|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
|  | |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ № 1 — 8** | |
| **по дисциплине** | |
| **«Интерфейсы прикладного программирования»** | |
| Выполнил студент группы **ИКБО-10-20** | ***Яковлева Е.А.*** |
| Принял преподаватель кафедры ИиППО | ***Зарипов Е.А.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2022г. |  |
| «Зачтено» | « » 2022г. |  |

Москва 2022

**ОГЛАВЛЕНИЕ**

[Практическая работа №1 3](#_Toc114758394)

[Цель работы 3](#_Toc114758395)

[Постановка задачи 3](#_Toc114758396)

[Выполнение задачи 3](#_Toc114758397)

[Выводы 4](#_Toc114758398)

[Практическая работа №2 5](#_Toc114758399)

[Цель работы 5](#_Toc114758400)

[Постановка задачи 5](#_Toc114758401)

[Выполнение задачи 5](#_Toc114758402)

[Выводы 10](#_Toc114758403)

# Практическая работа №1

## Цель работы

Целью данной работы является первичное знакомство студентов с интерфейсами прикладного программирования и их применением на практике при разработке программных систем.

## Постановка задачи

Используя любой публичный WEB API (большой список API, например, можно взять на сайте https://any-api.com/), создать приложение, демонстрирующие возможности выбранного API. Ограничений по используемым языкам, готовым библиотекам, упрощающим работу с API нет, но желательно использовать языки JAVA или Python.

## Выполнение задачи

Выбранное API: OpenWeatherMap.org.

Выбранный язык программирования: Python.

Создадим приложение, способное получать и выводить прогноз погоды на день и на 5 дней в зависимости от введённого города (Листинг 1).

Листинг 1

import requests  
  
s\_city = "Samara,RU"  
city\_id = 0  
appid = "3fdfb29fa01781d86e50693b4555ef02"  
try:#index of sity  
 res = requests.get("http://api.openweathermap.org/data/2.5/find",  
 params={'q': s\_city, 'type': 'like', 'units': 'metric', 'APPID': appid})  
 data = res.json()  
 cities = ["{} ({})".format(d['name'], d['sys']['country'])  
 for d in data['list']]  
 print("Город:", cities)  
 city\_id = data['list'][0]['id']  
 print('id =', city\_id)  
except Exception as e:  
 print("Exception (find):", e)  
 pass  
try:#today's weather  
 res = requests.get("http://api.openweathermap.org/data/2.5/weather",

*Продолжение листинга 1*

params={'id': city\_id, 'units': 'metric', 'lang': 'ru', 'APPID': appid})  
 data = res.json()  
 print("Погода:", data['weather'][0]['description'])  
 print("Температура:", data['main']['temp'])  
except Exception as e:  
 print("Exception (weather):", e)  
 pass

Город или другой населенный пункт можно менять в поле s\_city с указанием страны после запятой.

При запросе погоды делаем замену на метрические единицы: «units=metric». Описание погоды получаем на русском, поэтому указываем «lang=ru».

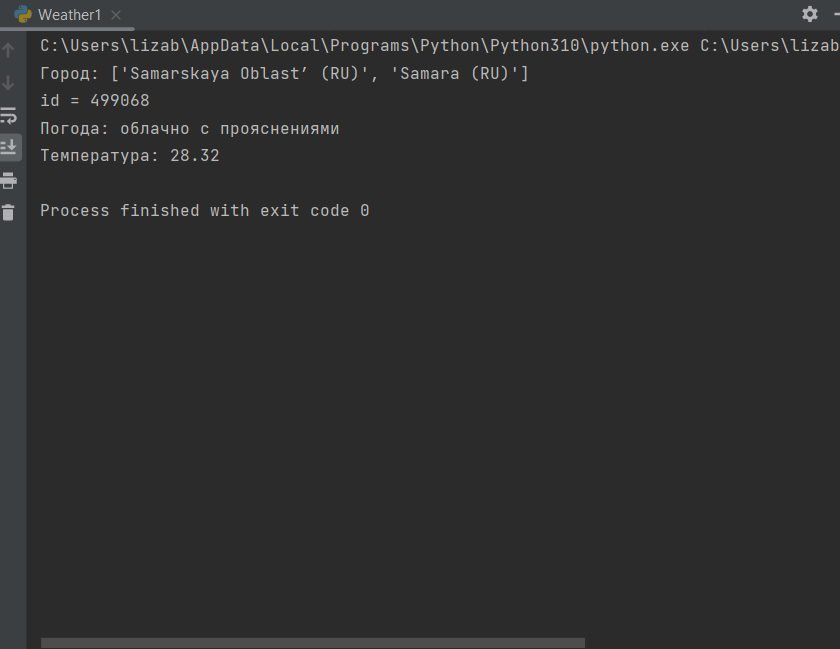


Рисунок 1 – Результат работы программы

## Выводы

В результате выполнения данной практической работы было создано приложение, демонстрирующие возможности OpenWeatherMap.org API.

# Практическая работа №2

## Цель работы

Целью данной работы является закрепление на практике навыков работы с публичными API, в части тестирования URL-адресов конечных точек REST для получения доступа к API, а также первичное знакомство студентов с графическими интерфейсами для выполнения запросов и более подробным описанием RESTful webAPI.

## Постановка задачи

1. Получить список поддерживаемых API Currency Layer валют.

2. Необходимо протестировать API Currency Layer путём создания запроса об исторических курсах валют для 22 февраля 2018 года, для евро, фунтов стерлингов и иен с исходной валютой доллар США.

3. Получить исторические данные о курсе евро к доллару США, начиная с 25 февраля 2016 по 21 февраля 2017 года.

Дополнительным заданием требуется ознакомиться с API поставщика погоды openweathermap, получив текущие погодные данные для любого выбранного местоположения и протестировать запрос на 16 дневний прогноз для любого города (https://openweathermap.org/forecast16).

Попробуйте протестировать API других поставщиков погоды (Gismeteo, Яндекс.Погоды и т. д.).

## Выполнение задачи

Выбранное API: API Currency Layer.

Сначала получаем ключ авторизации API Currency Layer и добавляем его в header на вкладке «Authorization».

Введем API Key в authorization в программе Postman. В поле Key вводим apikey, а в value сам API Key (Рисунок 2.1).

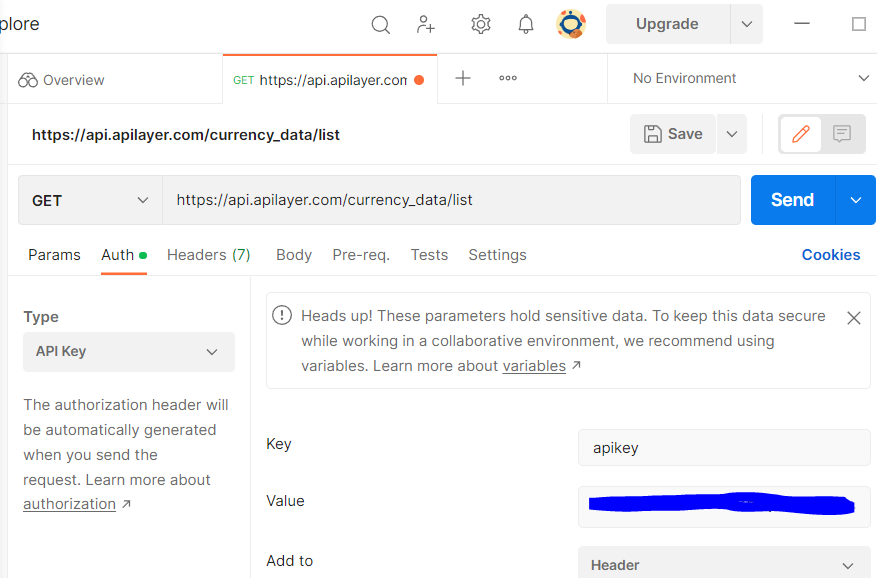


Рисунок 2.1 - Реализация 1 запроса

Создадим запрос об исторических курсах валют для 22 февраля 2018 года, для евро, фунтов стерлингов и иен с исходной валютой доллар США (Рисунок 2.2).

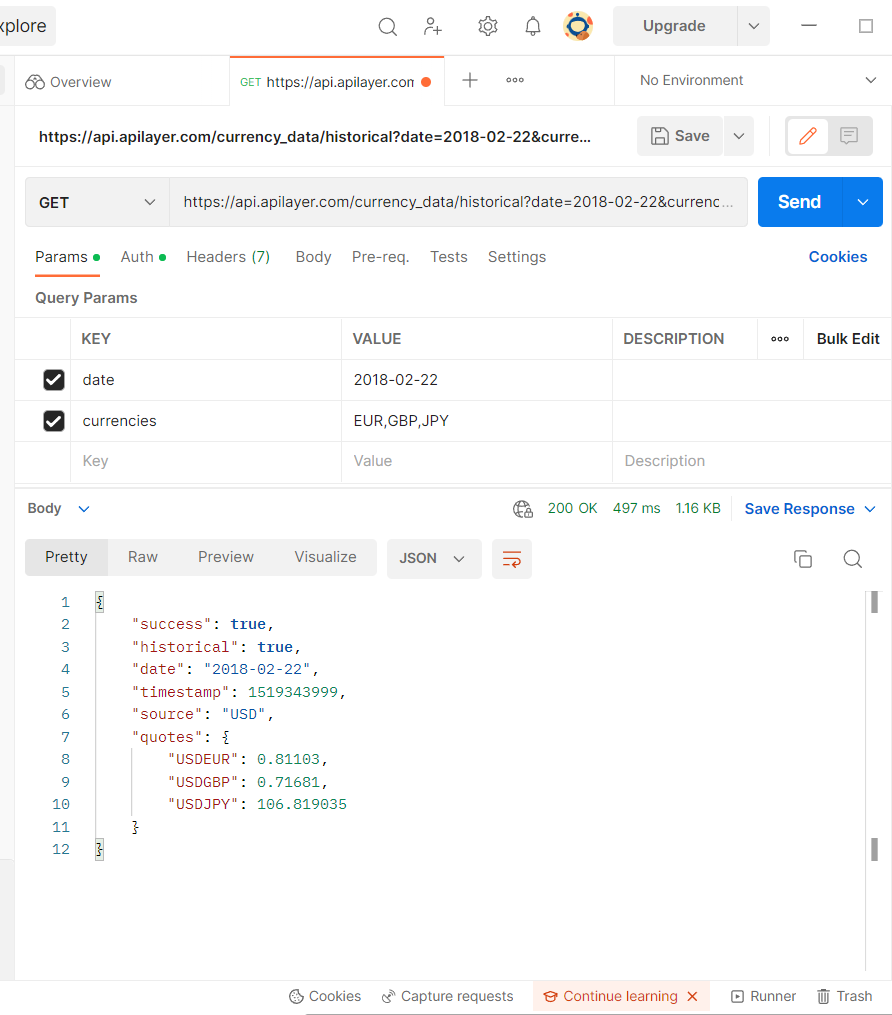


Рисунок 2.2 - Реализация 2 запроса

Получим исторические данные о курсе евро к доллару США, начиная с 25 февраля 2016 по 21 февраля 2017 года (Рисунок 2.3).

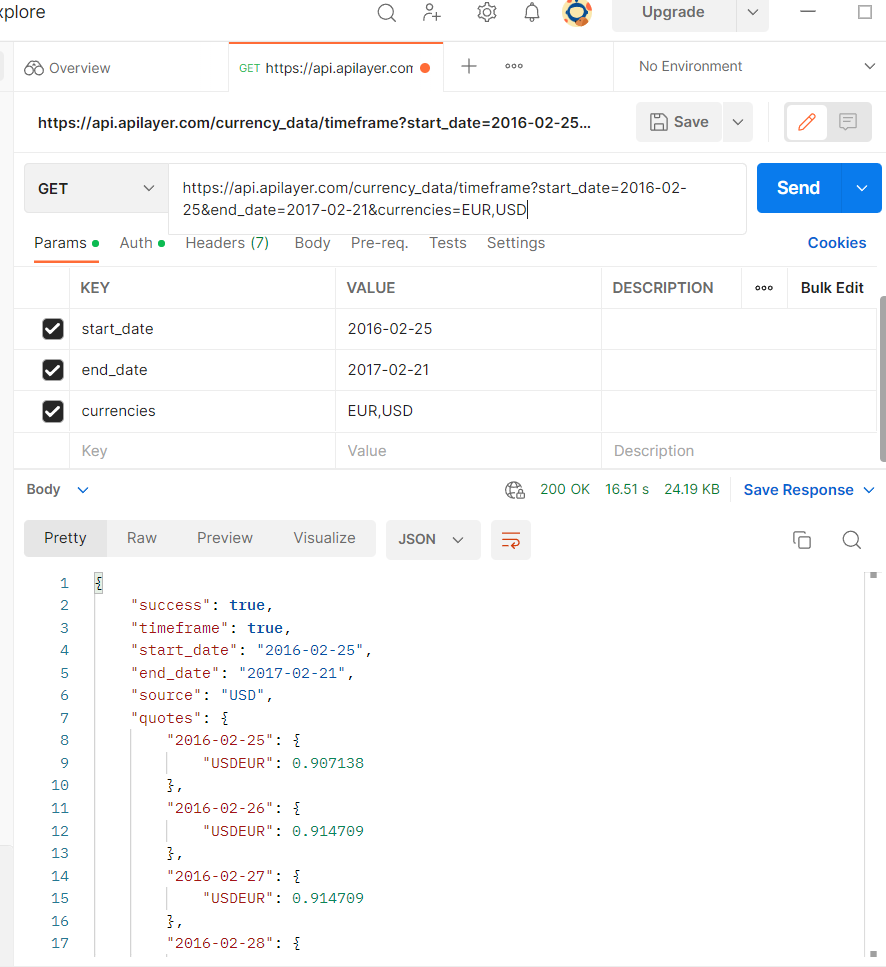


Рисунок 2.3 - Реализация 3 запроса

Выбранное API: openweathermap.

Сначала получаем ключ авторизации openweathermap API, в данном случае при регистрации на сайте openweathermap.

На рисунке 2.4 показан интерфейс Postman с установленными настройками и ответом в нижней панели.

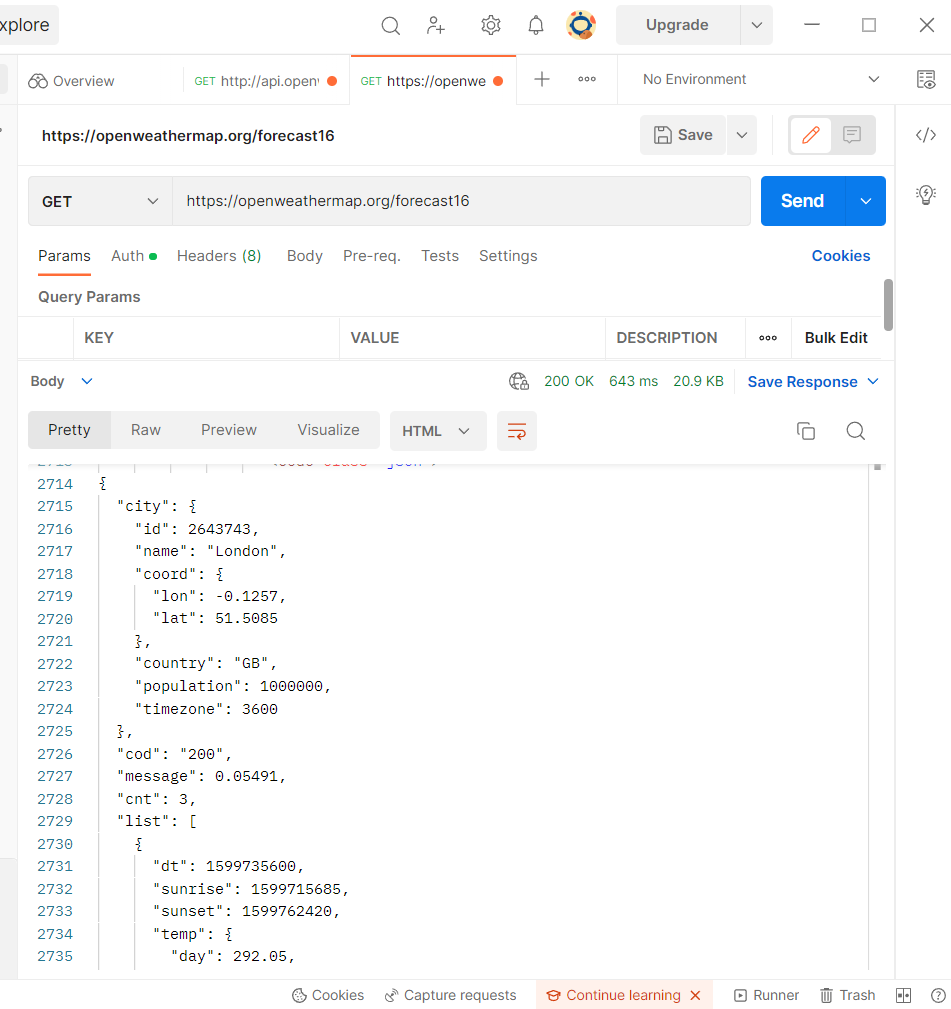


Рисунок 2.4 - Интерфейс Postman с ответом на запрос (openweathermap API)

Выбранное API: visualcrossing.

Сначала получаем ключ авторизации visualcrossing API, в данном случае при регистрации на сайте visualcrossing.

На рисунке 2.5 показан интерфейс Postman с установленными настройками и ответом в нижней панели.

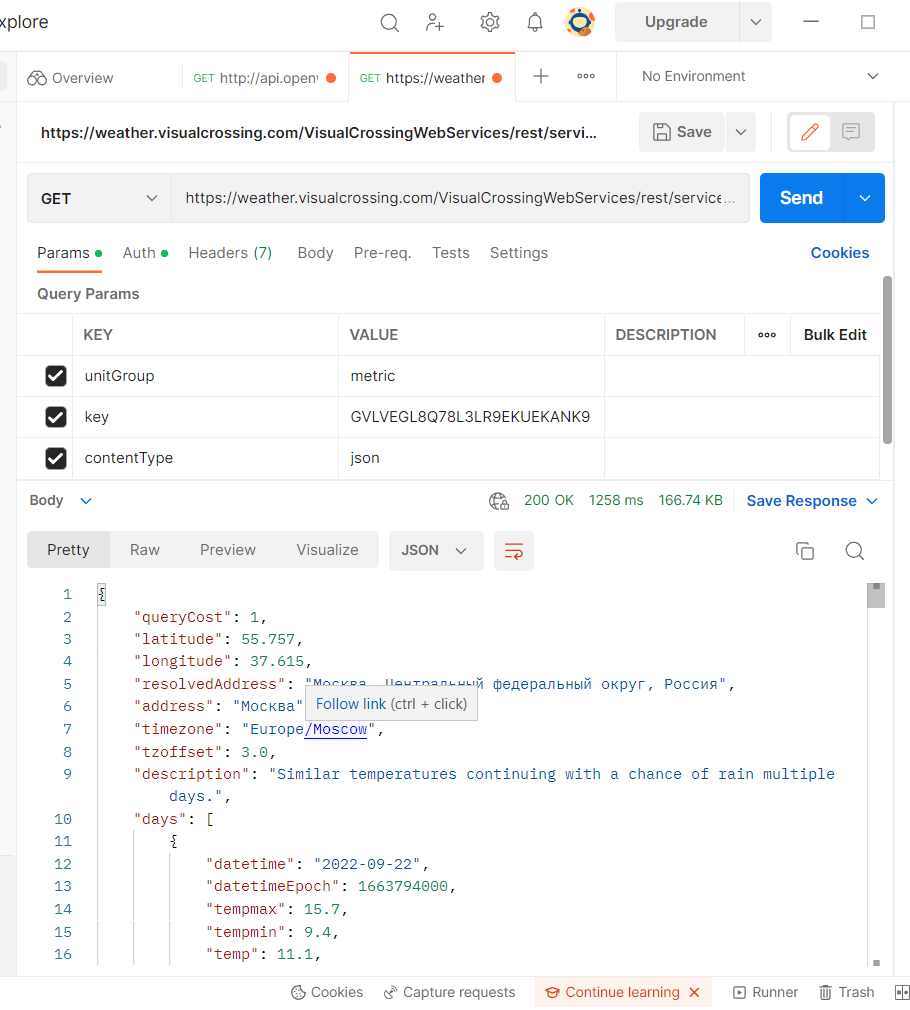


Рисунок 2.5 - Интерфейс Postman с ответом на запрос (openweathermap API)

## Выводы

В результате выполнения данной работы была протестирована работа openweathermap API путём создания запроса о текущих погодных данных для выбранного местоположения и сервиса погоды и API Currency Layer путём создания запроса об исторических курсах валют.

# Практическая работа №3

## Цель работы

Целью данной практической работы является первичное знакомство студентов с популярным инструментом командной строки для передачи файлов с синтаксисом URL – cURL.

## Постановка задачи

Необходимо создать веб-страницу, на которой делается запрос к API при помощи Asynchronous Javascript And Xml (AJAX). В качестве примера рассмотрим создание веб-страницы с информацией о текущем прогнозе погоды, полученной от поставщика openweathermap.org., а потом добавить информацию о максимальной и минимальной температуре, давлении и ветре.

## Выполнение задачи

Для начала выполним методические указания, чтобы создать базовую веб-страницу.

1. В текстовом редакторе создаем новый html-файл и вставляем в него следующий код, содержащий основные теги HTML, а также подключение библиотеки jQuery версии 3.6.0 и jQueryUI, что показано в листинге 3.1.

*Листинг 3.1 – Пример каркаса страницы для отображения погодных данных*

<!doctype html>

<html>

<meta charset="UTF-8">

<head>

<title>Карскас страницы с погодой</title>

<link rel="stylesheet" href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/themes/smoothness/jquery-ui.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js">

Здесь вставляем код

</script>

*Продолжение Листинга 3.1*

</head>

<body>

<h1>Страница с погодой</h1>

temp: <span id="Temp"></span>

</body>

</html>

2. Далее необходимо добавить содержимое скрипта, для того чтобы ajax (функция jQuery) получала ответ от API и назначала ответ response. Для этого также можно воспользоваться функционалом Postman. После формирования запроса и его проверки в Postman, можно нажать на кнопку «Code». В выпадающем меню необходимо выбрать jQuery, в результате чего будет сформирован каркас скрипта, как показано на рисунке 3.1.

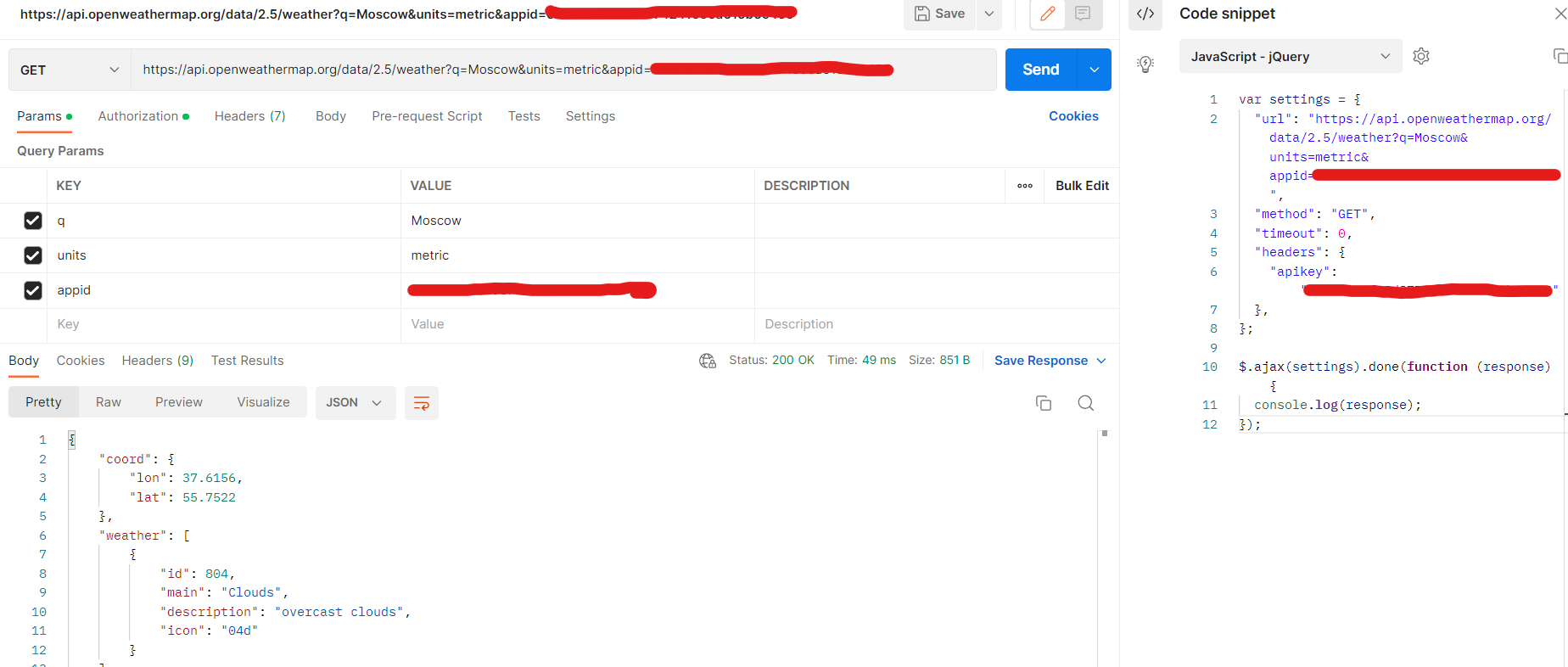


Рисунок 3. - Каркас скрипта

3. Далее сделаем несколько правок, как показано в листинге 3.2.

*Листинг 3.2 – Скрипт для вывода в консоль*

<script>

var settings = {

"url": "https://api.openweathermap.org/data/3.0/weather?q=Moscow&units=metric&appid=КЛЮЧ",

"method": "GET",

"timeout": 0,

};

*Продолжение Листинга 3.2*

$.ajax(settings).done(function(response, xhr) {

console.log("Return Code: " + xhr.status);

var content = response.main.temp;

$("#Temp").append(content);

});

</script>

4. В Chrome переходим в «Файл» -> «Открыть файл» и выбираем файл, где сохранен исходный код страницы.

5. Страница выглядит следующим образом, как показано на рисунке 3.2.

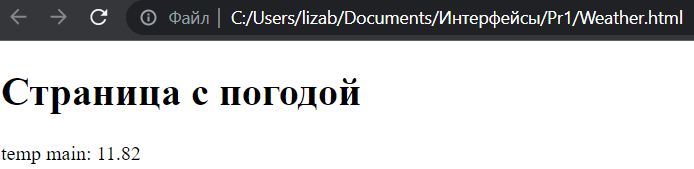


Рисунок 3.2 - Страница с текущей температурой в городе Москва

6. Теперь добавим информацию о максимальной и минимальной температуре, давлении и ветре (Листинг 3.3), и выведем ее на страницу (Рисунок 3.3).

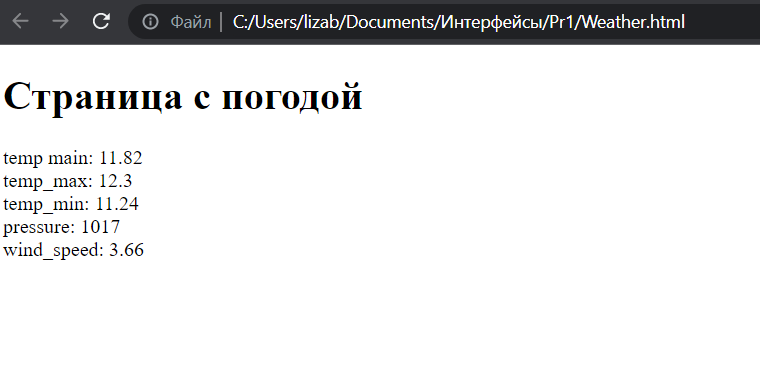


Рисунок 3.3 - Новая версия страницы

*Листинг 3.3 – Код страницы с внесенными изменениями*

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Каркас</title>

    <link rel="stylesheet" href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/themes/smoothness/jquery-ui.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <script>

    var settings = {

        "url": "https://api.openweathermap.org/data/2.5/weather?q=Moscow&units=metric&appid=3fdfb29fa01781d86e50693b4555ef02",

        "method": "GET",

        "timeout": 0,

        };

    $.ajax(settings).done(function(response, xhr) {

    console.log("Return Code: " + xhr.status);

    var content = response.main.temp;

    $("#Temp").append(content);

    content = response.main.temp\_max;

    $("#Temp\_max").append(content);

    content = response.main.temp\_min;

    $("#Temp\_min").append(content);

    content = response.main.pressure;

    $("#Pressure").append(content);

*Продолжение Листинга 3.3*

content = response.wind.speed;

    $("#Wind\_speed").append(content);

    });

    </script>

</head>

<body>

<h1>Страница с погодой</h1>

temp main: <span id="Temp"></span><br>

temp\_max: <span id="Temp\_max"></span><br>

temp\_min: <span id="Temp\_min"></span><br>

pressure: <span id="Pressure"></span><br>

wind\_speed: <span id="Wind\_speed"></span>

</body>

</html>

## Выводы

В результате выполнения данной работы была создана веб-страница, на которой делается запрос к API при помощи Asynchronous Javascript And Xml (AJAX) и отображается информация о текущем прогнозе погоды, полученной от поставщика openweathermap.org.

# Практическая работа №4

## Цель работы

Целью данной практической работы является знакомство с принципами REST и создание простейшего имитационного REST API.

## Постановка задачи

1. Проверить, установлен ли NodeJS. В случае отсутствия установить с официального сайта разработчика.

2. Необходимо с помощью json-server создать имитационный API для сервиса поиска книг в библиотеки. Для этого необходимо создать, как минимум три конечных точки: авторы, книги и библиотеки.

3. Протестировать работу с локальным сервисом, путём соответствующих запросов. Дать объяснение каждому запросу и возвращаемому ответу. Для этого желательно использовать curl или Postman.

## Выполнение задачи

В качестве основного средства для создания простейшего имитационного REST API будем использовать связку Node.js (https://nodejs.org/en/) и пакет JSON server, который доступен в виде пакета NPM (node package manager). NPM представляет собой менеджер пакетов, который идёт вместе с Node.js.

После того как скачен с официального сайта разработчика и развёрнут Node.js необходимо посредством утилиты Node.js command promt создать директорию для проекта, перейти в данную директорию и создать в ней файл со следующим содержанием, как показано в листинге 4.1.

Листинг 4.1 – Содержимое файла package.json

{ "name": "testAPI",

"version": "0.0.1",

"description": "JSON API",

"main": "index.js",

"scripts": {

Продолжение Листинга 4.1

"test": "echo \"NODE\_ERROR\" && exit 1"

},

"author": "DevNULL",

"license": "GNU General Public License"}

Далее необходимо установить глобально пакет json-server, выполнив команду npm install -g json-server --save, как показано на рисунке 4.1.

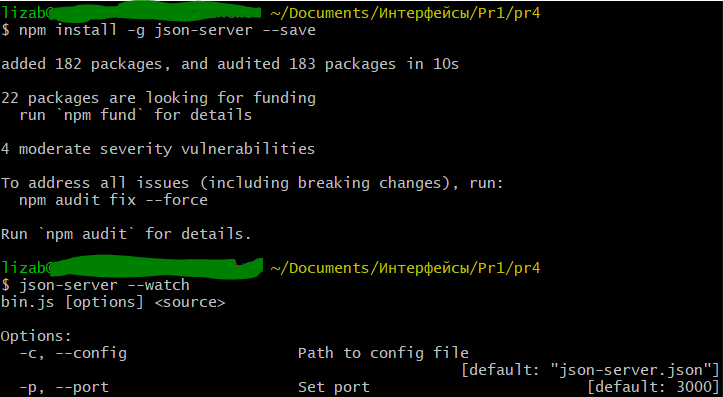


Рисунок 4.1 – Установка пакета json-server

Далее создадим источник данных для имитационного API. Для этого в директории нашего проекта создадим JSON файл и назовём его database.json. Содержание данного файла должно быть следующим, как показано в листинге 4.2.

Листинг 4.2 – Содержимое JSON файла источника данных

{

    "books": [

    {

    "id": 0,

    "name": "Евгений Онегин",

    "author": "Пушкин",

Продолжение листинга 4.2

    "library": "Российская государственная библиотека"

},

    {

      "id": 1,

      "name": "Война и мир",

      "author": "Толстой",

      "library": "Российская государственная библиотека"

    },

    {

      "id": 2,

      "name": "Алые паруса",

      "author": "Грин",

      "library": "Российская государственная детская библиотека"

    },

  {

      "id": 3,

      "name": "Дубровский",

      "author": "Пушкин",

      "library": "Научная библиотека МГУ"

    }

    ]

  }

Для добавления дополнительных маршрутов помимо маршрута «http://localhost:3000/books», воспользуемся --routes. Для этого создим файл routes.json, в котором перечислим все добавляемые маршруты, при этом каждый новый добавляемый маршрут должен начинаться с «/», как показано в листинге 4.3.

Листинг 4.3 – Содержимое JSON файла источника данных

{

"/books/name/:name": "/books?name=:name",

"/books/author/:author": "/books?author=:author",

"/books/library/:library": "/books?library=:library"

}

Далее необходимо запустить выполнив следующую команду json-server database.json --routes routes.json, как показано на рисунке 4.2.

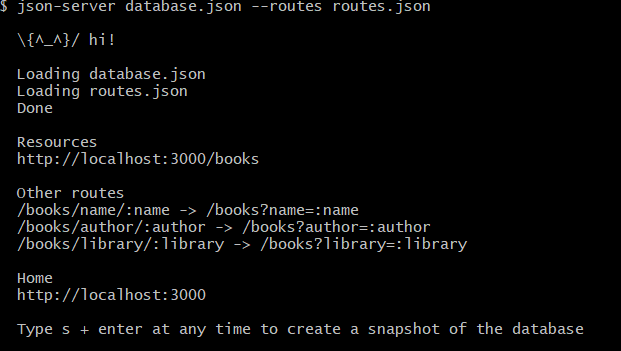


Рисунок 4.2 – Установка пакета json-server

Сформируем GET запрос для получения книг с автором Пушкин: http://localhost:3000/books/author/Пушкин. Ответ должен содержать массив всех книг с указанным автором, как показано на рисунке 4.3.

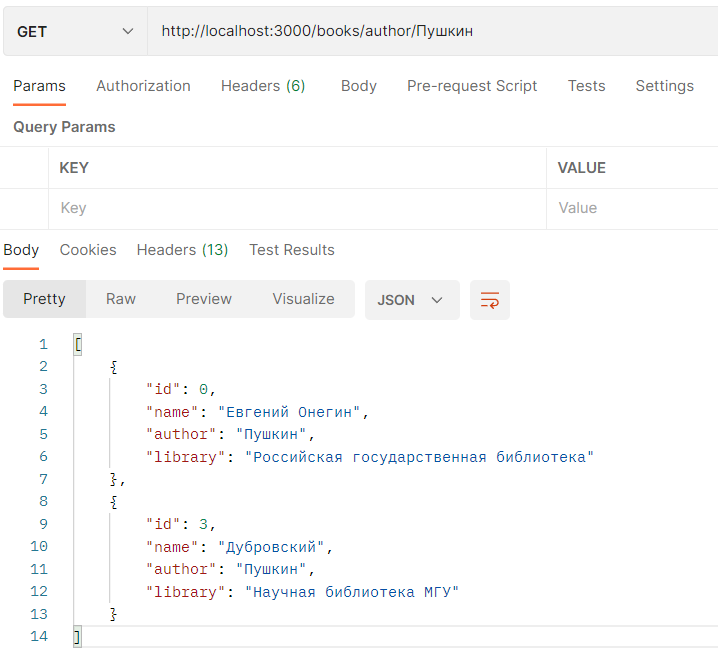


Рисунок 4.3 – GET-запрос на получение списка книг по автору

Далее попробуем изменить библиотеку для книги с названием «Алые паруса». Для этого сделаем PUT-запрос с телом Body, как показано в листинге 4.4.

Листинг 4.4 – Содержимое тела Body для PUT-запроса

{

"id": 2,

"name": "Алые паруса",

"author": "Грин",

"library": "library"

}

В настройках Postman во вкладке Body выберем «RAW» и в выпадающем списке изменим TEXT на JSON, как показано на рисунке 4.4.

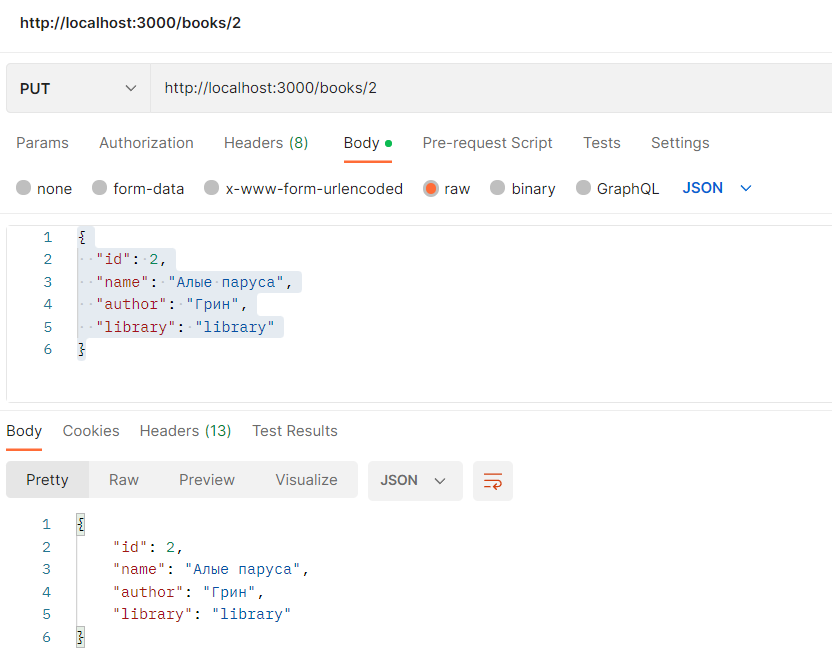


Рисунок 4.4 – Результат выполнения PUT-запроса

В результате в json файле сохранятся изменения, как показано на рисунке 4.5.

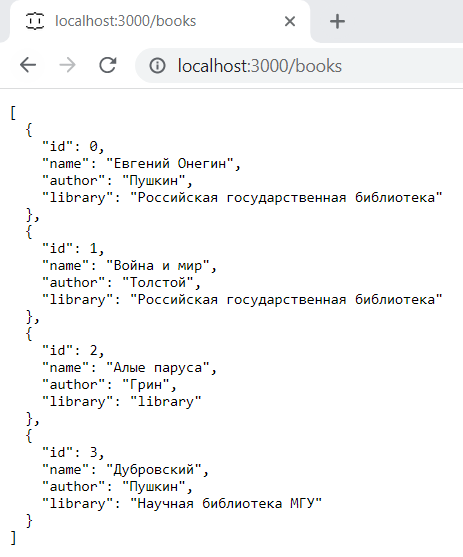


Рисунок 4.5 – Изменённый файл JSON

Сформируем GET запрос для получения книг из библиотеки «Российская государственная библиотека»: http://localhost:3000/books/library/Российская государственная библиотека. Ответ должен содержать массив всех книг с указанной библиотекой, как показано на рисунке 4.6.

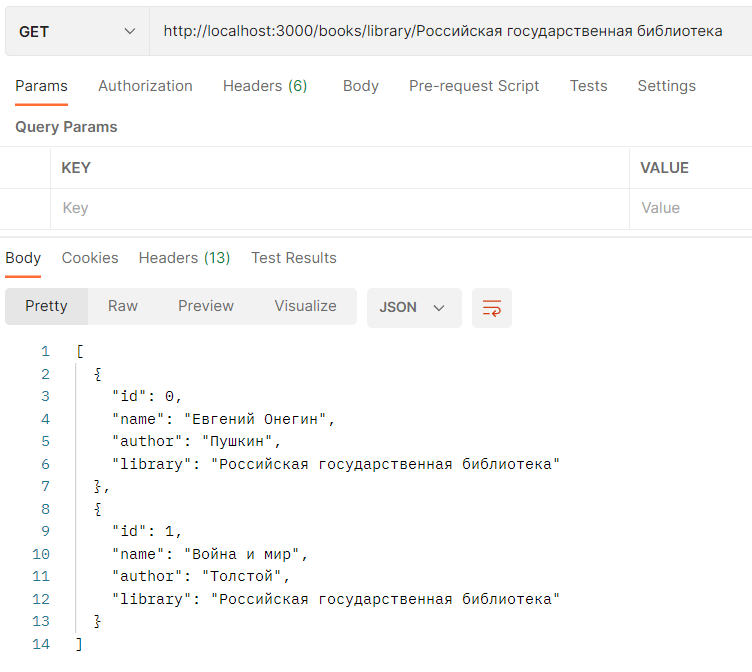


Рисунок 4.6 – GET-запрос на получение списка книг по библиотеке

Сформируем GET запрос для получения книг из той же библиотеки, отсортированных по названию: http://localhost:3000/books/library/Российская государственная библиотека&\_sort=name. Ответ должен содержать массив всех книг с указанной библиотекой, как показано на рисунке 4.7.

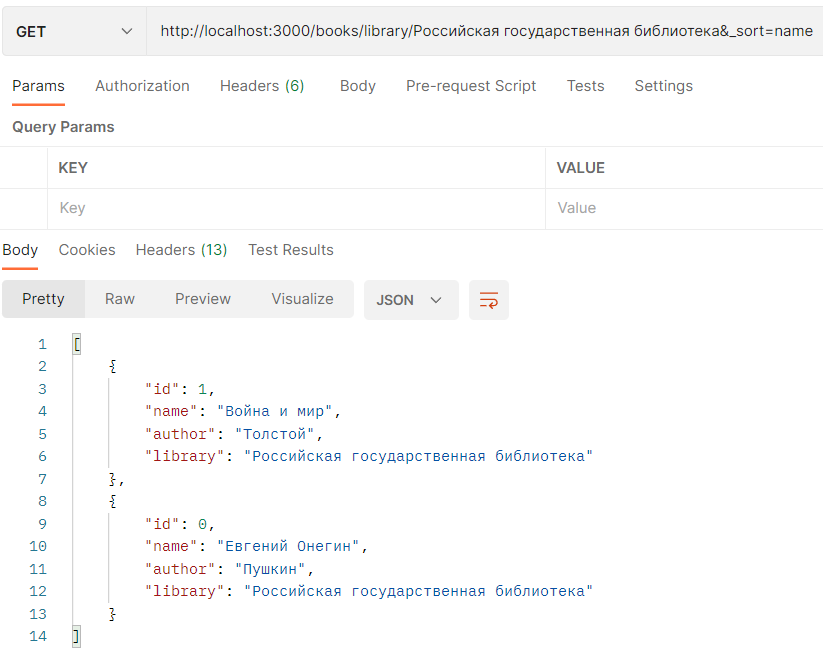


Рисунок 4.7 – GET-запрос на получение по библиотеке отсортированного по названию списка книг

## Выводы

В результате выполнения данной практической работы с помощью jsonserver был создан имитационный API для сервиса поиска книг в библиотеки, были созданы три конечные точки: авторы, книги и библиотеки, была протестирована работа с локальным сервисом путём соответствующих запросов.