**09.1 Postman**

**Создание набора тестов, использующего техники тест-дизайна: классы эквивалентности, граничных значений. Проверка поля ‘age\_type’.**

Выполнил: Герман Буйницкий

**Содержание:**

[**Задача:** 1](#_Toc70158835)

[**Решение:** 2](#_Toc70158836)

[**Реализация:** 2](#_Toc70158837)

[**Запуск коллекции, описание обнаруженных дефектов:** 6](#_Toc70158838)

# **Задача:**

{"Server IP": "116.203.27.46",

"Port": "5002",

"Postman":[{"endpoint": "age\_europe",

"method": "POST",

"body\_params": "age: int",

"age\_class": ["0-17", "18-50", "50-100"],

"task": "Create equivalence classes and boundary values tests"}

{"endpoint": "age\_usa",

"method": "POST",

"body\_params": "age: int",

"age\_class": ["0-20", "21-50", "50-100"],

"task": "Create equivalence classes and boundary values tests"}],

1. Создать набор тестов, использующий техники тест-дизайна: классы эквивалентности, граничных значений для двух запросов age\_europe и age\_usa.

2. Ожидаемые классы:

age\_europe: "age\_class": "0-17", "18-50", "50-100".

age\_usa: "age\_class": "0-20", "21-50", "50-100".

REQUEST:

Метод: POST

Body: "age: int"

RESPONSE example:

"age": "1",

"age\_type": "underage",

"content": [

"Tom & Jerry (Feb. 26)",

"Raya and the Last Dragon (Mar. 5)",

"Peter Rabbit 2: The Runaway (April 2)",

"Rumble (May 14)",

"Minions: The Rise of Gru (July 2)",

"Space Jam 2: A New Legacy (July 16)",

"The Addams Family 2 (Oct. 8)",

"Encanto (Nov. 24)"

]

# **Решение:**

Для начала создадим тест для запроса age\_europe.

В ходе дополнительных уточнений по заданию было выяснено, что пересечение двух классов на возрасте 50 можно трактовать произвольно, поэтому для теста были выбраны диапазоны 18-50, 51-100.

Ожидаемые ответы для классов: ключ age\_type в ответах для каждого класса – разный. 0-17: ‘underage’, 18-50: ‘adult’, 51-100: ‘antiquity’.

Так же было выяснено, что ожидаемый результат работы системы за границами классов: сообщение об ошибке 500.

Таким образом разбиваем тест одного запроса на 4 группы:

* '0-17' (тестовые данные: 10, 0, 17),
* '18-50' (тестовые данные: 25, 18, 50),
* '51-100' (тестовые данные: 70, 51, 100),
* 'Status 500' (тестовые данные: -1, 101, дроби, пробел, пустое значение, спецсимволы).

Для выполнения данной задачи было выбрано решение использовать массивы внутри теста, что увеличивает гибкость для ввода тестовых данных, а также исключает лишнюю документацию для теста. Для запуска теста требуется только коллекция с заданными переменными окружения.

# **Реализация:**

Создаем массив с тестовыми данными:

let arr **=** [10, '0', 17];

Задаем переменную intex и кидаем ее в цикл for, где ей присваевается значение 0, производится сравнение с длиной заданного массива и после выполнения итерации index увеличивается на 1:

let index;

**for** (index **=** 0; index **<** arr.length; **++**index) {

Создаем переменную для тестовых данных testData, которые будут извлекаться из массива. В первой итерации index = 0, поэтому testData присваивается значение первого элемента массива: 10:

    let testData **=** arr[index];

В цикле отправляется запрос с заданными параметрами, где в body запроса закидывается ключ ‘age’ со значением arr[index] (В первой итерации index = 0, поэтому testData присваивается значение первого элемента массива: 10):

    pm.sendRequest({

        url: 'http://116.203.27.46:5002/age\_europe',

        method: 'POST',

        headers: {},

        body: {

            mode: 'formdata',

            formdata: [

                {key: 'age', value: arr[index]}

            ]

        }

    }, **function**(err, res) {

В консоли выводим Age value с тестовыми данными. Делается это для того, чтобы при прогоне большого количества данных в консоли можно было легко найти нужный запрос:

        console.**log**('Age value: ' **+** testData);

Создаем переменную, в которую помещаем номер статус кода response (внутри теста ответы на запросы разбиваются на объекты, из которых выдергивается необходимый элемент. В нашем случае 3 индекс массива, где содержится статус код):

        let statusCode **=** Object.**values**(res)[3];

Делаем проверку на статус код 200. Если в ответе был статус код 200, то создается переменная resType, в которую помещается json ответа. В тесте производится проверка, что в ответе значение ключа age\_type имеет ‘underage’. Таким образом, если в ответе ключ age\_type имеет значение underage, то тест pass:

**if** (statusCode **===** 200) {

                let resType **=** res.json();

                pm.**test**('Age value: ' **+** testData, **function** () {

                pm.expect(resType.age\_type).to.include('underage');

                });

Если же статус код ответа был не равен 200, то делаем тест на проверку статус кода. Тест заведомо будет fail, но благодаря ему мы узнаем, какой код вернулся вместо 200 и будем визуально виеть, с какими тестовыми данными это произошло:

            } **else** {

                pm.**test**('Age value: ' **+** testData, **function** () {

                pm.expect(res).to.have.property('code', 200);

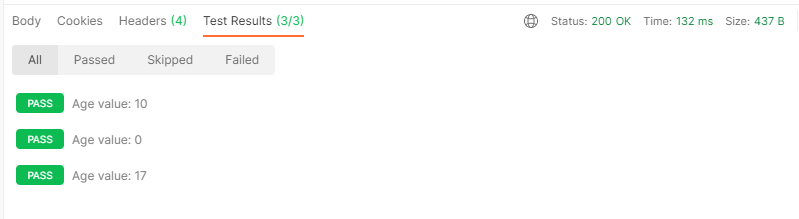
                });

            };

        });

}

Тестируем:



С нашими тестовыми данными все три теста прошли успешно. Внутри цикла все запросы получили ответ, в котором был ключ age\_type и как ожидалось для данного класса, он содержал значение ‘underage’.

Создаем аналогичные тесты для классов 18-50: ‘adult’ и 51-100: ‘antiquity’.

Достаточно изменить тестовые данные внутри массива и изменить строку

pm.expect(resType.age\_type).to.include('underage');

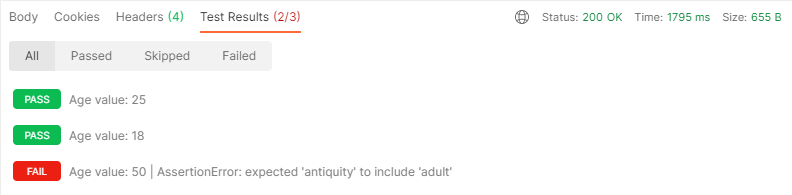
на

pm.expect(resType.age\_type).to.include('adult');

и для теста класса 51-100

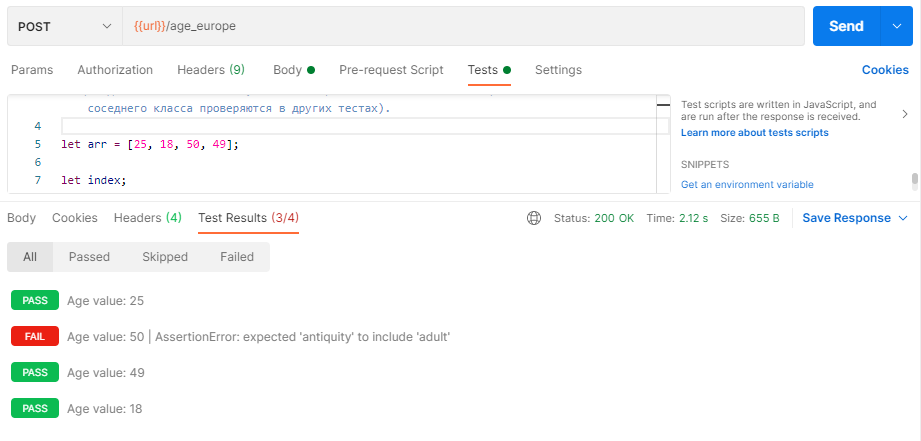
pm.expect(resType.age\_type).to.include('antiquity');

Запускаем тест 18-50 (adult):

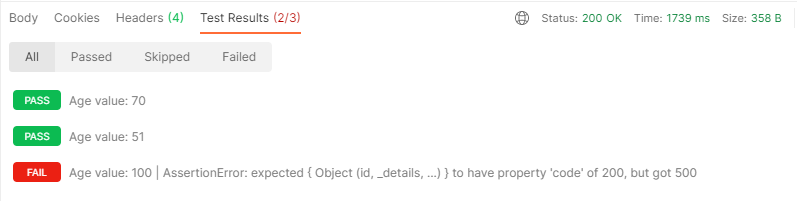


С одним из граничных значений (50) тест fail. В ответе в age\_type пришло antiquity, хотя мы ожидали adult. Найден баг. Добавим проверку с возрастом 49, чтобы выловить границу класса. Достаточно ввести в массив еще один элемент. Тест с 49 – pass:

.



Тестируем 51-100: (antiquity)



С age 100 возвращается ошибка 500. Найден еще один баг. C age 99 тест pass.

Теперь займемся четвертым классом эквивалентности – всем тем, что не входит в три предыдущие класса: от минус бесконечности до -1, от 101 до бесконечности, также дроби, пробел, пустое значение, спецсимволы.

Ожидается, что данные будут возвращать 'Status 500'.

На основе предыдущих тестов создадим новый. В массив закинем тестовые данные:

let arr **=** ['', ' ', **-**1, 101, 1000, **-**1000, 0.5, '0,5', 'abc', '.', '/', '?', ':', '-', '+', '=', '#', '$', '%', '&', ',', '?', '@', '"', ';', '\*'];

let index;

**for** (index **=** 0; index **<** arr.length; **++**index) {

    let testData **=** arr[index];

    pm.sendRequest({

        url: 'http://116.203.27.46:5002/age\_europe',

        method: 'POST',

        headers: {},

        body: {

            mode: 'formdata',

            formdata: [

                {key: 'age', value: arr[index]}

            ]

        }

    }, **function**(err, res) {

        console.**log**('Age value: ' **+** testData);

В данном классе достаточно одного теста: мы проверяем, что статус код ответа – 500.

        pm.**test**('Age value: ' **+** testData, **function** () {

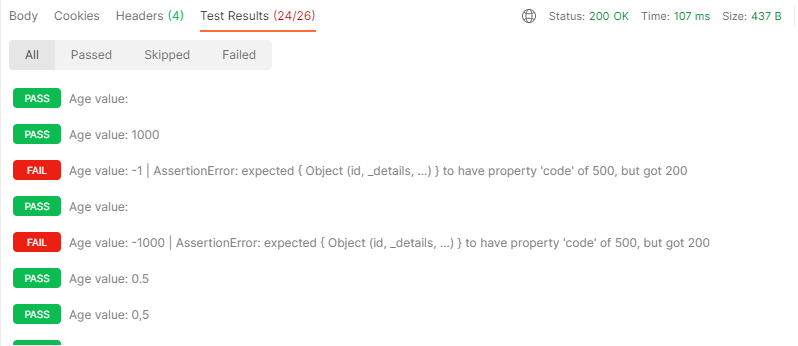
            pm.expect(res).to.have.property('code', 500);

        });

    });

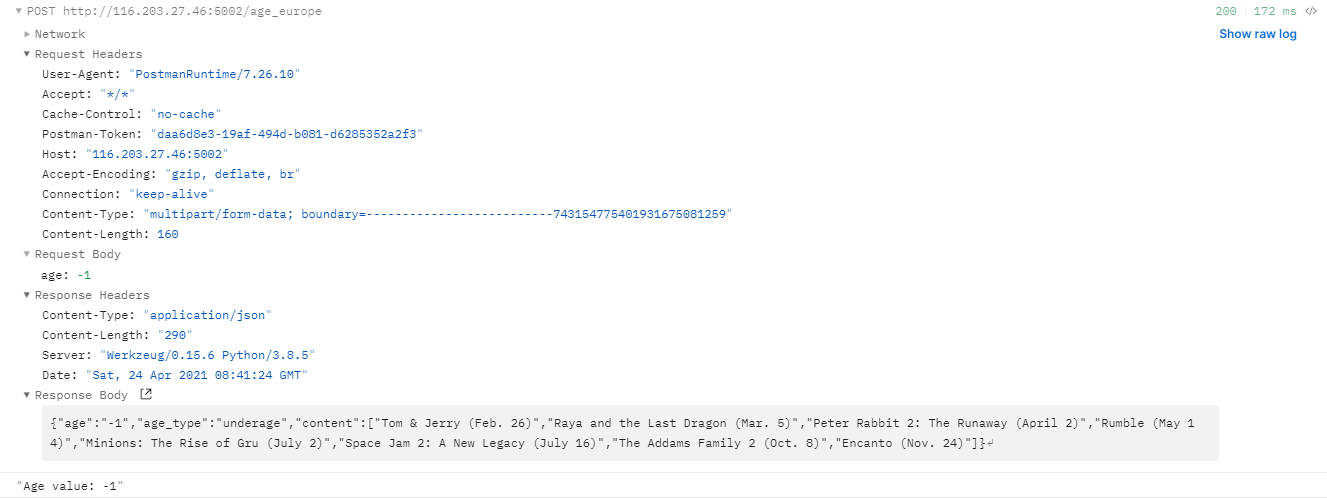
}

Запускаем тест:



Два теста fail: оба с отрицательными числами. -1000 произвольное число из нашего класса, -1 граничное значение класса.

Идем в консоль:



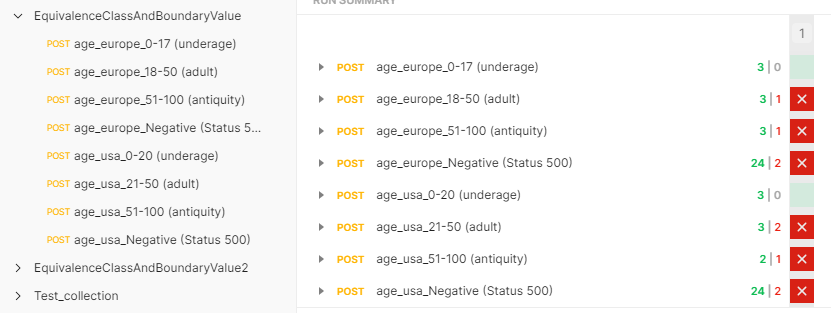
В консоли видим, что система реагирует на отрицательные числа так, будто это несовершеннолетний (underage).

Найден баг.

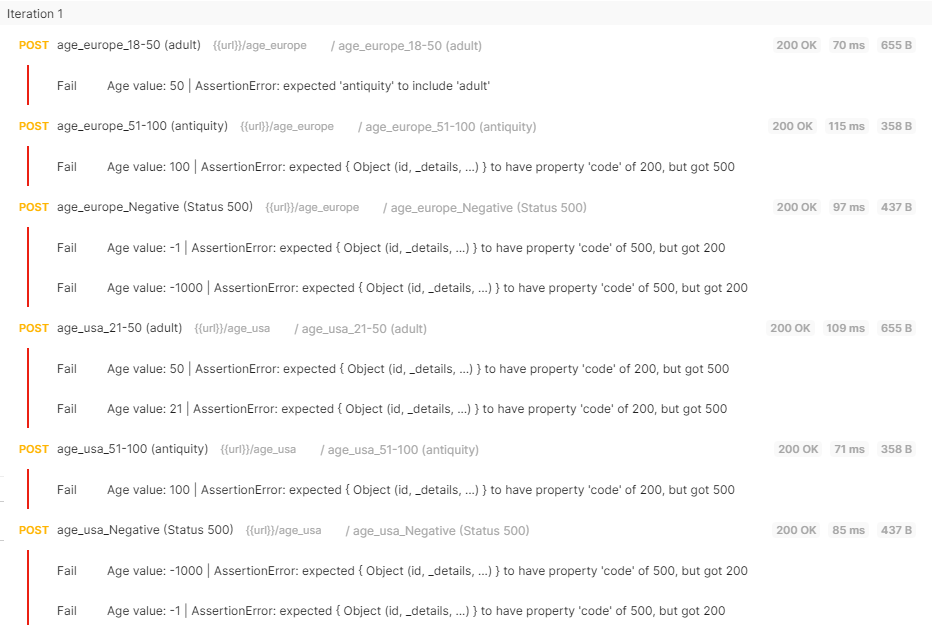
# **Запуск коллекции, описание обнаруженных дефектов:**

Итог: мы имеем 4 теста для 4 классов эквивалентности. Данную коллекцию можно в несколько нажатий запустить через раннер, а в случае необходимости изменения тестовых данных – легко внести их в массив в тесте, не углубляясь в структуру кода.

Для второго запроса age\_usa продублируем данные тесты, сделав поправку в тестовых данных (элементы массива и url в запросе теста), так как у нас для данного запроса ожидаемые ответы для классов: ключ age\_type в ответах для каждого класса – разный. 0-20: ‘underage’, 21-50: ‘adult’, 51-100: ‘antiquity’.



*Результат тестирования*



*Выявленные ошибки*

Проанализируем выловленные дефекты. В первом запросе age\_europe:

* В age\_type тип возраста ‘antiquity’ в ответе запроса, когда в запросе отправляется age: 50. (Ожидаемый результат ‘adult’)
* Ошибка 500 при отправке запроса с age: 100

(Ожидаемый резульатат: status 200, age\_type ‘antiquity’)

* Status 200, age\_type ‘underage’ в ответе запроса, при отправе запроса с age: отрицательное число (-1, -1000)

(Ожидаемый результат: Status 500)

Дефекты в запросе age\_usa:

* Ошибка 500 при отправке запроса с age: 21

(Ожидаемый резульатат: status 200, age\_type ‘adult’)

* Ошибка 500 при отправке запроса с age: 50

(Ожидаемый резульатат: status 200, age\_type ‘adult’)

* Ошибка 500 при отправке запроса с age: 100

(Ожидаемый резульатат: status 200, age\_type ‘antiquity’)

* Status 200, age\_type ‘underage’ в ответе запроса, при отправке запроса с age: отрицательное число (-1, -1000)

(Ожидаемый результат: Status 500)