Так как создание коллекций и запросов в конструкторе не должно вызвать затруднений, практическую часть посвятим написанию скриптов, и рассмотрим как создать цепочку запросов с тестами. Перед началом создайте новую коллекцию, назовите её “Habra”, затем создайте новое окружение под названием "Habra.Env"

**Шаг 1**

Создайте новый запрос, в качестве URL укажите <https://postman-echo.com/get?userId=777>, а метод оставьте GET. Здесь и далее для простоты и удобства мы будем использовать echo-сервер любезно предоставленный разработчиками Postman. Сохраните запрос в коллекцию "Habra" и назовите “Get user id”, он будет имитировать получение идентификатора пользователя от сервера. Перейдите на вкладку "Tests" и напишите следующий код:

*// Тестируем статус ответа и формат*

pm.test("Status is ok, response is json", function () {

pm.response.to.be.ok; *// проверка статуса*

pm.response.to.be.json; *// проверка формата*

});

try {

*// сохраняем userId из json ответа в переменную окружения*

pm.environment.set("userId", pm.response.json().args.userId);

} catch(e) {

*// выводим исключение в консоль*

console.log(e);

}

С помощью этого скрипта мы проверили статус и формат ответа, а также сохранили полученный из ответа идентификатор пользователя в переменную окружения для последующего использования в других запросах. Чтобы проверить работу нашего теста, запустите запрос. В панели информации об ответе, во вкладке "Tests" вы должны увидеть зелёный лейбл “PASS”, а рядом надпись “Status is ok, response is json”.

**Шаг 2**

Теперь давайте попробуем вставить идентификатор нашего пользователя в json запрос. Создайте новый запрос, в качестве URL укажите <https://postman-echo.com/post>, выберите метод POST, установите тип для тела запроса raw — application/json, а в само тело вставьте:

{"userId": {{userId}}, "username": "Bob"}

Сохраните запрос в нашу коллекцию, и назовите "Update username", он будет запрашивать импровизированную конечную точку для обновления username пользователя. Теперь при вызове запроса вместо {{userId}} система автоматически будет подставлять значение из переменной окружения. Давайте проверим, что это действительно так, и напишем небольшой тест для нашего нового запроса:

*// тестируем статус ответа и формат*

pm.test("Status is ok, response is json", function () {

pm.response.to.be.ok;

pm.response.to.be.json;

});

*// проверим, что userId из окружения равен userId из ответа*

pm.test("userId from env is equal to userId from response", function () {

pm.expect(parseInt(pm.environment.get("userId"))).to.equal(

pm.response.json().data.userId

);

});

В нашем новом тесте мы сравниваем полученный от сервера userId с тем, что хранится у нас в переменной окружения, они должны совпадать. Запустите запрос и убедитесь, что тесты прошли. Хорошо, двигаемся дальше.

**Шаг 3**

Вы могли заметить, что два тестовых скрипта имеют одинаковую проверку формата и статуса:

pm.test("Status is ok, response is json", function () {

pm.response.to.be.ok;

pm.response.to.be.json;

});

Пока мы не зашли слишком далеко, давайте исправим эту ситуацию и перенесем эти тесты на уровень коллекции. Откройте редактирование нашей коллекции, и перенесите проверку формата и статуса во вкладку "Tests", а из запросов их можно удалить. Теперь эти проверки будут автоматически вызываться перед скриптами "Tests" для каждого запроса в коллекции. Таким образом мы избавились от копирования данной проверки в каждый запрос.

**Шаг 4**

Мы научились записывать и получать переменные окружения, настало время перейти к чему-то потяжелее. Давайте воспользуемся встроенной библиотекой tv4 (TinyValidator) и попробуем проверить правильность схемы json объекта. Создайте новый запрос, в качестве URL используйте <https://postman-echo.com/post>, установите метод в значение POST, для тела запроса укажите raw — application/json, и вставьте в него:

{

"profile" : {

"userId": {{userId}},

"username": "Bob",

"scores": [1, 2, 3, 4, 5],

"age": 21,

"rating": {"min": 20, "max": 100}

}

}

После запуска запроса echo-сервер должен вернуть нам нашу json модель в качестве ответа в поле "data", таким образом мы имитируем работу реального сервера, который мог бы прислать нам информацию о профиле Боба. Модель готова, напишем тест, проверяющий правильность схемы:

*// получаем профиль из ответа*

var profile = pm.response.json().data.profile;

*// описываем схему модели*

var scheme = {

*// указываем тип объекта*

"type": "object",

*// указываем обязательные свойства*

"required": ["userId", "username"],

*// описываем свойства*

"properties": {

"userId": {"type": "integer"},

"username": {"type": "string"},

"age": {"type": "integer"},

*// описываем массив*

"scores": {

"type": "array",

*// тип элементов*

"items": {"type": "integer"}

},

*// описываем вложенный объект*

"rating": {

"type": "object",

"properties": {

"min": {"type": "integer"},

"max": {"type": "integer"}

}

}

}

};

pm.test('Schema is valid', function() {

*// валидируем объект profile с помощью правил из scheme*

var isValidScheme = tv4.validate(profile, scheme, true, true);

*// ожидаем, что результат валидации true*

pm.expect(isValidScheme).to.be.true;

});

Готово, мы провалидировали схему объекта profile. Запустите запрос и убедитесь, что тесты проходят.

**Шаг 5**

У нас получился небольшой путь из 3-х запросов и нескольких тестов. Настало время полностью его протестировать. Но перед этим, давайте внесем небольшую модификацию в наш первый запрос: замените значение “777” в URL на плейсхолдер “{{newUserId}}”, и через интерфейс добавьте в окружение "Habra.Env" переменную “newUserId” со значением “777”. Создайте файл users.json на своём компьютере и поместите туда следующий json массив:

[

{"newUserId": 100},

{"newUserId": 200},

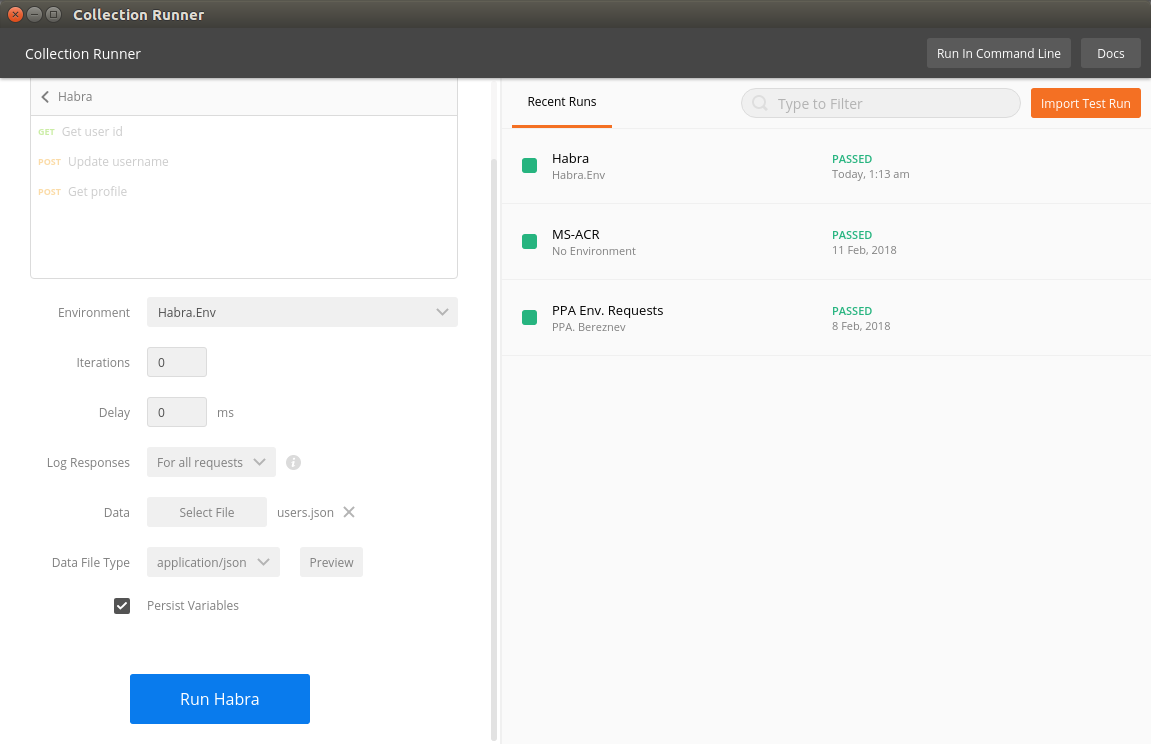
{"newUserId": 300},

{"newUserId": 50000}

]

Теперь запустим Runner. В качестве "Collection Folder" выберем "Habra". В качестве "Environment" поставим "Habra.Env". Количество итераций оставляем 0, а в поле "Data" выбираем наш файл users.json. По этому файлу "Collection Runner" поймет, что ему нужно совершить 4 итерации, и в каждой итерации он заменит значение переменной "newUserId" значением из массива в файле. Если указать количество итераций больше, чем количество элементов в массиве, то все последующие итерации будут происходить с последним значением из массива данных. В нашем случае после 4 итерации значение "newUserId" всегда будет равно 50000. Наш “Data” файл поможет пройти весь наш путь с разными newUserId, и убедиться, что независимо от пользователя все выполняется стабильно.

Запускаем наши тесты нажатием "Run Habra". Поздравляем вы успешно создали коллекцию и протестировали наш импровизированный путь с помощью автотестов!

  
*Тестирование коллекции "Habra"*

Теперь вы знаете, как создавать скрипты и вполне можете написать парочку тестов для своего проекта.

Отдельно хочу отметить, что в реальном проекте мы не держим все запросы в корне коллекции, а стараемся раскладывать их по папкам. Например, в папке "Requests" мы складываем все возможные запросы описывающие наш API, а тесты храним в отдельной папке "Tests". И когда нам нужно протестировать проект, в "Runner" запускаем только папочку "Tests".