САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №4

Выполнил:

Власов Владислав

K3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- реализовать автодокументирование средствами swagger;
- реализовать документацию API средствами Postman.

Ход работы

Была составлена серия тестов, описанная ниже:

Авторизация:

```
pm.test("Get access token from response",
    function () {
        const jsonData = pm.response.json();
        pm.response.to.have.jsonBody("token");
        pm.collectionVariables.set("access", jsonData.token);
    }
);
```

Создание эффекта:

```
pm.test("Status code is 201 Created", function() {
    pm.response.to.have.status(201);
});

const requestData = JSON.parse(pm.request.body.raw);
const responseData = pm.response.json();

const DEFAULT_VALUES = {
    description: "",
};

pm.test("Response includes all request fields with correct values", function() {
    for (let field in requestData) {
        pm.expect(responseData).to.have.property(field);
        pm.expect(responseData[field]).to.eql(requestData[field]);
    }
});

pm.test("Non-passed fields should have default values", function() {
    for (let field in DEFAULT_VALUES) {
        if (!requestData.hasOwnProperty(field)) {
            pm.expect(responseData).to.have.property(field);
            pm.expect(responseData[field]).to.eql(DEFAULT_VALUES[field]);
        }
    }
```

```
});

pm.collectionVariables.set("createdEffectId", responseData.id);

{
  "name": "",
  "description": "Гарантировано наносит 1 рану или шок",
  "isTemp": false,
  "ActionTime": 1,
  "type": "attack"
}
```

Создание 2-ух условийч:

```
pm.test("Status code is 201 Created", function() {
    pm.response.to.have.status(201);
});

const requestData = JSON.parse(pm.request.body.raw);
const responseData = pm.response.json();

pm.test("Response includes all request fields with correct values", function() {
    for (let field in requestData) {
        pm.expect(responseData).to.have.property(field);
        pm.expect(responseData[field]).to.eql(requestData[field]);
    }
});

pm.collectionVariables.set("createdConditionId1", responseData.id);

{
  "type": "triggering",
  "targetType": "target",
  "parameter": "isShocked",
  "operand": "==",
  "value": "1",
  "effect": {{createdEffectId}}
}
{
  "type": "effect",
  "targetType": "target",
  "parameter": "damage",
  "operand": "=",
  "value": "toughness + armor",
  "effect": {{createdEffectId}}
}
```

Затем создание предмета:

```
pm.test("Status code is 201 Created", function() {
    pm.response.to.have.status(201);
});

const requestData = JSON.parse(pm.request.body.raw);
const responseData = pm.response.json();

const DEFAULT_VALUES = {
    description: "",
    cost: 0,
    weight: 0
};

pm.test("Response includes all request fields with correct values", function() {
    for (let field in requestData) {
        pm.expect(responseData).to.have.property(field);
        pm.expect(responseData[field]).to.eql(requestData[field]);
    }
});

pm.test("Non-passed fields should have default values", function() {
    for (let field in DEFAULT_VALUES) {
        if (!requestData.hasOwnProperty(field)) {
            pm.expect(responseData).to.have.property(field);
            pm.expect(responseData[field]).to.eql(DEFAULT_VALUES[field]);
        }
    }
});

pm.collectionVariables.set("createdItemId", responseData.id);

{
  "name": "Шокирующий клинок",
  "description": "Всегда ввергнет в шок, но не более",
  "type": "weapon",
  "cost": 25
}
```

И добавление созданного эффекта предмету

```
pm.test("Status code is 201 Created", function() {
    pm.response.to.have.status(201);
});
```

После создаётся 2 персонажа (статист и нет):

```
pm.test("Status code is 201 Created", function() {
    pm.response.to.have.status(201);
});
```

```
const requestData = JSON.parse(pm.request.body.raw);
const responseData = pm.response.json();

const DEFAULT_VALUES = {
    "name": "",
    "weight": 0,
    "agility": 4,
    "smarts": 4,
    "spirit": 4,
    "strenght": 4,
    "vigor": 4,
    "pace": 6,
    "run": 6,
    "parry": 2,
    "toughness": 4,
    "armor": 0,
    "wounds": 0,
    "fatigue": 0,
    "maxFatigue": 2,
    "isShaken": false,
    "isDead": false,
    "maxWeight": 10,
    "hp": 3,
    "wildDice": 6,
    "advance": 0
};

pm.test("Response includes all request fields with correct values", function() {
    for (let field in requestData) {
        pm.expect(responseData).to.have.property(field);
        pm.expect(responseData[field]).to.eql(requestData[field]);
    }
});

pm.test("Non-passed fields should have default values", function() {
    for (let field in DEFAULT_VALUES) {
        if (!requestData.hasOwnProperty(field)) {
            pm.expect(responseData).to.have.property(field);
            pm.expect(responseData[field]).to.eql(DEFAULT_VALUES[field]);
        }
    }
});

pm.collectionVariables.set("createdAttId", responseData.id);



{
  "playerId": 13,
  "isWildCard": true
}
```

```
{
  "playerId": 13,
  "isWildCard": false
}
```

Атакующему (Дикой Карте) выдаётся навык драки:

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

[
    {{createdItemId}}
]
```

А защищающемуся понижается параметр парирования, чтобы атака всегда попадала, при этом сбрасываются параметры, меняемые при атаки для повторных тестов:

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

const requestData = JSON.parse(pm.request.body.raw);
const responseData = pm.response.json();


pm.test("Response includes all request fields with correct values", function() {
    for (let field in requestData) {
        pm.expect(responseData).to.have.property(field);
        pm.expect(responseData[field]).to.eql(requestData[field]);
    }
});
```

```
{
  "parry": 0,
  "isShaken": false,
  "wounds": 0
}
```

После атакующему выдаётся оружие и экипируется

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

Оружие экипируется:

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

```
{
  "status": "equipped"
}
```

на этом этапе важно передавать в качестве параметра не id предмета, а id записи ассоциативной таблицы.

Наконец можно провести саму атаку:

{{baseUrl}}/api/battle/{{createdAttId}}/attack/{{createdDefId}}/

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

И далее проверить, что атака прошла так, как задумано (сколько бы раз подряд не вызывалась конечная точка, результат не должен меняться):

{{baseUrl}}/api/characters/{{createdDefId}}

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

const responseData = pm.response.json();

const DEFAULT_VALUES = {
    isShaken: true,
    wounds: 0
};

pm.test("Non-passed fields should have default values", function() {
    pm.expect(responseData["isShaken"]).to.eql(DEFAULT_VALUES["isShaken"]);
    pm.expect(responseData["wounds"]).to.eql(DEFAULT_VALUES["wounds"]);
});

pm.environment.set("createdEffectId", responseData.id);
```

Вывод

Таким образом, был составлен комплексный тест на корректную работу условий.