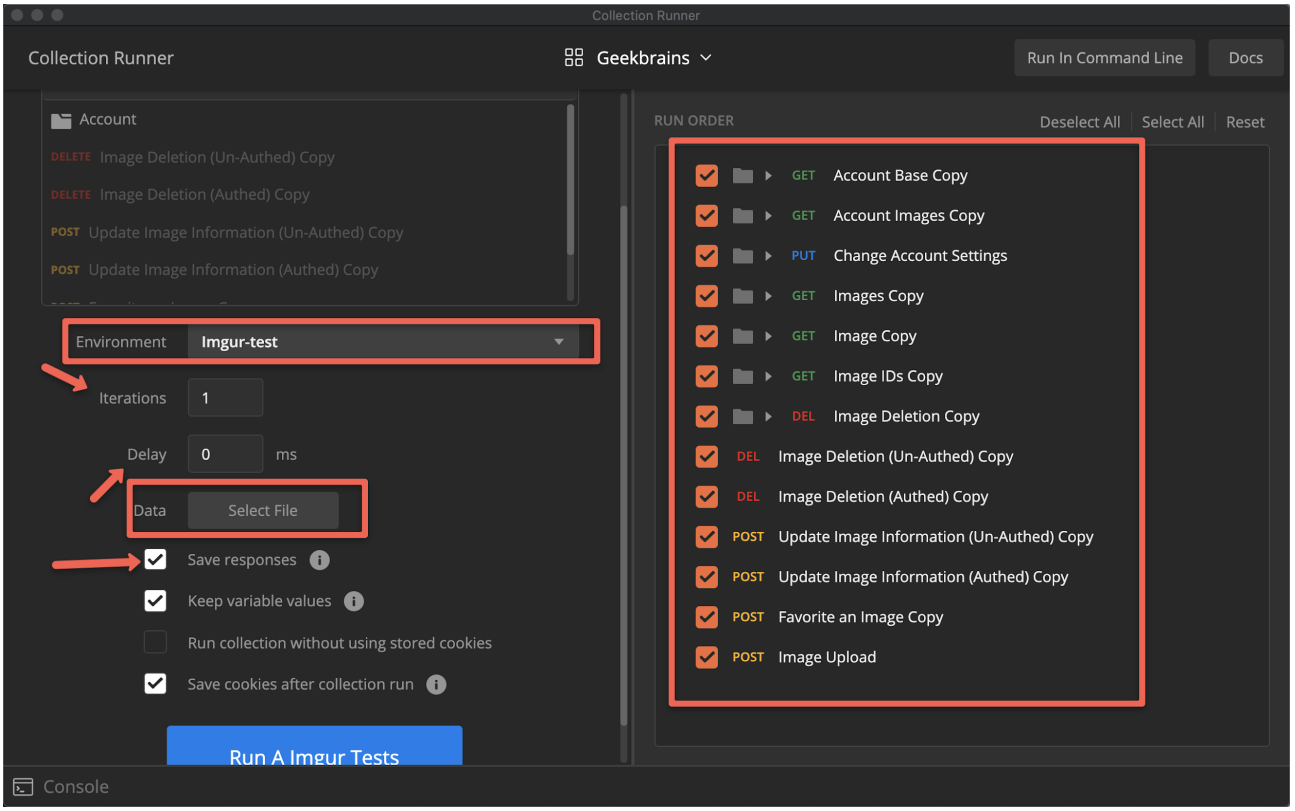


В открывшемся окне мы видим много опций. Это главные из них:

1. Выбрать окружение, где будут прогоняться тесты.
2. Указать количество итераций (повторов прогона).
3. Указать конкретные тесты, которые (не) будут прогоняться, а затем поменять их порядок.
4. Указать задержку перед каждым тестом, например, если мы знаем, что потребуется некоторое время для обновления данных на бэкенде.
5. Указать источник данных не только из environment, но и из файла (data). Таким образом, можно создать настоящие параметризованные тесты, которые будут прогоняться столько раз,

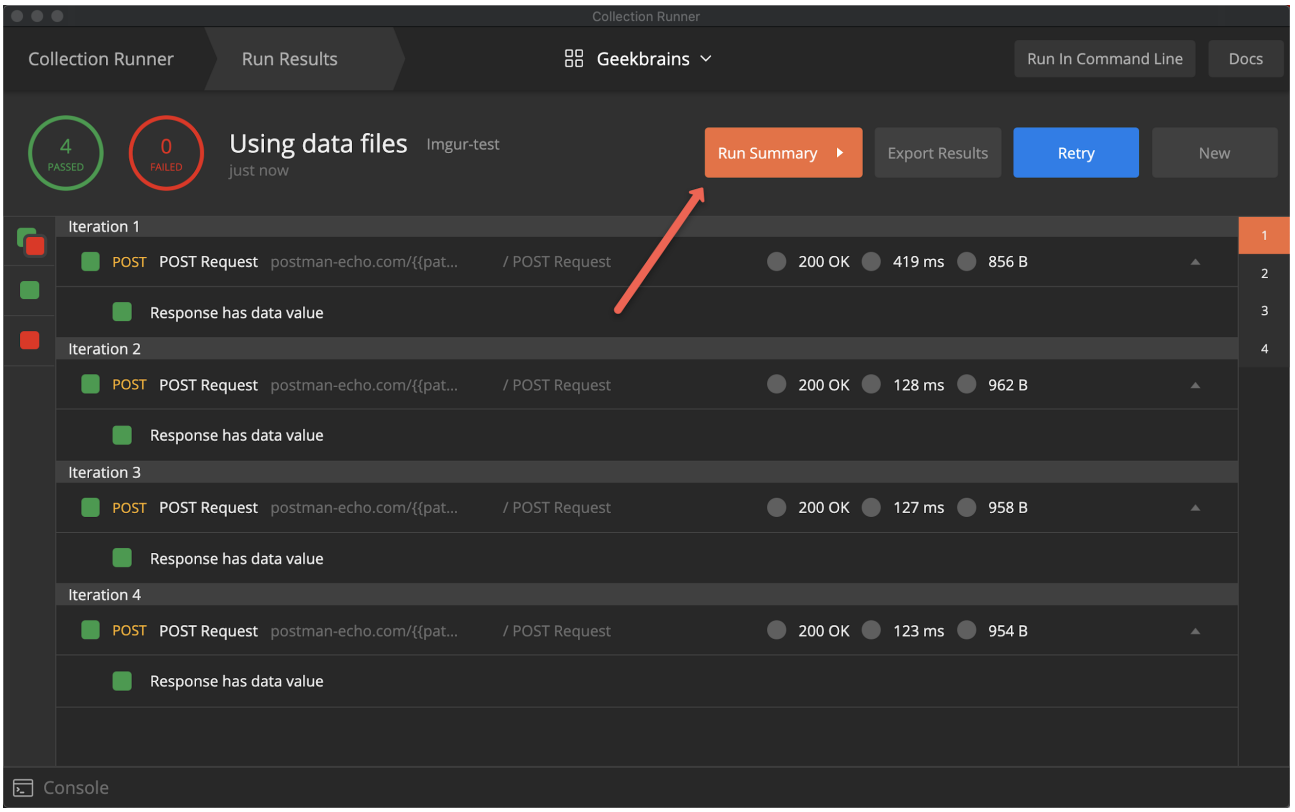
сколько наборов данных мы указали. Форматы файлов — csv или json. Количество итераций изменится автоматически в зависимости от количества тестовых наборов. Посмотреть, как это работает, можно здесь.

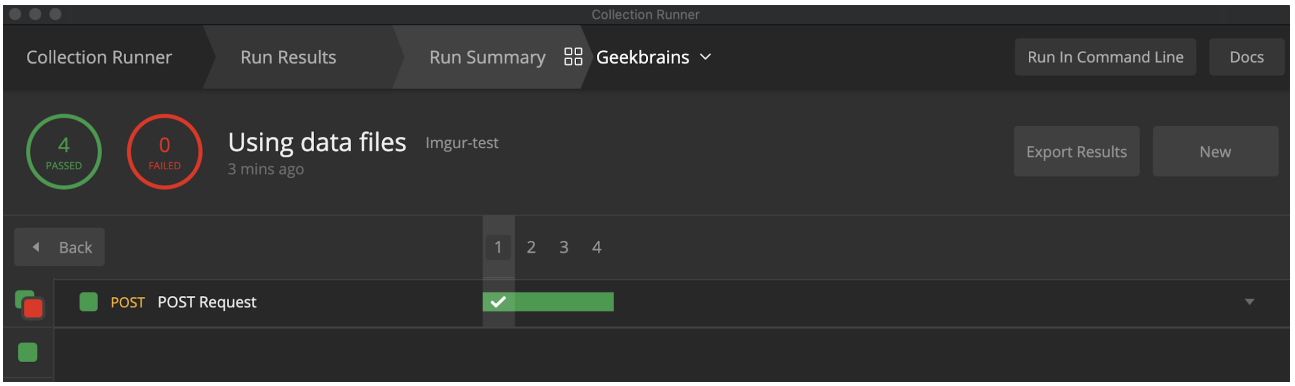
6. Указать (обязательно) чекбокс, чтобы в отчёте сохранялись данные респонсов.



После нажатия на кнопку Run коллекция запустится и остановится при первом падении теста. Если итераций больше 1, то сразу начнётся следующая итерация.

Чтобы увидеть краткий отчёт по всем итерациям, надо нажать на кнопку Run Summary.





Данные отчёта также экспортируются в формате JSON.

Newman

Collection Runner запускается из консоли. Особенно это важно для CI и запуска в облаке. Чтобы воспользоваться этой возможностью, надо установить Newman, экспортировать коллекцию и выполнить команду в консоли:

```
newman run <json-файл коллекции>
```

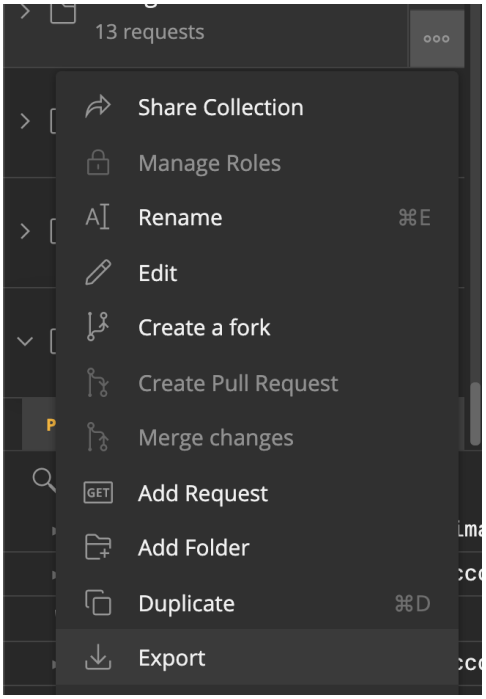
Подробная документация — здесь.

Практическое задание

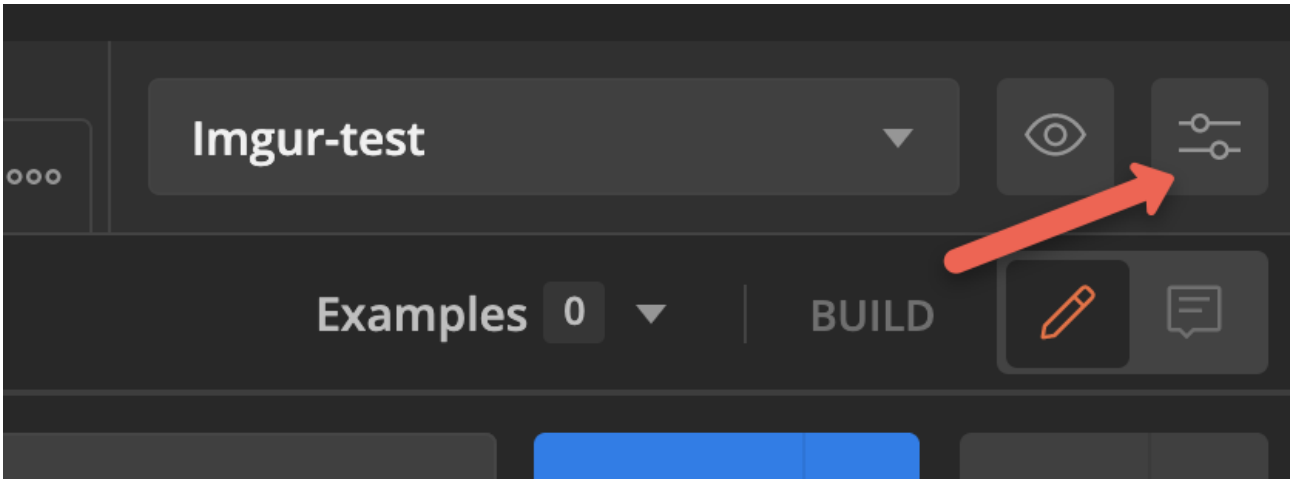
Работа с Spoonacular API.

Формат сдачи: экспортированная коллекция и файл окружения (если нужен) в zip-архиве.

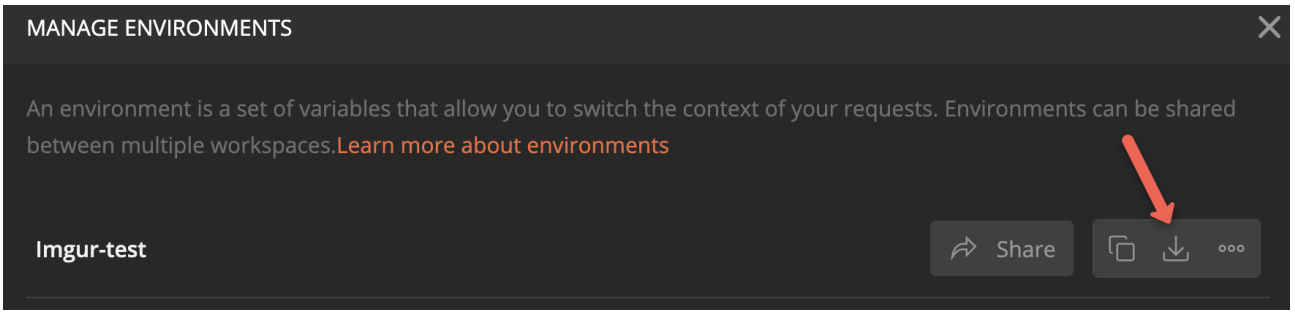
Коллекция экспортируется по пункту контекстного меню:



Файл окружения экспортируется через вызов настройки окружений:



И через нажатие кнопки Export около выбранного окружения:



1. **Автоматизируйте тестирование для GET /recepies/complexSearch (минимум 20 кейсов) и POST /recipes/cuisine (минимум 10 кейсов)**
2. Реализуйте и приложите файл с параметрами для запуска тестов в коллекции.
3. *Реализуйте хотя бы 1–2 цепочки тестов для своего любимого продукта с открытым API, например:
    - Twitter API;
    - «ВКонтакте».

Дополнительные материалы

1. Презентация на тему «Тест-дизайн и автоматизация REST API».

Используемые источники

1. Документация Postman.