***Laboratory work # 5***

***Web Application Development***

***Purpose of the work:*** develop a web application for presenting information stored in a database (DB) to the user.

***Task:***

1) Get acquainted with the code and structure of the web application (see **sample app Python Flask in the pikpo5\_python\_flask folder**). To perform lab work, you can also use the platform as an alternative web framework NodeJS (see **sample app NodeJS Express in the pikpo5 folder\_nodejs\_express**).

2) Implement methods (queries) for fetching the necessary data from the database for web-applications.

3) Using the built-in web page template engine, implement data insertion into your own templates. html-pages (in accordance with the design of web pages created during laboratory work #4).

4) Implement constant scanning of the specified folder for new files (data sources) in the data handler and their automatic processing.

**To successfully defend their lab work, students must submit a project (demo of the application) and a report to it containing screenshots of the main content pages of the web application, a listing of templates html-pages, as well as a listing of the methods used for working with the database.**

Report design requirements:

The text execution method should be the same for all work. **Font** –

**Times New Roman**,size14,**line spacing**–1,5,**field sizes**:left–30 mm; right – 10 mm, top-20 mm; bottom-20 mm. Abbreviations of words in the text are allowed only in general terms.

**Paragraph indent**(1,25)it should be the same in all work.**Page numbering**the main text should be end-to-end.Page number on the title pagethe list is not specified. The number itself is located at the bottom center of the page or on the right.

***Methodological recommendations for performing laboratory work***

**1. Web application development**

**1.1 Example of a web application project (Flask).**Let's look at the basic structure of a web application:

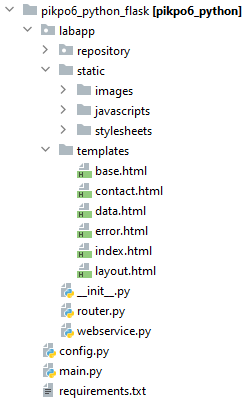


Fig. 1-File structure of the web application project on Python (Flask)

Consider the purpose of the main files and directories:

**main.py** – a web application startup file that specifies the port and interface for network communication. «**host='0.0.0.0', port=8000**"- means that your application will accept a connection on all available network interfaces using port 8000.

**config.py** – application configuration module. The main parameter is specified here **DB\_URL** – the DB connection string, which must match the connection string used for the data handler.

**/labapp** - a package that contains the working files of the web application and combines them into a separate package, since the file is located in this directory. **\_\_init\_\_.py** (package initializer in the environment Python). This file also initializes the web application Flask.

**/static** – internal folder of the app that contains static app resources: images (images), web page styles (stylesheets), JavaScript- files, etc.

**/templates** – internal folder of the application containing templates html-pages where variables will be inserted ("rendering"). Page data and their "rendering" (filling in with data) are actively used in the implementation of the classic approach. MVC-applications.

**repository** - a package containing methods for connecting and executing queries to the database. Almost completely repeats the same package in the data handler (see lab # 4), except for the methods (queries)themselves to the database (file **sql\_api.py**).

**routes.py** - route registration module (URLs) to implement traffic to the corresponding pages.

**requirements.txt** - file with a list of dependencies (required libraries) python-applications. In the future, for example, when deploying or transferring a project to another PC, all the necessary libraries can be installed with one command:

**pip install -r requirements.txt**

To install dependencies in a project **NodeJS**, go to the root of the project (in the directory where the file is located **package.json**) and run the command in the terminal:

**npm install**

**1.2 Connecting and processing routes to web pages.** For a web application Flask connecting the address (route) to the page processing method is done as follows::

**@<page\_path>**

<declaring your page processing method>

For example, the transition to the index page will look like this:

**@app.route('/', methods=['GET'])**

**def index():**

…

return **render\_template**('index.html', title='HOME')

I.e. the decorator **@app.route()**allows you to assign an executable function when processing a given address and HTTP-method (this processing mechanism is described in more detail HTTP- queries are parsed as part of the course "Architecture of computer systems and computer networks" in laboratory works No. 4, 5).

If you need to pass a parameter to the executable function (in this example, the file ID), then this parameter can be defined as the modified part of the route. To do this, you must declare the variable and the type of the variable in URL and and further define the corresponding argument of the executable function:

@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')

In this case, click-throughs will become available, for example:

/data/**1**

/data/**2**

and t.d.

You can also pass an additional parameter (for example, Id) from the following query string:

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

In this case, in the executable function, the value of this parameter can be obtained using the following statement:

paramId = **request.args.get('Id')**

The sample application also implements POST request processing to demonstrate the AJAX approach (for more information, see formsend.js, and laboratory work No. 5 of the course "Architecture of computer systems and computer networks"), which processes data from the form on the page **contact.html**:

**@app.route('/api/contactrequest', methods=['POST'])**

**def post\_contact():**

…

**1.3 Working with the Web application template engine Flask.**Framework Flask by default, it has a built-in template engine **Jinja2**. Instruction manual **render\_template** the information provided in the code examples above is used in the web application Flask to insert ("render") data on a web page. To render a template, you must declare the corresponding argument in the render\_template method (for example, **title=**) and assign the required value to it. The specified argument must also be declared in html- a template with double curly brackets, for example:

<title>**{{title}}**</title>

In addition to regular variables, you can also pass arrays to the template, including multidimensional ones. For example, in the code, a two-dimensional processed\_files array of the following format is passed to the template as one of the parameters:

**[(1, 'seeds\_dataset.csv', '2022-10-25 02:11:47'),**

**(2, 'seeds\_dataset.csv', '2022-10-25 02:12:01')]**

Inserting this array into a template is possible using the following circular template engine construction:

**{% 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 %}**

Templates can also be included in each other. The place where content from another template is inserted is indicated by a construction (for example, in the base template **layout.html**):

**{% block content %}**

**{% endblock %}**

At the same time, the template to insert must specify the template to insert (using the instructions). **extends**) and the content to be inserted:

**{% extends "layout.html" %}**

{% block content %}

<the contents of this block will be inserted in **layout.html** >

{% endblock %}

**Additional information about the template engine Flask:**

<https://habr.com/ru/post/193260/>