|  |  |
| --- | --- |
| |  | | --- | | МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |

Институт Информационных Технологий

Кафедра корпоративных информационных систем (КИС)

**ОТЧЕТ**

по Практическим работам

по дисциплине

«Мировые информационные ресурсы»

Выполнил студент группы ИКБО-08-18 Валяев Д.А.

Принял к.т.н., доцент Крюков Д.А.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_2021 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_2021 г.

**Практическая работа №1**

1. В качестве опытной поисковой системы была выбрана поисковая система «Google».

2. Ознакомиться с документацией к языку расширенных запросов выбранной поисковой системы можно по ссылке «https://support.google.com/websearch/answer/2466433?hl=ru».

3. Примеры поисковых запросов:

- Заключение слова или фразы в кавычки позволяет осуществлять дословные поиск, например, выполним запрос "Хорошее воспитание — это умение скрывать, как много мы думаем о себе и как мало о других.", ответом на данный запрос будет список сайтов, где использовалась данная цитата.

- Слово «site:» позволяет выполнять поиск только на определенном сайте или домене, если добавить «site:.ru» запросу из прошлого пункта, то ответом на данный запрос будет список сайтов на домене «.ru», которые использовали цитату.

- Использование символа «\*» заменяет пропущенное слово, подобный поисковый запрос может быть полезен, когда необходимо найти цитату, но вы забыли какое-то слово, например, если заменить слово «воспитание» в запросе из первого пункта, первым ответом будет искомая цитата.

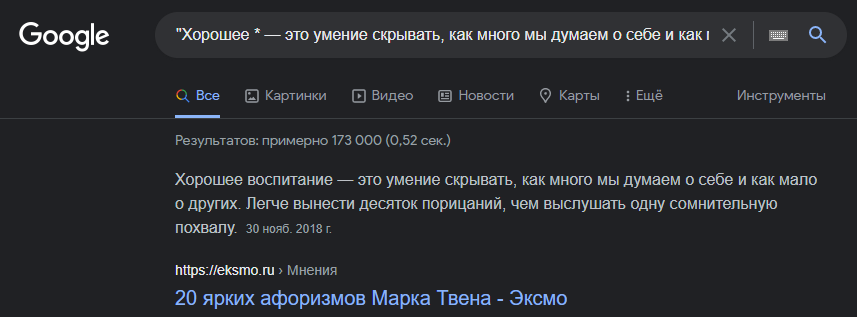


Рис 1.1 – поисковый запрос с пропущенным словом

- Символ «+» позволяет добавить слово, которое должно содержаться в отображенные результатах, например, запрос «Памятник Пушкину +бульвар +октябрьский», вернет список источников, в которых описывается памятник, стоящий именно на Октябрьском бульваре

- Символ «-» дает возможность указать слово, которого не должно быть в результатах поиска, например, запрос «Памятник Пушкину -"Октябрьский бульвар"» вернет список источников, где упоминается памятник Пушкину, но не упоминается Октябрьский бульвар.

- Использование символа «|» позволяет указать несколько слов/фраз, результаты поиска должны содержать хотя бы одно из этих них, например, запрос «Памятник Пушкину ["Октябрьский бульвар"|"Пушкинская площадь"]» вернет список источник, в которых есть упоминание памятника Пушкину на Октябрьском бульваре или Пушкинской площади.

4. Система семантического поиска учитывает:

- текущие тенденции;

- вариации слов («техника» - тс, «техники»-методики, «техник» - должность);

- местонахождение пользователя;

- концептуальное множество;

- синонимы;

- простой язык при запросах (по запросу «погода в Москве» необходимы выводить не сайты, где встречаются слова «погода» и «Москва», а выдать погоду в Москве).

5. В качестве опытных семантических поисковых систем были выбраны Sensebot и WolframAlpha.

6. Для сравнения выбранных систем выполним в них запрос «How many days in september», обе системы вывели результаты, соответствующие ожиданию, следовательно, достоверность – 1, обе системы выдали схожие результаты, полнота ответов:

Sensebot - 50%

WolframAlpha – 50%

7. Для определение средней полноты и достоверности поисковой системы, необходимо выполнить несколько запросов и сравнить результаты выдачи систем.

Первым запросом стал «How many days in year». WolframAlpha chfpe вывел 365, в то время как Sensebot выводил статьи на тему запроса.

Достоверность:

Sensebot – 1

WolframAlpha – 1

Полнота:

Sensebot - 40%

WolframAlpha – 60%

Второй запрос «number pi», обе системы вывели схожие результаты, но результат выдачи WolframAlpha был более полным.

Достоверность:

Sensebot – 1

WolframAlpha – 1

Полнота:

Sensebot – 40%

WolframAlpha – 60%

Третий запрос «volume of pyramid», обе системы вывели схожие результаты.

Достоверность:

Sensebot – 1

WolframAlpha – 1

Полнота:

Sensebot – 50%

WolframAlpha – 50%

По результатам выполнения нескольких запросов можно подсчитать среднюю достоверность и полноту.

Средняя достоверность:

Sensebot – 1

WolframAlpha – 1

Средняя полнота:

Sensebot – 43%

WolframAlpha – 57%

**Практическая работа №2**

1. Первым шагом необходимо создать учетную запись DockerHub.

2. В качестве материнской операционную системы для развёртывания контейнеров Docker используется ОС Windows.

3. Ознакомиться с документацией по развертыванию ПО Docker на

выбранной операционной системе можно по ссылке https://docs.docker.com/.

4. После установки необходимо проверить статуса демона/службы Docker.

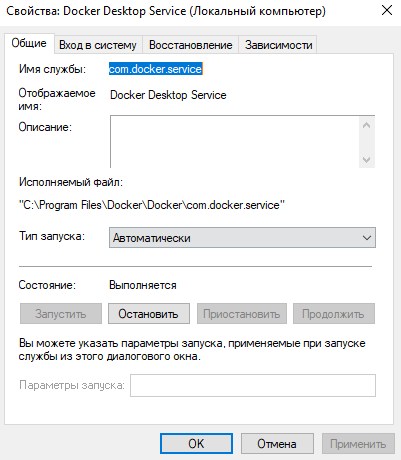


Рис 2.1 - статус сервиса Docker

5. Для проверки списка существующих контейнеров, необходимо ввести команду «docker ps».

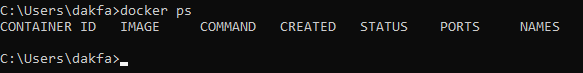


Рис 2.2 - список существующих контейнеров

6. В качестве образа операционной системы для контейнера был выбран образ «alpine».

7. Чтобы получить образ выбранной операционной системы с репозитория DockerHub, требуется ввести команду «docker pull apline».

8. Запустим образ, для этого создаем контейнер Docker из одного слоя –

операционной системы, с помощью команды «docker run alpune».

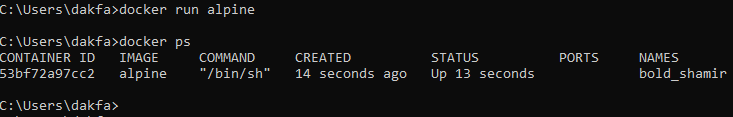


Рис 2.3 - запуск контейнера из одного слоя

9. Далее создадим контейнер из двух слоев. Первый слой – базовый, операционная система. Второй слой скрипт выполнения команды «Hello world», например, /bin/echo 'Hello world'.



Рис 2.4 - создание контейнера из двух слоев

10. Добавление в контейнер прикладного программного обеспечения, например, образ web-сервера nginx, производится с помощью команды «apk add nginx».

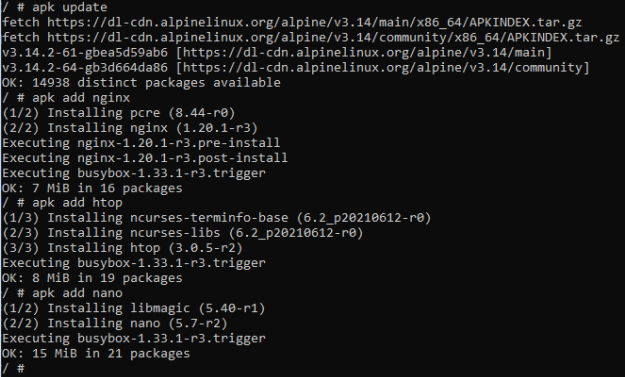


Рис 2.5 - добавление прикладного ПО в контейнер

11. Изучить инструкции DockerFile можно по ссылке https://docs.docker.com/engine/reference/builder/.

12. Создать с использованием инструкций DockerFile новый контейнер, содержащий не менее пяти слоев.

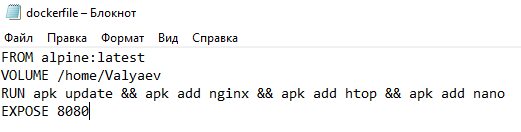


Рис 2.6 - DockerFile для создания контейнера

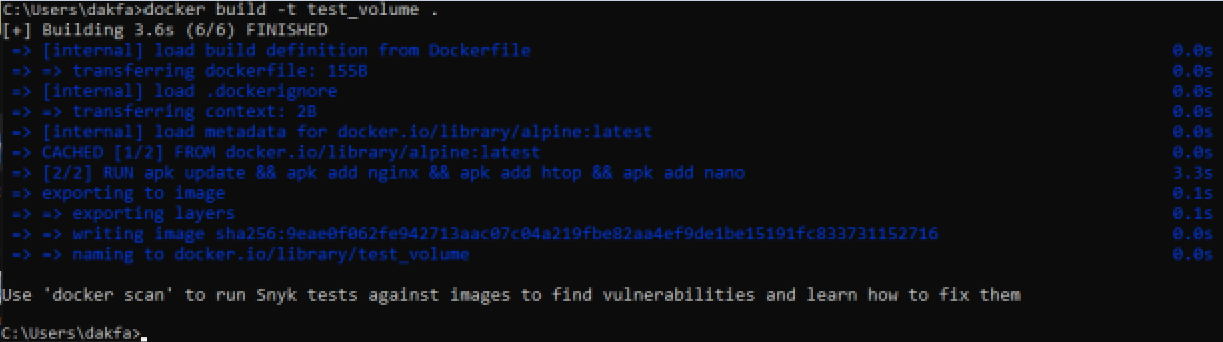


Рис 2.7 - создание образа контейнера с использованием DockerFile



Рис 2.8 - создание контейнера с использованием созданного ранее образа

13. Проверить список существующих контейнеров.

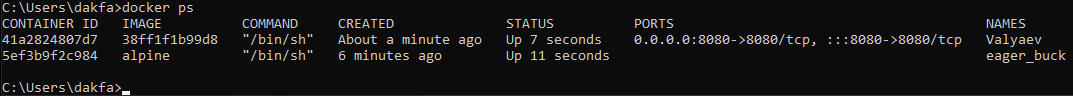


Рис 2.9 - список существующих контейнеров

**Практическая работа №3**

1. В качестве браузера для изучения функционала штатного диагностического инструмента был выбран браузер Opera.
2. Перечень доступных инструментов панели разработчика:

- Elements

- Console

- Sources

- Network

- Toggle device toolbar

- Styles

- Event listeners

- Performance

- Memory

- Application

3. Анализ возможностей инструментов и степень полезности в web-разработке:

Elements – Достаточно полезный инструмент в веб-разработке, позволяет просмотреть/отредактировать разметку страницы;

Console – Незаменимый инструмент в веб-разработке, в консоли выводятся все ошибки и предупреждения, возникающие в процессе работы;

Sources – Полезный инструмент, который позволяет просмотреть все подключенные к странице ресурсы, часто используется для отладки JavaScript используя брейкпоинты;

Network – Незаменимы инструмент в веб-разработке, который отображает все отправляемые запросы, их тип, статус, время обработки, отправляемые/получаемые данные;

Toggle device toolbar – Незаменимый инструмент при разработке адаптивного интерфейса. Данный инструмент позволяет разработчику проверить отображение страницы на различных устройствах;

Styles – Достаточно полезный инструмент в веб-разработке, позволяет просмотреть/отредактировать стили используемые на странице;

Event listeners – Полезный инструмент, позволяющий просмотреть список событий, на которые может реагировать страница;

Performance – Полезный инструмент, позволяющий проверить потребляемые страницей ресурсы компьютера и сети;

Memory – Полезный инструмент, позволяет находить и устранять утечки памяти и отображает распределение памяти среди JavaScript объектов;

Application – Полезный инструмент в веб-разработке, который позволяет просмотреть/очистить загруженные ресурсы, такие как: изображения, шрифты и файлы cookie.

**Практическая работа №4**

1. Для начала работы необходимо скачать и установить XAMPP по ссылке https://www.apachefriends.org/ru/download.html.

2. Далее требуется выбрать необходимые для работы элементы

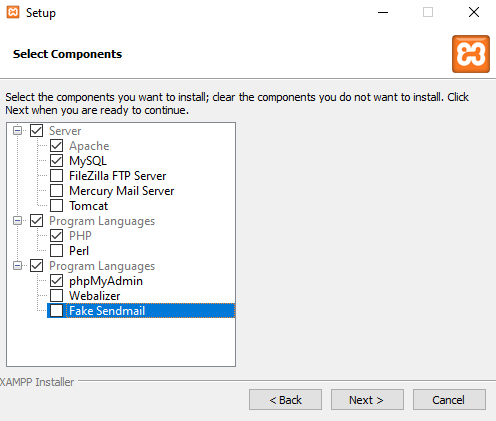


Рис 4.1 – установка необходимых компонентов

3. Запуск компонентов веб-сервера производится через XAMPP Control Panel

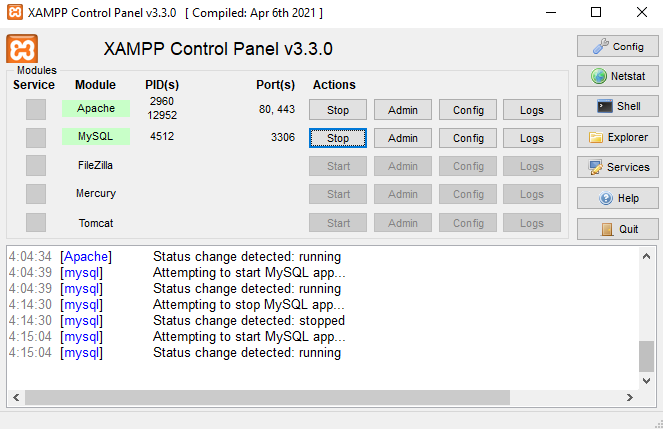


Рис 4.2 – запуск компонентов веб-сервера

4. Для проверки работы веб-сервера, переходим по адресу «http://localhost/»



Рис 4.3 – успешный запуск веб-сервера

5. В качестве корневой директории было решено использовать «test/», в директории был создан файл «index.php» с выводом тестового сообщения

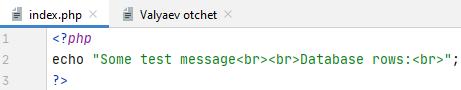


Рис 4.4 – код файла «index.php»

6. Для проверки работы веб-сервера, переходим по адресу «http://localhost/test»

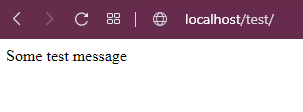


Рис 4.5 – вывод тестового сообщения

7. С использованием phpMyAdmin создана база данных и таблица в двумя столбцами. Первый столбец: тип данных - integer, индекс – primary, auto\_increment. Второй столбец типа text

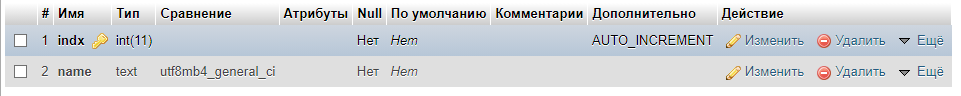


Рис 4.6 – поля тестовой таблицы

8. Далее необходимо наполнить базу данных тестовыми записями

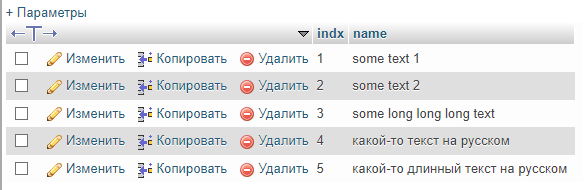


Рис 4.7 – тестовые записи

9. Заключительным шагом необходимо добавить в «index.php» код вывода данных из таблицы.

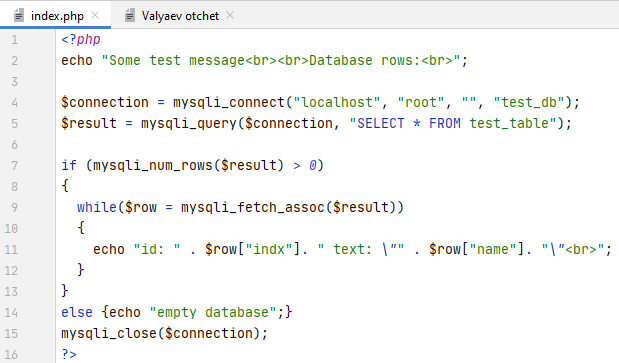


Рис 4.8 – код «index.php»



Рис 4.9 – отображение записей из базы данных

**Практическая работа №5**

1. В качестве скриптового языка был выбран Python и база данных PostgreSQL.
2. В качестве библиотеки для взаимодействия с базой данных используется psycopg2.
3. Реализация функции подключения к базе данных, закрытия соединения, создания базы данных, создания таблиц базы данных, наполнения таблиц баз данных, вывод с объединением данных в таблицах.

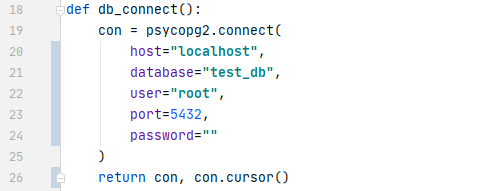


Рис 5.1 – функция подключения к базе данных

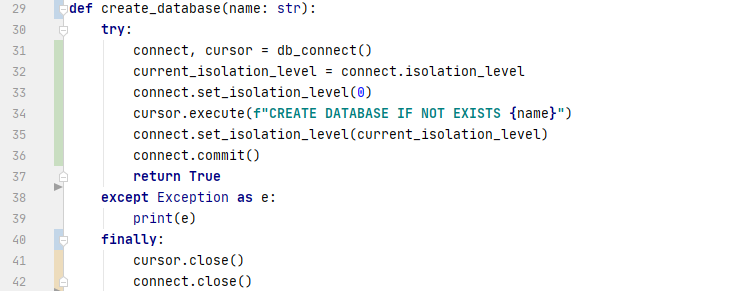


Рис 5.2 – функция создания базы данных

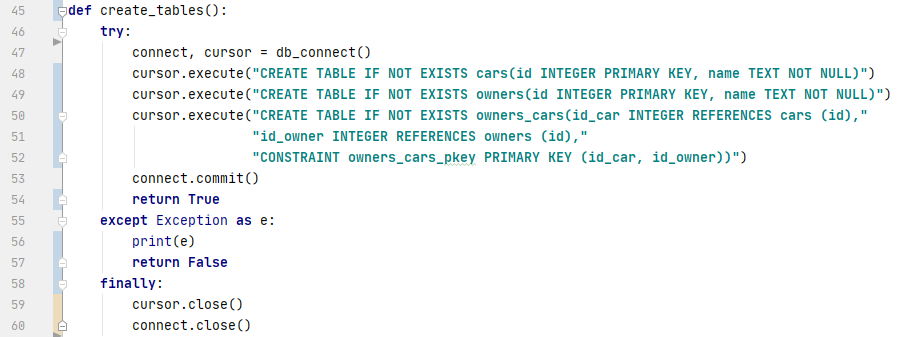


Рис 5.3 – функция создания таблиц в базе данных

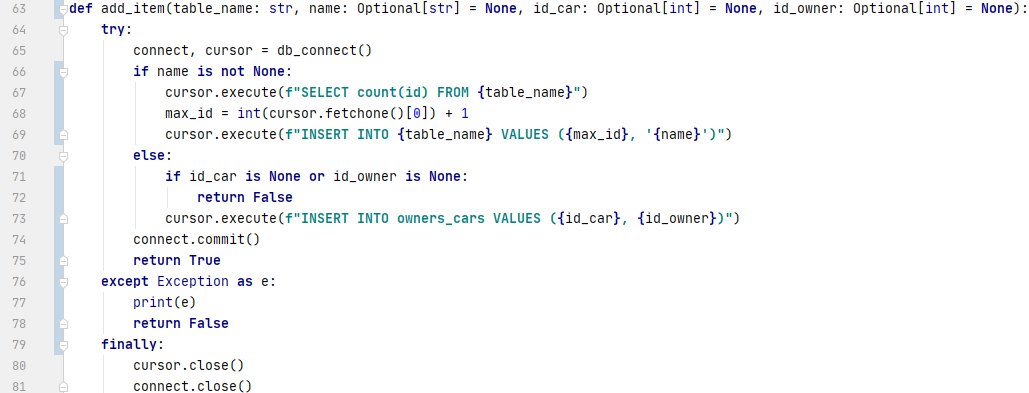


Рис 5.4 – функция добавления записи в базу данных

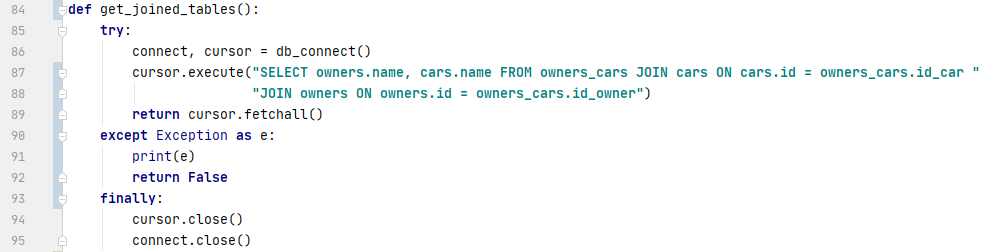


Рис 5.5 – функция вывода объединенных таблиц

1. Далее необходимо создать две таблицы со связью «многие ко многим»

- Название таблиц: «cars» и «owners», обе таблицы содержать по два поля:

«id» типа INTEGER, с первичным ключом и «name» типа TEXT с ограничением NOT NULL. А также таблицу-справочник для связи «один ко многим»

- Название таблицы «owners\_cars», таблица содержит ограничение на уникальность записи и два поля: «id\_car» и «id\_owner», оба поля имеют тип INTEGER со ссылкой на поле «id» в таблицах «cars» и «owners» соответственно.

1. Заполнение таблиц базы данных

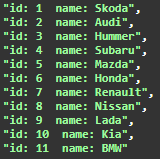


Рис 5.6 – записи в таблице «cars»

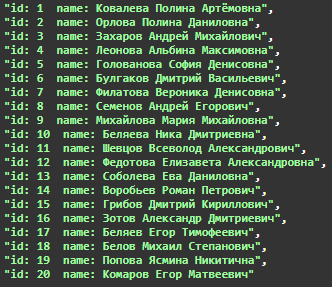


Рис 5.7 – записи в таблице «owners»

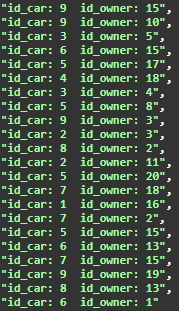


Рис 5.8 – записи в таблице «owners\_cars»

1. Вывод данных объединенных по трем таблицам



Рис 5.9 – вывод объединенных таблиц

1. Схема базы данных

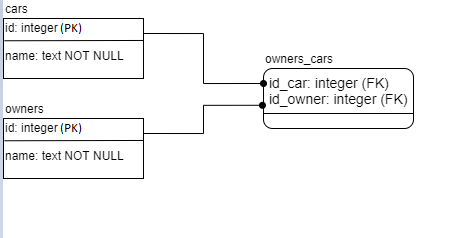


Рис 5.10 – схема базы данных

**Практическая работа №6**

1. В данной практической работе был использован cURL, предустановленный в git bash.
2. Следующим шагом отправим GET-запрос на сайт «sql-ex.ru» с помощью команды «curl https://sql-ex.ru», результатом выполнения команды будет исходный код страницы, выведенный в консоль.
3. Сохраняет содержание сайта в файл, для этого нам понадобится флаг «-o», выполним команду «curl https://sql-ex.ru -o get.txt», результатом выполнения команды будет появление в директории файла с названием «get.txt».

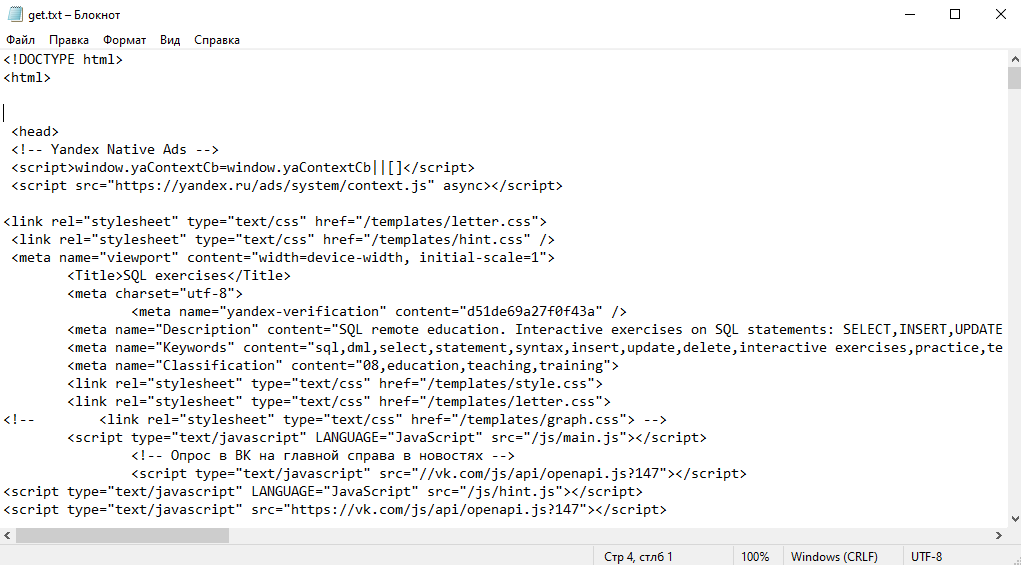


Рис 6.1 – фрагмент файла

1. Выбираем прокси сервер на сайте «https://hidemy.name/ru/proxy-list/» в нашем случае это будет «51.79.248.149:8080».
2. Отправим GET-запрос на сайт «sql-ex.ru» с использованием прокси сервера, для этого необходимо добавить флаг «--proxy», выполним команду «curl --proxy 51.79.248.149:8080 https://sql-ex.ru», результатом выполнения команды будет исходный код страницы, выведенный в консоль.
3. Для отправки запросов с использованием эмуляции User Agent, необходимо добавить флаг «--user-agent».

- User Agent десктопного браузера: Mozilla/5.0 (X11; FreeBSD amd64; rv:68.0) Gecko/20100101 Firefox/90.0

- User Agent мобильного браузера: Mozilla/5.0 (Linux; Android 4.4.2; ASUS\_T00J Build/KVT49L) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Safari/537.36

- User Agent поискового робота: Googlebot-Image/1.0

Последовательно выполним команды с использованием эмуляции User Agent:

curl -L --proxy 51.79.248.149:8080 https://vk.com --user-agent "Mozilla/5.0 (X11; FreeBSD amd64; rv:68.0) Gecko/20100101 Firefox/90.0" -o file\_pc.txt

curl -L --proxy 51.79.248.149:8080 https://vk.com --user-agent "Mozilla/5.0 (Linux; Android 4.4.2; ASUS\_T00J Build/KVT49L) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Safari/537.36" -o file\_mobile.txt

curl -L --proxy 51.79.248.149:8080 https://vk.com --user-agent "Googlebot-Image/1.0" -o file\_bot.txt (search bot Googlebot Image)

Результатом выполнения команд будет создание в директории трех файлов, с названиями: «file\_pc.txt», «file\_mobile.txt», «file\_bot.txt»

Исходный код страницы, полученной с эмуляцией User Agent поискового робота очень похож на код, полученный с десктопного браузера, но отсутствует фрагмент с полями для ввода данных регистрации.

Также можно заметить, что код страницы, полученной с использованием User Agent мобильного браузера имеет меньший размер и полностью отличается от страниц, полученных с использованием других User Agent

1. При переходе на адрес «https:/facebook.com» открывается страница для ввода данных авторизации, а при отправке GET-запроса по тому же адресу, командой «curl https://facebook.com» возвращается пустой ответ, чтобы выяснить причину такого ответа проверим заголовки командой «curl –i https://facebook.com»

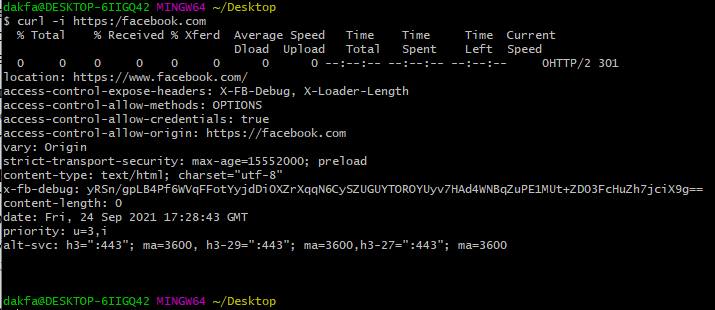


Рис 6.2 – заголовки GET-ответа

На рисунке 6.2 можно увидеть код ответа «301», что говорит о перемещении ресурса, следовательно, для перехода на «https:/facebook.com» потребуется добавить флаг, включающий перенаправление «-L», выполним команду «curl -L https:/facebook.com», результатом выполнения команды будет вывод кода страницы авторизации в консоль.

1. Для выполнения запроса с авторизацией необходимо узнать название полей формы авторизации сайта «sql-ex.ru» с помощью консоли разработчика в браузере.

- название поля ввода логина: «login»

- название поля ввода пароля: «psw»

Для отправки GET-запроса авторизации выполняем команду «curl 'https://www.sql-ex.ru/?login=\*\*\*\*\*&psw=\*\*\*\*\*'», результатом выполнения команды будет возврат на страницу авторизации, т.к. для авторизации необходимо отправить POST-запрос, выполним команду «curl 'https://www.sql-ex.ru' --data-raw 'login=Valyaev\_Danila&psw=rec%21\*3CVu3Ag7X'», результатом выполнения команды будет переход на авторизованную страницу.

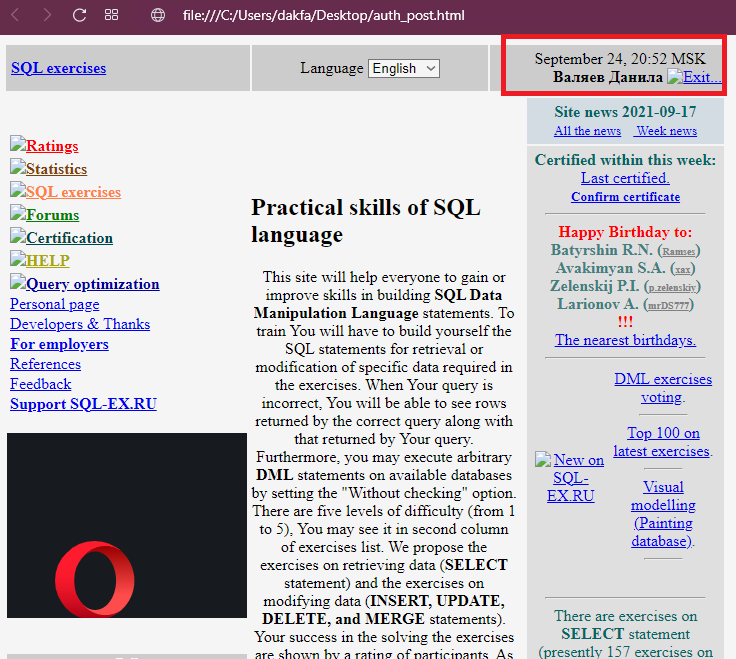


Рис 6.3 – авторизованная страница

**Практическая работа №7**

1. Первым шагом необходимо создать web-страницу index1.php, содержащую форму, состоящую из двух полей ввода и кнопки. По нажатии на кнопку осуществляется переход на вторую web-страницу index2.php

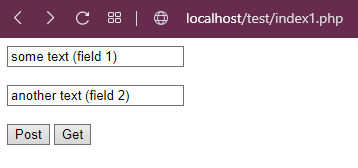


Рис 7.1 – страница «index1.php»

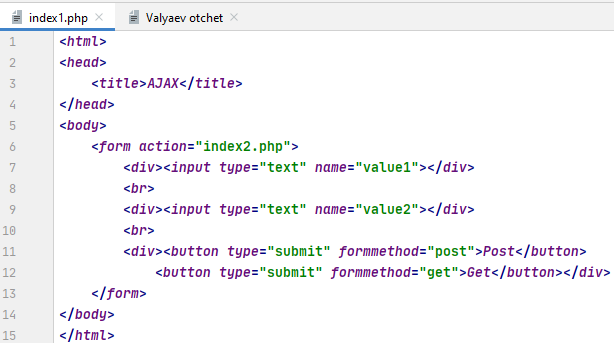


Рис 7.2 – разметка страницы «index1.php»

1. На web-странице index2.php требуется реализовать вывод содержимого первого и второго поля формы ввода index1.php

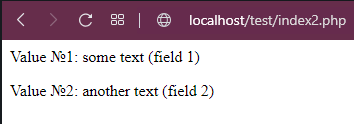


Рис 7.3 – страница «index2.php»

1. Реализуем передачу вводимых значений от страницы index1.php к странице index2.php двумя способами: с помощью методов POST и GET

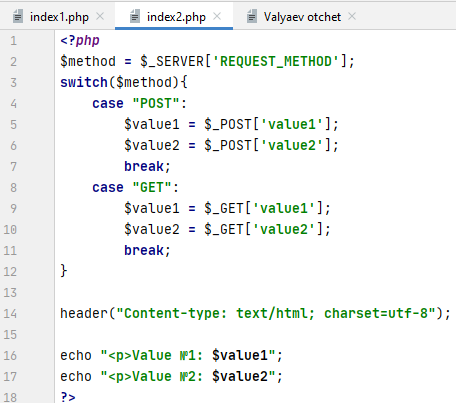


Рис 7.4 – обработка GET и POST запросов

4. Создаем web-страницу index3.php, содержащую однострочное

текстовое полей ввода и многострочное поле вывода;

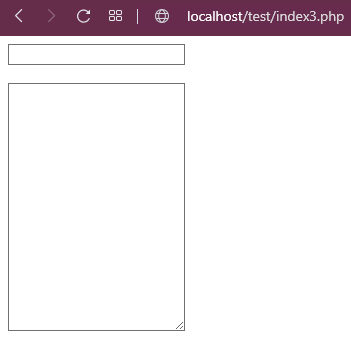


Рис 7.5 – страница «index3.php»

5. Далее необходимо создать «index4.php», который будет осуществлять регистронезависимый поиск по заранее заполненному списку массиву с именами городов.

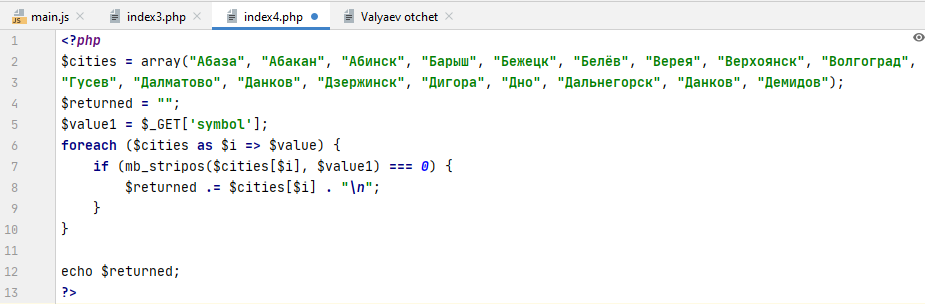


Рис 7.6 – код страницы «index4.php»

7. В поле ввода web-страницы index3.php необходимо вводить

начальные буквы города.

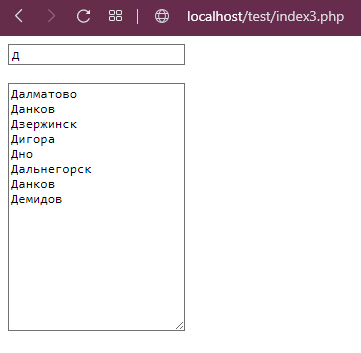


Рис 7.7 – вывод городов, начинающихся на букву «д»

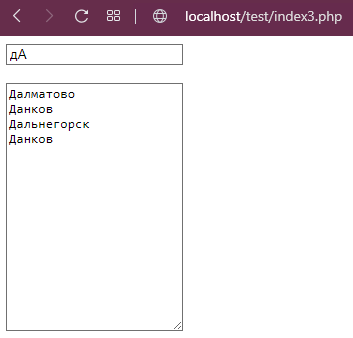


Рис 7.8 – вывод городов, начинающихся на «дА»

8. Взаимодействие между index3.php и index4.php организовается при

помощи технологии AJAX без перезагрузки страницы

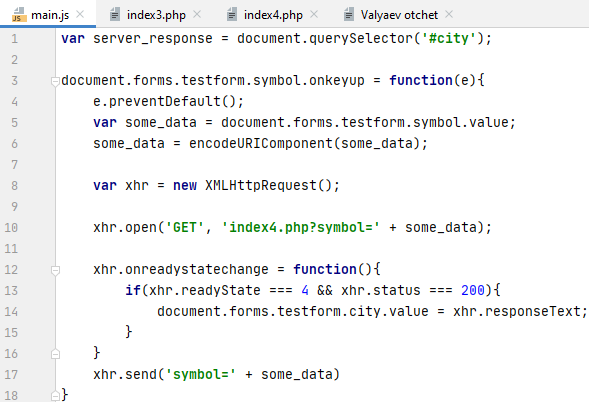


Рис 7.9 – скрипт, реализующий технологию AJAX

**Практическая работа №8**

1. В качестве информационного ресурса для взаимодействия посредством API был выбран WeatherBit.

2. Ознакомиться с документацией на API информационного ресурса можно по ссылке https://www.weatherbit.io/api.

3. Чтобы убедиться в корректности ответов API, необходимы выполнить несколько запросов

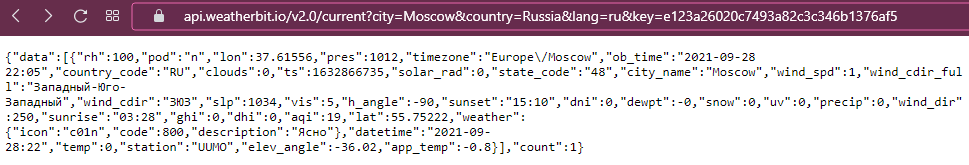


Рис 8.1 – запрос текущей погоду в Москве

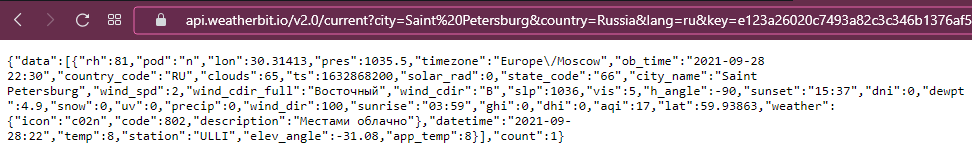


Рис 8.2 – запрос текущей погоды в Санкт Петербурге

1. Формат ответных сообщений:

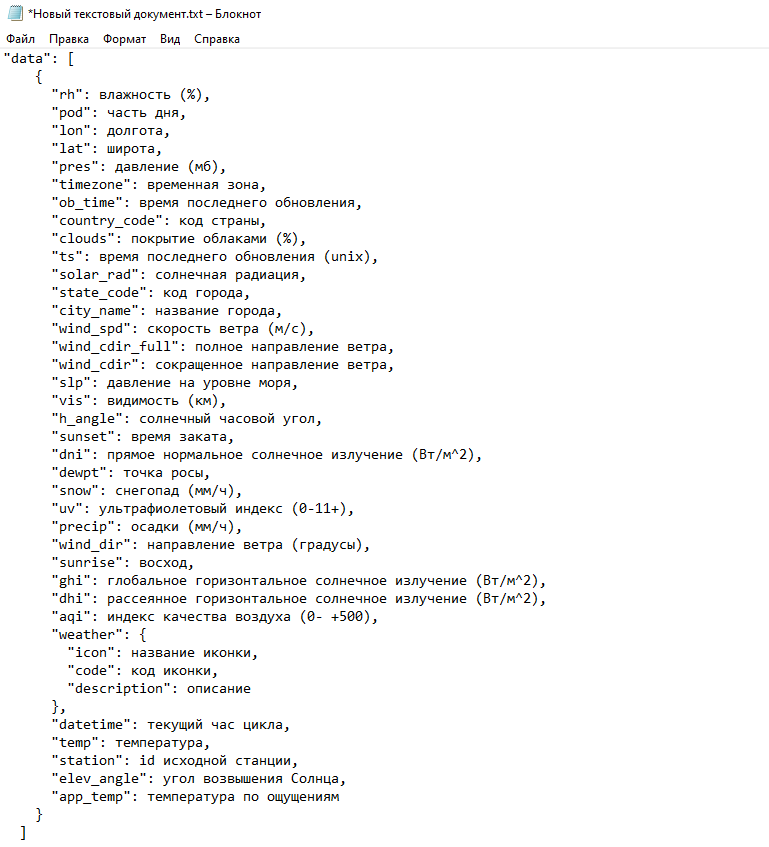


Рис 8.3 – формат ответных сообщений

1. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, была создана страница, с полями ввода страны и города, кнопки для выполнения запроса и полями вывода информации о погоде.

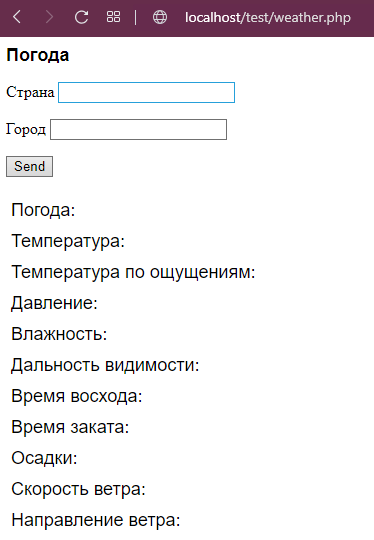


Рис 8.4 – страница для взаимодействия с API

6. При выполнении запросов использовалась технология AJAX для того, чтобы можно было выводит полученную от API информацию без обновления страницы.

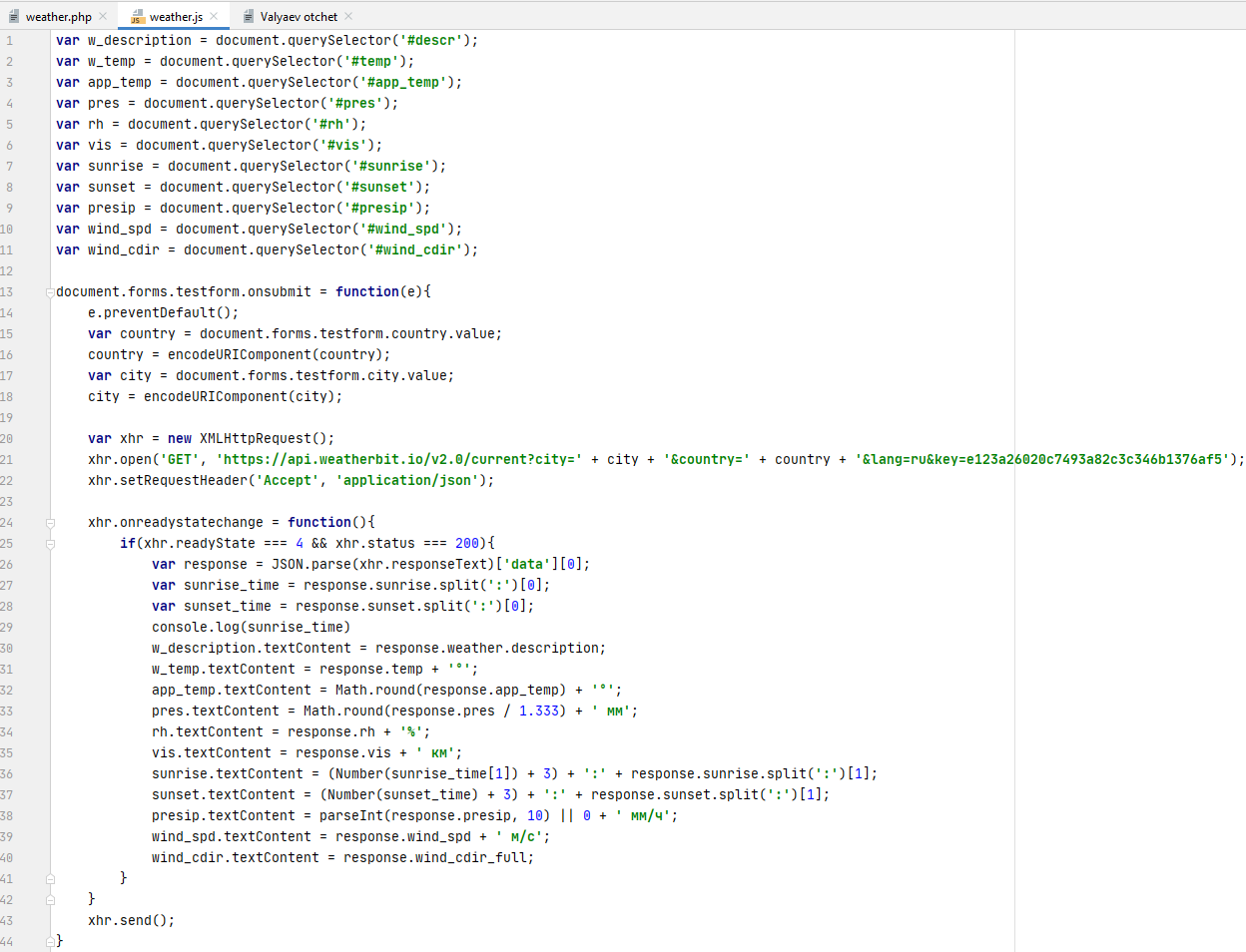


Рис 8.5 – код скрипта, выполняющего запросы

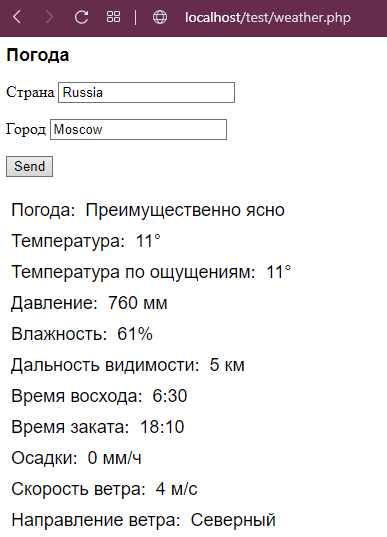


Рис 8.6 – запрос информации о погоде в Москве

**Практическая работа №9**

1. Для начала необходимо создать страницу содержащую поисковую строку для ввода критерия поиска финансовой организации по БИК и SWIFT, поля вывода информации о финансовой организации, элемент управления.

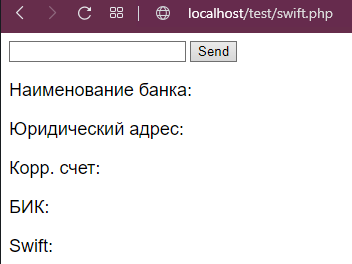


Рис 9.1 – страница «swift.php»

2. В качестве провайдера, предоставляющего доступ через API, позволяющий реализовать функцию поиска финансовой организации был выбран dadata.ru.

3. По нажатии на элемент управления осуществляется отправка HTTPзапроса методом POST в информационный ресурс с использованием API, обработка JSON/XML-ответа и вывод полученных сведений в поле отчета;

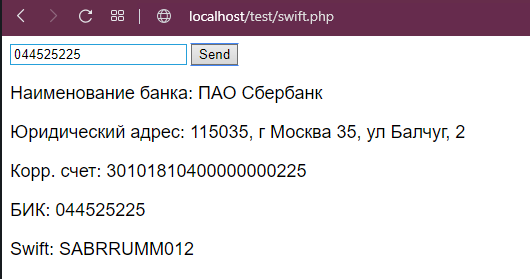


Рис 9.2 – вывод информации о финансовой организации

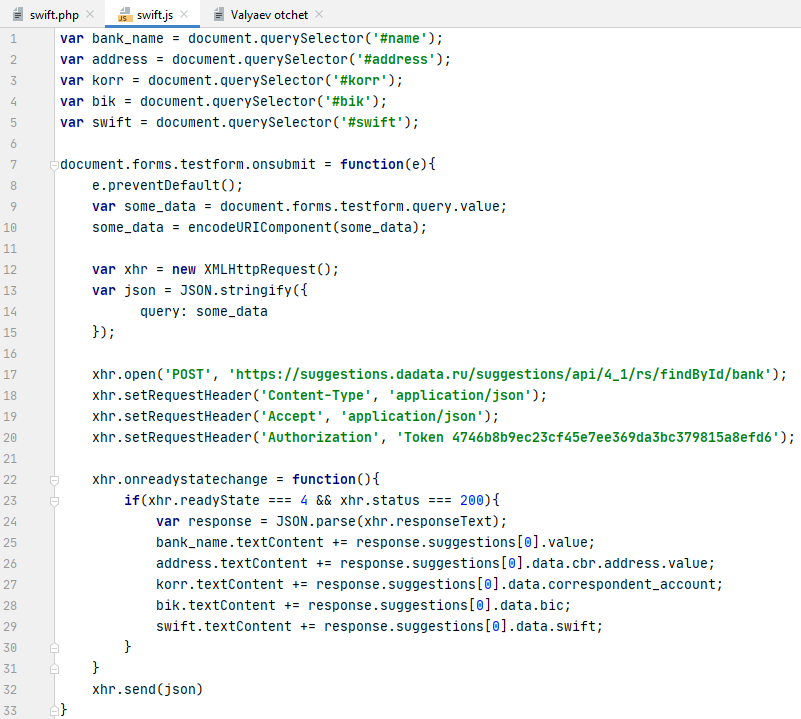


Рис 9.3 – скрипт реализации AJAX и обработка JSON-ответа

**Практическая работа №10**

1. С использованием инфраструктуры, полученной в результате выполнения практического задания №4, была создана страница для поиска географического адреса с полями вывода информации об индексе, кодах ФИАС, КЛАДР.

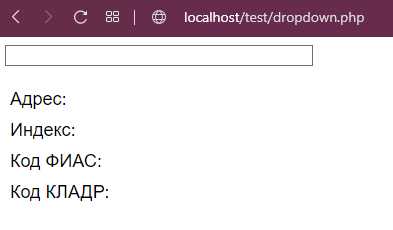


Рис 10.1 – страница поиска адреса

2. Далее реализовать выпадающий список вариантов автоматического дополнения (подсказок) географического адреса без обновления страницы.

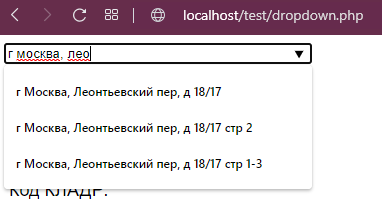


Рис 10.2 – выпадающий список с подсказками

3. В качестве провайдера, предоставляющего доступ через API, позволяющего реализовать функцию поиска адреса в Федеральной информационной адресной системе был выбран dadata.ru.

4. При вводе поискового запроса происходит отправка HTTP-запросов методом POST в информационный ресурс с использованием API, обработка JSON/XML-ответа и вывод полученных сведений.

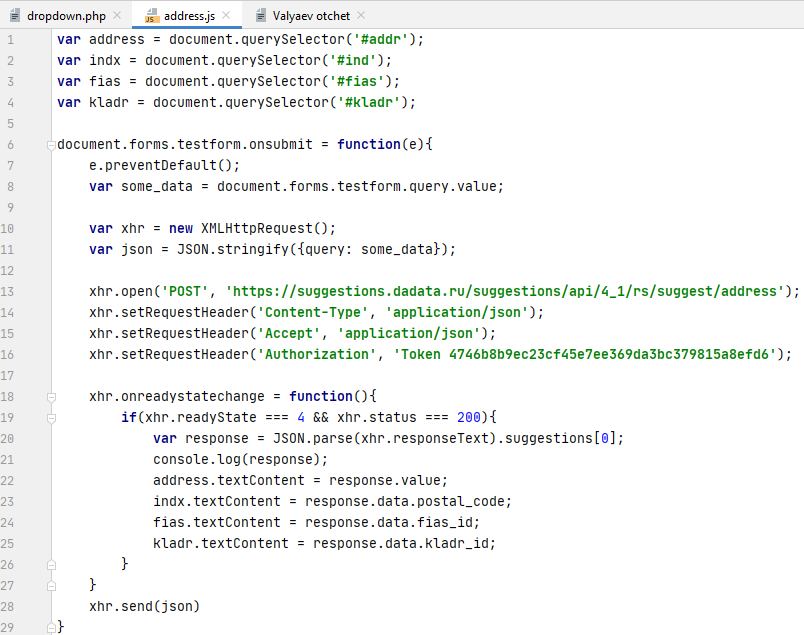


Рис 10.3 – отправка POST-запросов для поиска адресов

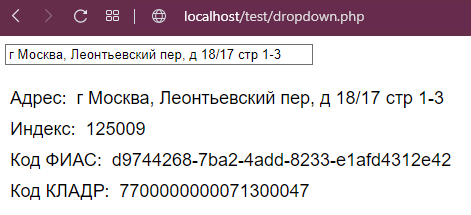


Рис 10.4 – пример поиска информации об адресе

**Практическая работа №11**

1. В качестве новостного провайдера, имеющего RSS-ленту был выбран, https://1prime.ru/export/rss2/index.xml. Тег записи для большинства RSS-лент «item», 1prime не исключение.

2. Поля, которыми задаются свойства новостей:

- title – заголовок новости;

- link – ссылка на новость;

- description – краткое описание новости;

- pubDate – дата публикации новости;

- dc:subject – категории, к которым относится новость;

- tags – теги новости;

- guid– постоянная ссылка на новость;

- enclosure – изображение к новости (ссылка, тип, длинна).

3. Составить ER-диаграмму класса-элемента RSS;

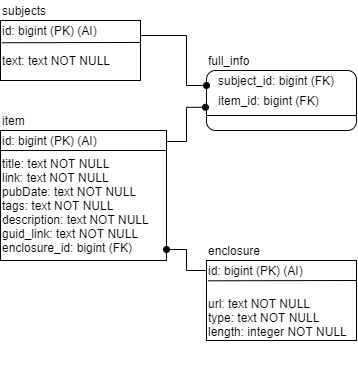


Рис 11.1 – ER-диаграмма класса

4. В качестве языка для парсинга RSS был выбран Python с использованием библиотеки BeautifulSoup и библиотеки psycopg2 для работы с базой данных postgresql.

5. Загружаем RSS-описание с использованием библиотек requests и beautifulsoup для последующей структуризации данных.



Рис 11.2 – загрузка RSS-описания

6. Структурируем данные RSS-ленты и выводим первую новость длдя проверки правильности работы.

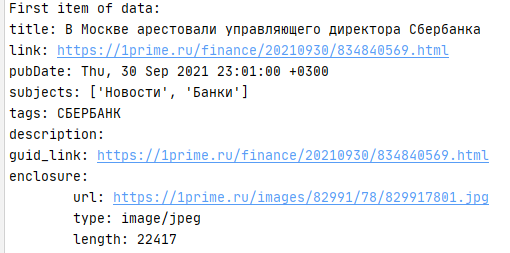


Рис 11.3 – вывод первой новости

7. Далее необходимо создать класс для хранения записей из ленты с методом вывода всех полей класса в строку.

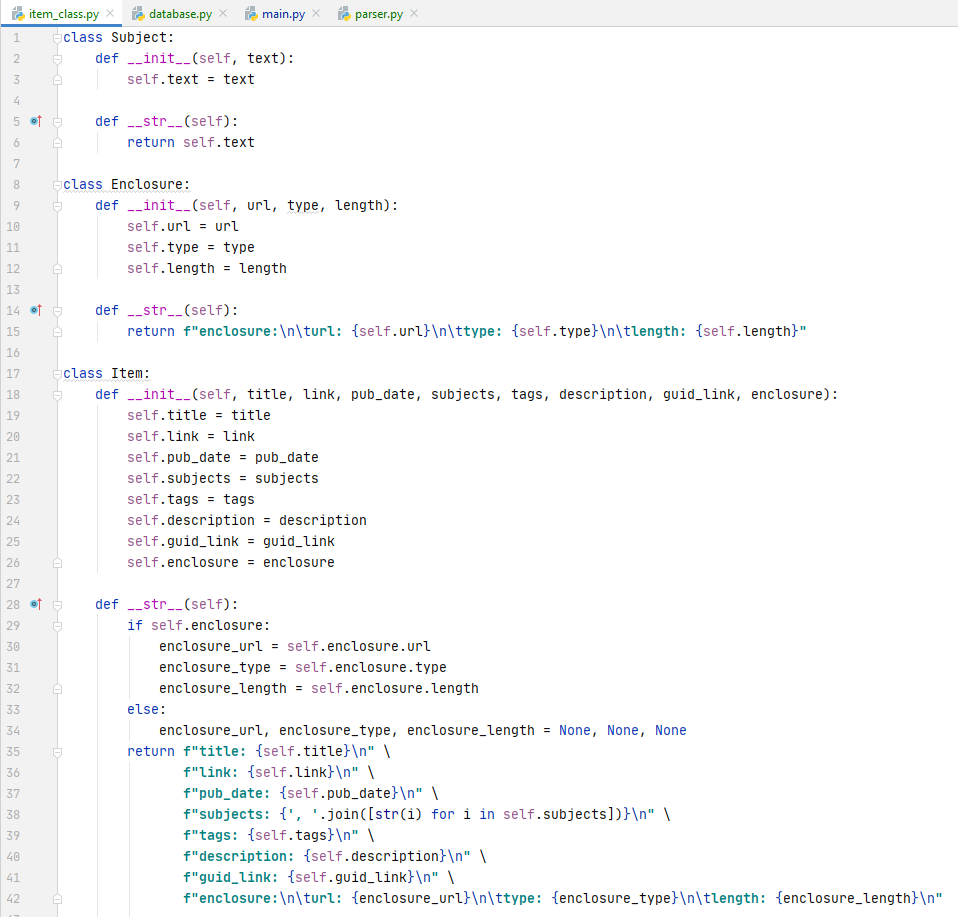


Рис 11.4 – реализация класса для хранения записей

8. Далее реализовываем добавление экземпляров класса в список для последующего вывода.

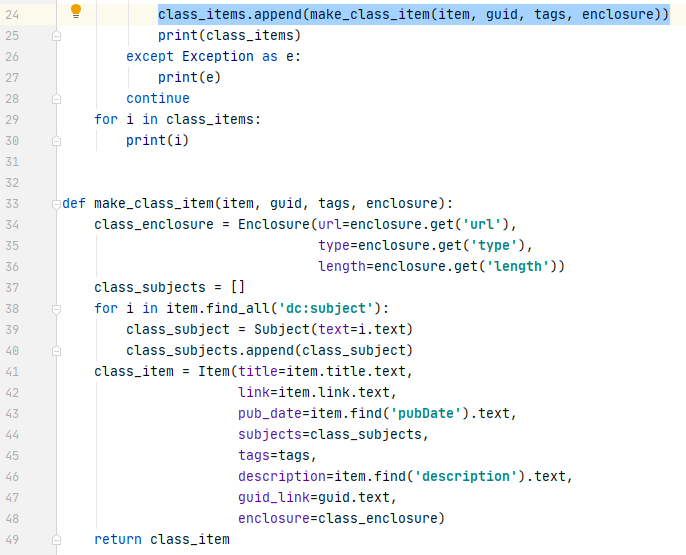


Рис 11.5 – создание экземпляра и добавление его в список

9. Далее реализовываем метод загрузки в базу данных и вызываем его во время парсинга.

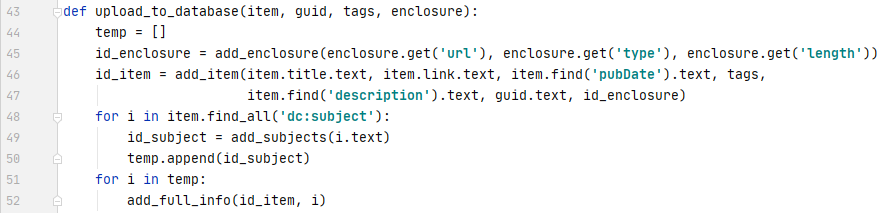


Рис 11.6 – метод загрузки в базу данных