Домашнее задание 3

JSON и JSON схема

JSON (JavaScript Object Notation) — один из форматов, принятых для передачи данных по протоколу HTTP. Формат подразумевает, что данные передаются в виде объектов JavaScript (передача происходит в строковом формате) или массива JavaScript (значения имеют любой тип принятый в JavaScript). Например:

```
{
    "name": "Anton,
    "age": 33,
    "pets": [
        {
            "name": "murka",
            "age": 2
        },
        {
            "name": "bobik",
            "age": 7
        }
    ]
}
```

JSON схема — схема данных, передаваемых в формате JSON. Описание того из каких объектов состоит передаваемое значение, какие у объектов есть свойства, и какого они типа. Например, для объекта JSON выше схема выглядит так:

```
{
    "name": string,
    "age": number,
    "pets": [
        {
            "name": string,
            "age": number
        },
        ...
    ]
}
```

JSON схема для домашнего задания

Данные, которые обрабатываются в рамках домашнего задания, описываются следующей JSON схемой:

```
{
  "current_page": number,
  "data": [
    {
      "breed": string,
      "country": string,
      "origin": string,
      "coat": string,
      "pattern": string
    },
    ...
  ],
  "first_page_url": string,
  "from": number,
  "last_page": number,
  "last_page_url": string,
  "links": [
    {
      "url": string,
      "label": string,
      "active": boolean
    },
    ...
  ],
  "next_page_url": string,
  "path": string,
  "per_page": number,
  "prev_page_url": string,
  "to": number,
  "total": number
}
```

Пагинация

Пагинация — процесс разбиения большого объема данных на части и организации к ним доступа через URL-ссылки на разные ресурсы или через указания параметров запросов. Каждая часть данных называется страницей. Как правило URL-ссылка по которой можно получить следующую и предыдущую страницы передаётся вместе с данными на текущей странице. Например, в схеме выше адрес следующей страницы передаётся в свойстве `next_page_url`, а предыдущей `prev_page_url`. Если предыдущей или следующей страницы нет, то соответствующее свойство будет равно `null`.

Задание:

Реализовать функции для считывания данных о породах кошек и вычисления статистики на основе этих данных. Для этого выполнить следующие шаги:

1. Прочитать про функцию `fetch` при помощи которой делаются запросы к серверу тут или тут.
2. Реализовать в отдельном модуле функцию `async loadData()` которая возвращает массив состоящий из элементов поля `data` (схема данных выше) со всех страниц начиная с адреса

"https://catfact.ninja/breeds".

3. Реализовать в отдельном модуле функцию `calcStats(catsInfo)`, которая вычисляет статистику по полю `country`. `catsInfo` — массив из объектов со схемой

```
{
  "breed": string,
  "country": string,
  "origin": string,
  "coat": string,
  "pattern": string
}
```

4. Реализовать в отдельном модуле функцию `async calcStatsFromAPI()`, которая загружает данные с удалённого сервера при помощи функции `loadData` из п.2, передаёт данные в `calcStats(catsInfo)` и возвращает результат работы этой функции.

5. Выполнить тестирование функции `async calcStatsFromAPI()` при помощи фреймворка Jest.

Так как функция внутри вызывает метод `loadData`, который в свою очередь делает запрос на сторонний ресурс, результат функции не может на 100% контролироваться разработчиком (например, сервер, к которому выполняется запрос, будет недоступен во время тестирования), необходимо подменить во время тестирования функцию `loadData` таким образом, чтобы функция не обращалась к удалённому серверу и возвращала всегда одинаковый результат и выполнить тестирование с подменой. Для этого выполните следующие шаги:

1. Установите Jest в папку проекта.

```
npm install --save-dev jest
```

или

```
yarn add --dev jest
```

2. Прочитайте как выполняется тестирование при помощи mock-функций, какие техники существуют ссылка на статью.

Пример:

asyncModule.js

```
async function asyncFunction() {
    return 1;
}
```

```
module.exports.asyncFunction = asyncFunction;
```

consumerAsyncModule.js

```
const asyncModule = require('./asyncModule');

async function consumerAsyncFunction() {
    return asyncModule.asyncFunction().then(x => x + 1);
}

module.exports.consumerAsyncFunction = consumerAsyncFunction;
```

mytest.test.js

```
const asyncModule = require('./asyncModule');
const consumerModule = require('./consumerAsyncModule');

test("test", async () => {
    const fMock = jest.spyOn(asyncModule, "asyncFunction");
    fMock.mockImplementation(() => new Promise((resolve, reject) => resolve(22)));
    let res = await consumerModule.consumerAsyncFunction();
    fMock.mockRestore();
    expect(res).toBe(23);
});
```

3. Выполните подмену функции `loadData` при помощи `jest.spyOn` так, чтобы она возвращала всегда одинаковый массив с данными, например:

```
[
        {
            breed: 'Turkish Van',
            country: 'developed in the United Kingdom (founding stock from
Turkey)',
            origin: 'Natural',
            coat: 'Semi-long',
            pattern: 'Van'
        },
        {
            breed: 'York Chocolate',
            country: 'United States (New York)',
            origin: 'Natural',
            coat: 'Long',
            pattern: 'Solid'
        }
]
```

4. Посмотрите пример тестирования асинхронных функций — можно посмотреть только про async/await

5. Выполните тестирование функции `async calcStatsFromAPI()`. При тестировании проверьте, что функция `loadData` вызывается один раз и результат `calcStatsFromAPI` соответствует ожидаемому.