*Лабораторная работа № 5*

*Разработка веб-приложения*

***Цель работы:*** разработать веб-приложение для представления пользователю информации, хранящейся в базе данных (БД).

***Задание:***

1) Ознакомиться с кодом и структурой веб-приложения (см. **пример приложения Python Flask в папке pikpo5_python_flask**). Для выполнения лабораторной работы в качестве альтернативного веб-фреймворка можно также использовать платформу NodeJS (см. **пример приложения NodeJS Express в папке pikpo5_nodejs_express**).

2) Реализовать методы (запросы) для выборки необходимых данных из БД для web-приложения.

3) Используя встроенный шаблонизатор веб-страниц, реализовать вставку данных в ваши html-страницы (в соответствии с дизайном веб-страниц, созданных в ходе выполнения лабораторной работы №4).

4) Реализовать в обработчике данных постоянное сканирование заданной папки на наличие новых файлов (источников данных) и их автоматическую обработку.

**Для успешной защиты лабораторной работы студенты должны предоставить проект (демонстрацию приложения) и отчет к нему, содержащий скриншоты основных контентных страниц веб-приложения, листинг шаблонов html-страниц, а также листинг используемых методов для работы с БД.**

Требования к оформлению отчета:

Способ выполнения текста должен быть единым для всей работы. **Шрифт** – **Times New Roman**, кегль 14, **межстрочный интервал** – 1,5, **размеры полей**: левое – 30 мм; правое – 10 мм, верхнее – 20 мм; нижнее – 20 мм. Сокращения слов в тексте допускаются только общепринятые.

**Абзацный отступ** (1,25) должен быть одинаковым во всей работе. **Нумерация страниц** основного текста должна быть сквозной. Номер страницы на титульном листе не указывается. Сам номер располагается внизу по центру страницы или справа.

## 1. Разработка веб-приложения

**1.1 Пример проекта веб-приложения (Flask).** Рассмотрим базовую структуру веб-приложения:

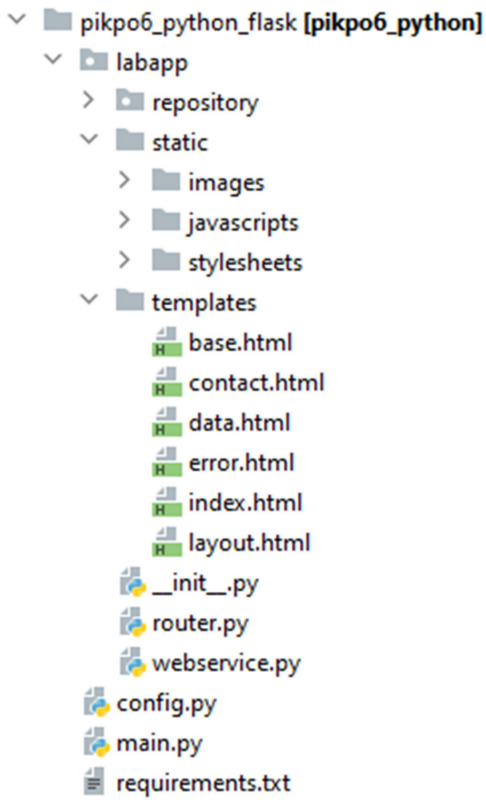


Рис. 1 – Файловая структура проекта веб-приложения на Python (Flask)

Рассмотрим назначение основных фалов и директорий:

**`main.py`** – файл запуска веб-приложения с указанием порта и интерфейса для сетевого обмена. «**host='0.0.0.0', port=8000**» - означает, что ваше приложение будет принимать соединение на всех доступных сетевых интерфейсах, используя порт 8000.

**`config.py`** – модуль конфигурации приложени. Здесь указывается основной параметр **`DB_URL`** – строка подключения к БД, которая должна совпадать со строкой подключения, используемой для обработчика данных.

**`/labapp`** – пакет, который содержит рабочие файлы веб-приложения и объединяет их в отдельный пакет, т.к. в данной директории находится файл **__init__.py** (инициализатор пакета в среде Python). Этот файл также инициализирует веб-приложение Flask.

**`/static`** – внутренняя папка приложения, содержащая статичные ресурсы приложения: изображения (images), стили веб-страниц (stylesheets), JavaScript-файлы и т.п.

**`/templates`** – внутренняя папка приложения, содержащая шаблоны html-страниц, в которые будет делаться вставка переменных («рендеринг»). Данные страницы и их «рендеринг» (заполнение данными) активно используются при реализации классического MVC-приложения.

**repository** – пакет, содержащий методы для подключения и исполнения запросов к БД. Практически полностью повторяет аналогичный пакет в обработчике данных (см. лабораторную работу №4), кроме самих методов (запросов) к БД (файл **sql_api.py**).

**routes.py** – модуль регистрации маршрутов (URLs) для реализации переходов на соответствующие страницы.

**requirements.txt** – файл со списком зависимостей (необходимых библиотек) python-приложения. В дальнейшем, например, при развертывании или переносе проекта на другой ПК все необходимые библиотеки можно будет поставить одной командой:

```
pip install -r requirements.txt
```

Для установки зависимостей в проекте **NodeJS**, перейдите в корень проекта (в директорию, где находится файл **package.json**) и выполните команду в терминале:

```
npm install
```

**1.2 Подключение и обработка маршрутов к веб-страницам.** Для веб-приложения Flask подключение адреса (маршрута) к методу обработки страницы осуществляется следующим образом:

```
@<путь_к_странице>
<объявление вашего метода обработки страницы>
```

Например, переход к индексной странице будет выглядеть следующим образом:

```
@app.route('/', methods=['GET'])
def index():
…
return render_template('index.html', title='HOME')
```

Т.е. декоратор `@app.route()` позволяет назначить исполняемую функцию при обработке заданного адреса и HTTP-метода (более подробно данный механизм обработки HTTP-запросов разобран в рамках курса «Архитектура вычислительных систем и компьютерные сети» в лабораторных работах №4, 5).

Если необходимо в исполняемую функцию передать какой-то параметр (в данном примере идентификатор файла), то данный параметр можно определить как изменяемую часть маршрута. Для этого необходимо объявить переменную и тип переменной в URL и и далее определить соответствующий аргумент исполняемой функции:

```
@app.route('/data/<int:source_file_id>', methods=['GET'])
def get_data(source_file_id: int):
    processed_data = webservice.get_processed_data(source_file_id)
    return render_template('data.html', title='MY_DATA_PAGE')
```

В таком случае станут доступны переходы, например:

`/data/1`

`/data/2`

и т.д.

Также можно передать дополнительный параметр (например, Id) из следующей строки запроса:

```
http://127.0.0.1:8000/contacts?Id=5
```

В таком случае, в исполняемой функции значение данного параметра можно получить при помощи следующей инструкции:

```
paramId = request.args.get('Id')
```

Также в примере приложения реализована обработка POST-запроса для демонстрации подхода AJAX (подробнее см. formsend.js, а также лабораторную работу №5 курса «Архитектура вычислительных систем и компьютерные сети»), который обрабатывает данные с формы на странице **contact.html**:

```
@app.route('/api/contactrequest', methods=['POST'])
def post_contact():
…
```

**1.3 Работа с шаблонизатором веб-приложения Flask.** Фреймворк Flask по умолчанию имеет встроенный шаблонизатор **Jinja2**. Инструкция **render_template**, представленная в примерах кода выше, используется в веб-приложении Flask для вставки («рендеринг») данных на веб-страницу. Для рендеринга шаблона необходимо в методе render_template объявить соответствующий аргумент (например, **title=**) и присвоить ему необходимое значение. Заданный аргумент должен быть также объявлен в html-шаблоне в двойных фигурных скобках, например:

```
<title>{{title}}</title>
```

Помимо обычных переменных в шаблон можно также передавать и массивы, в том числе и многомерные. Например, в коде, как один из параметров, в шаблон передается двумерный массив processed_files следующего вида:

```
[(1, 'seeds_dataset.csv', '2022-10-25 02:11:47'),
(2, 'seeds_dataset.csv', '2022-10-25 02:12:01')]
```

Вставка данного массива в шаблон возможна с помощью следующей циклической конструкции шаблонизатора:

```
{% for row in processed_files %}
    <tr>
        <td>
            <a href="/data/{{row[0]}}">{{row[1]}}</a>
        </td>
        <td>
            {{row[0]}}
        </td>
        <td>
            {{row[2]}}
        </td>
    </tr>
{% endfor %}
```

Шаблоны также могут включаться друг в друга. Место вставки содержимого из другого шаблона обозначается конструкцией (например, в базовом шаблоне **layout.html**):

```
{% block content %}
{% endblock %}
```

При этом во вставляемом шаблоне должен быть указан шаблон для вставки (с помощью инструкции **extends**) и содержимое, которое будет вставлено:

```
{% extends "layout.html" %}
    {% block content %}
        <содержимое этого блока будет вставлено в layout.html >
    {% endblock %}
```

**Дополнительная информация по шаблонизатору Flask:**
https://habr.com/ru/post/193260/