**Тестирование API в Postman**

В Postman есть возможность проводить тестирование API с помощь скриптов JavaScript. Для того чтобы создать тест необходимо:

1. Создать коллекцию в Postman и добавить туда необходимый вам endpoint.

2. Зайти в раздел Tests

3. У глобального объекта pm использовать объект test. Он предоставляет доступ для написания тестов. 1 аргумент конструктора — название теста, 2 аргумент — функция для проверки выполнения какого-либо условия. Это делается с помощью другой функции pm.expect(), которая будет принимать какое-либо значение и если оно будет не равно значению справа, тест будет провален. Это своеобразный аналог assert в других ЯП-ах.

pm.response предоставляет доступ к результатам последнего выполненного запроса. Он содержит различные свойства и методы для работы с ответом сервера.

Тест, который будет проверять, чтобы время выполнения запроса не превышало 500 мс, будет выглядеть примерно так:

pm.**test**("Response time is less than 500ms", **function** () {

pm.expect(pm.response.responseTime).to.be.below(500);

});



pm.**test**("Response time is less than 1ms", **function** () {

pm.expect(pm.response.responseTime).to.be.below(1);

});



Можно написать тест, который проверяет пустоту тела запроса:

pm.**test**("Response body is not empty", **function** () {

pm.expect(pm.response.text()).to.not.be.empty;

});



Тест, который проверяет код выполнения запроса:

pm.test("Status code is 200", function () {

pm.expect(pm.response.code).to.equal(200);

});

Можно было записать по другому. В данном случае происходит проверка непосредственно тела запроса, который принадлежит объекту pm, т.ч. то, что принадлежит pm можно записывать без expect:

pm.**test**("Status code is 200", **function** () {

pm.response.to.have.status(200);

});

pm.sendRequst(url, method, headers, function). Этот метод используется для отправки HTTP запросов на сервер из тестов. Он позволяет выполнять асинхронные запросы и обрабатывать результаты. Последний аргумент принимает функцию с 2 аргументами — ошибкой и результатом. В этой функции будет прописана логика для обработки того и того. Пример:

var startPromise **=** **new** Promise(**function**(resolve, reject) {

pm.sendRequest({

url: 'http://localhost:8080/counter/count',

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

}, **function** (err, res) {

**if** (err) {

reject(err);

} **else** {

resolve(res.json());

}

});

});

В данном случае мы записываем результат запроса в функцию startPromise. Таким образом можно прописать нагрузочный тест, алгоритм следующий:

1. Получаем начальное количество запросов

2. Отправляем n запросов на сервер

3. Получаем итоговое количество запросов

4. Проверяем, чтобы end — start был равен n

var startPromise **=** **new** Promise(**function**(resolve, reject) {

pm.sendRequest({

url: 'http://localhost:8080/counter/count',

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

}, **function** (err, res) {

**if** (err) {

reject(err);

} **else** {

resolve(res.json());

}

});

});

**for** (var i **=** 0; i **<** 1000; i**++**) {

pm.sendRequest({

url: 'http://localhost:8080/ips-info/find',

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

}, **function** (err, res) {

console.**log**(res);

});

}

var endPromise **=** **new** Promise(**function**(resolve, reject) {

pm.sendRequest({

url: 'http://localhost:8080/counter/count',

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

}, **function** (err, res) {

**if** (err) {

reject(err);

} **else** {

resolve(res.json());

}

});

});

Promise.**all**([startPromise, endPromise]).**then**(**function**(results) {

var start **=** results[0];

var end **=** results[1];

pm.**test**("Counter work correct", **function** () {

pm.expect(end **-** start).to.eql(1000);

});

});

Таким образом мы можем тестировать API не со стороны сервера, а со стороны клиента. Это позволяет нам проводить нагрузочное тестирование, а так же тестирование под различные платформы, если речь идет о разных браузерах.