

Создание клиентского приложения при помощи Flask

Глоссарий

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Werkzeug — набор инструментов WSGI, стандартного интерфейса Python для развертывания веб-приложений и взаимодействия между ними и различными серверами разработки.

Jinja — это шаблонизатор для языка программирования Python. Он подобен шаблонизатору Django, но предоставляет Python-подобные выражения, обеспечивая исполнение шаблонов в песочнице. Это текстовый шаблонизатор, поэтому он может быть использован для создания любого вида разметки, а также исходного кода.

Ход работы

1. Установите интерпретатор Python (<https://www.python.org/downloads/>). Для написания данной методички использовался Python 3.7.2
2. Python включает в себя систему управления пакетами pip. При помощи этой системы установите библиотеку Flask, используя команду:
pip install flask
3. Также для создания HTML-форм потребуется пакет WTForms. Установите его при помощи команды:
pip install flask-wtf
4. Чтобы сделать процесс написания кода более удобным, скачайте IDE PyCharm (<https://www.jetbrains.com/pycharm/>) для Python, которое поддерживает создание шаблона Flask-приложения.
5. В PyCharm создайте проект Flask. В качестве названия проекта используйте **deanery-ivbo-02-15-«номер бригады»**.

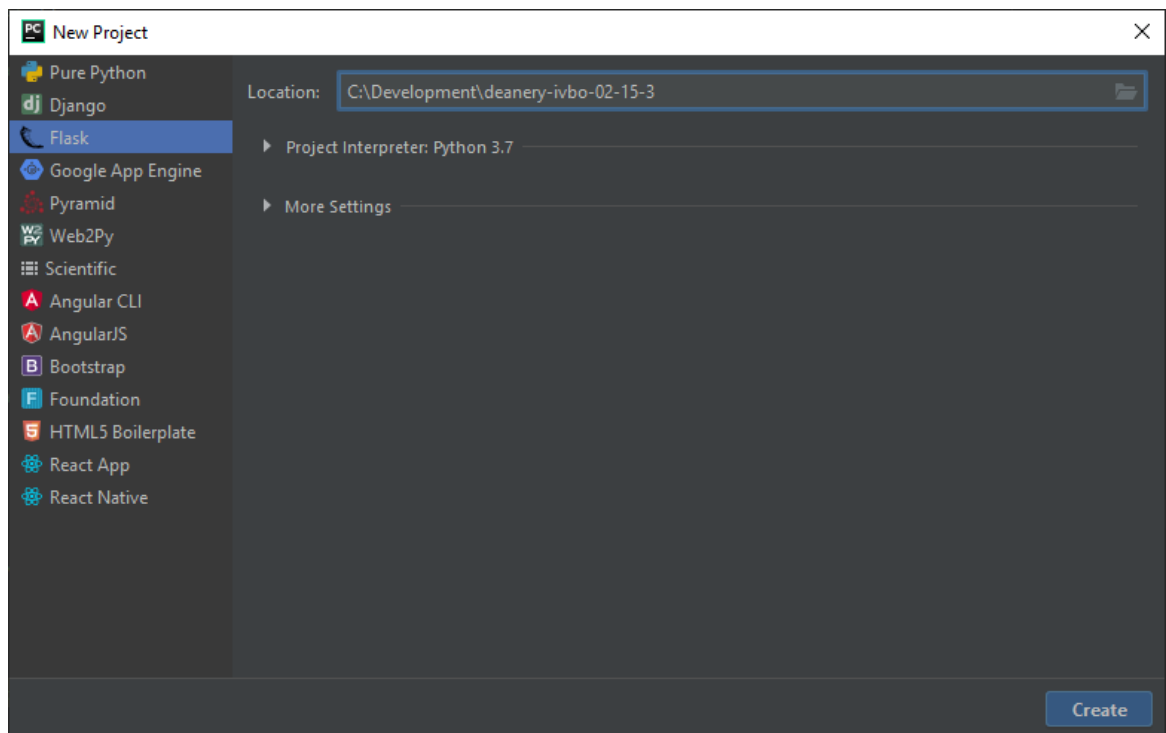


Рис.1 Создание проекта

Для данного проекта необходима виртуальная среда, поэтому необходимо убедиться, что она будет создана в проекте. Для этого разверните раздел Project Interpreter и выберите вариант New environment using Virtualenv.

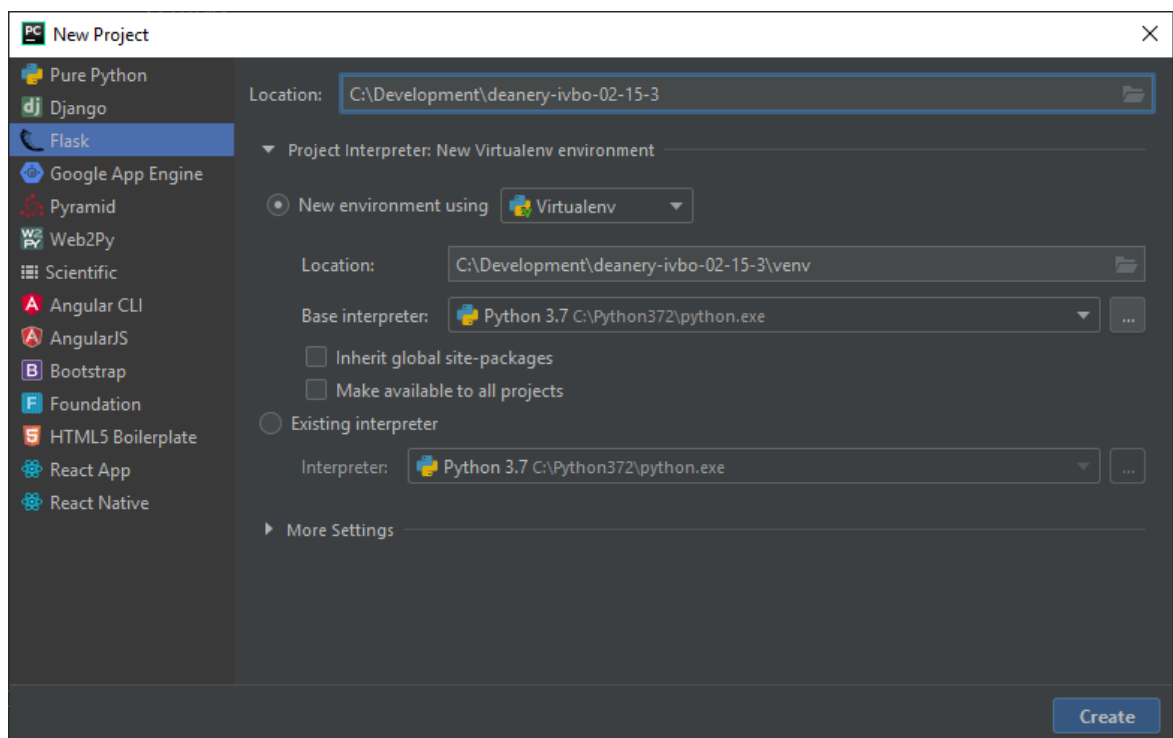
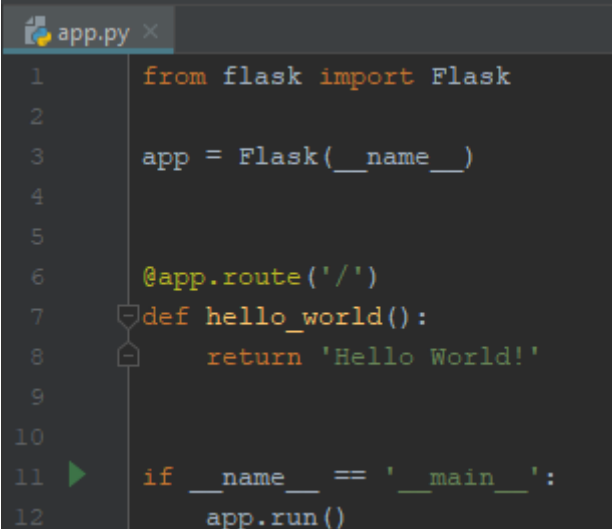


Рис.2 Настройка проекта

После создания получили стандартный шаблон приложения.



```

1  from flask import Flask
2
3  app = Flask(__name__)
4
5
6  @app.route('/')
7  def hello_world():
8      return 'Hello World!'
9
10
11 if __name__ == '__main__':
12     app.run()

```

Рис.3 Созданное приложение

Запустите приложение и перейдите по адресу 127.0.0.1:5000. Вы должны увидеть страницу с текстом Hello World!.

6. Подключите набор инструментов Bootstrap для стилизации приложения. Для этого откройте терминал в нижней левой части окна PyCharm и введите следующую команду:

pip install flask-bootstrap

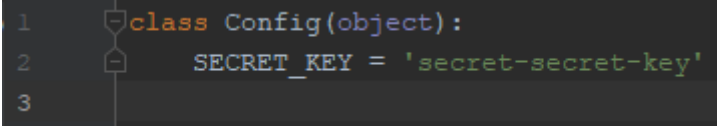
7. Создадим более правильную структуру проекта. Для этого сначала в корне проекта создайте пакет Python с именем app и перенесите туда каталоги static и templates.

После этого создайте в каталоге app создайте модуль routes.py - для хранения функций переходов по всем URL сайта.

Также в корне проекта создайте модуль config.py, он будет хранить необходимую информацию о конфигурации проекта. И переименуйте app.py в run.py.

Приступим к разбиению проекта на модули.

В модуле config.py создайте класс конфигурации приложения.



```

1  class Config(object):
2      SECRET_KEY = 'secret-secret-key'
3

```

Рис.4 Код модуля config.py

В модуль __init__.py перенесите создание приложения. Здесь создается экземпляр Flask приложения, конфигурируется, а также инициализируется расширение Flask-Bootstrap.

```

1  from flask import Flask
2  from config import Config
3  from flask_bootstrap import Bootstrap
4
5  app = Flask(__name__)
6  app.config.from_object(Config)
7  bootstrap = Bootstrap(app)
8
9  from app import routes
10

```

Рис.5 Код модуля app/__init__.py

В модуль routes.py перенесите код с функцией обработки перехода в корень сайта.

```

1  from app import app
2
3
4  @app.route('/')
5  def hello_world():
6      return 'Hello World!'
7

```

Рис.6 Код модуля app/routes.py

Переименуйте app.py в run.py и перенесите туда код запуска приложения.

```

1  from app import app
2
3  if __name__ == '__main__':
4      app.run()
5

```

Рис.7 Код модуля run.py

8. Пора приступить к созданию первой страницы.

Для начала создадим основную страницу, в которой будет происходить рендер всего содержимого. В каталоге templates создадим файл base.html. Вот код, который необходимо в него поместить.

```

{% extends 'bootstrap/base.html' %}

{% block title %}
    Flask app
{% endblock %}

{% block content %}
    <h1>Hello Flask!</h1>
{% endblock %}

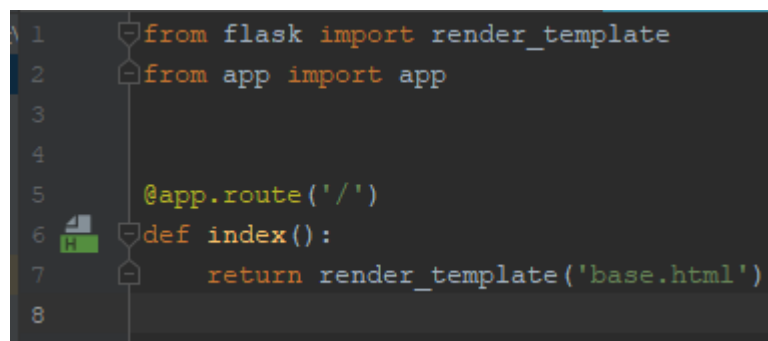
```

Поскольку используется Flask-Bootstrap, страница изначально будет рендериться самим Bootstrap, а добавляемая нами страница будет содержать блоки, добавляемые в Bootstrap верстку. Строка `{% extends 'bootstrap/base.html' %}` указывает Jinja, что данный html файл будет использован в качестве расширения для шаблонной страницы Bootstrap.

Блок `title` содержит текст, который будет отображаться в тэге `title`.

Блок `content` - это основное содержимое страницы. Для примера создадим в нём заголовок `h1` с текстом `Hello Flask!`.

Теперь изменим функцию обработки перехода на главную страницу сайта. По умолчанию она возвращает строку. Изменим это импортировав из библиотеки `flask` функцию `render_template`, которая отвечает за рендер страниц, и изменим функцию под рендера только что созданной страницы.



```
1 from flask import render_template
2 from app import app
3
4
5 @app.route('/')
6 def index():
7     return render_template('base.html')
8
```


Рис.8 Измененный модуль `routes.py`

Запустите приложение и убедитесь, что на ней появляется текст `Hello Flask!` в тэге `h1`.

9. В предыдущих пунктах мы генерировали статические страницы. Теперь разберёмся как отправлять данные на страницу. Изменим блок

```
{% block content %}
    <h1>Hello {{user}}!</h1>
{% endblock %}
```

В фигурных скобках указывается имя передаваемой переменной. Теперь наша функция будет выглядеть так:



```
@app.route('/')
def index():
    return render_template("index.html", user = "Viktor")
```

Рис.9 Измененная функция index

Запустите программу и убедитесь, что теперь выводится сообщение "Hello Viktor!".

- Очень часто приходится отображать табличные данные. Таблица в представлении Python может быть представлена как двумерный список.

Например так: `[[1, "Иван", "Иванов"], [2, "Петр", "Петров"]]`

Чтобы выводить табличные данные будем использовать встроенный в Jinja2 оператор цикла. А для оформления воспользуемся классами из bootstrap.

Теперь основная часть шаблона выглядит так:

```
{% block content %}
<div class="container">
  <table class="table table-bordered">
    <tr>
      <th class="text-center">id</th>
      <th class="text-center">Имя</th>
      <th class="text-center">Фамилия</th>
    </tr>
    {% for elem in elems %}
    <tr>
      <td class="text-center">{{ elem[0] }}</td>
      <td class="text-center">{{ elem[1] }}</td>
      <td class="text-center">{{ elem[2] }}</td>
    </tr>
    {% endfor %}
  </table>
</div>
{% endblock %}
```

А функция - вот так.



```
@app.route('/')
def index():
    data = [[1, "Иван", "Иванов"], [1, "Петр", "Петров"]]
    return render_template("i2.html", elems=data)
```

Рис.10 Измененная функция index

В результате при обращении к сайту должна вывестись похожая таблица.

id	Имя	Фамилия
1	Иван	Иванов
2	Петр	Петров

Рис.11 Пример таблицы

11. Чтобы получать данные с нашего сервера необходимо отправлять на него запросы и получать ответы в формате JSON. Для этого в python есть библиотека **requests**.

Поскольку данной библиотеки нет в виртуальной среде, создаваемой для Flask приложения, установим ее с помощью команды:

pip install requests

Абстрагируясь от создания интерфейса, попробуем получить данные с сервера. Для этого нам понадобится всего две строчки:



```
import requests
data = requests.get("http://localhost:8080/students").json()
```

Рис.12 Пример работы с requests

Теперь в переменной data лежит список JSON, у каждого из которого есть именованные поля. К которым можно обратиться через оператор “Квадратные скобки”

12. Для красивой обработки пользовательских действий очень часто используются формы. Разумеется можно использовать формы, объявленные в html, но гораздо удобнее создавать формы используя уже установленный нами модуль Flask-WTF.

Создадим форму для валидации студентов по группам. То есть, в зависимости от номера группы будет выводиться таблица студентов

ТОЛЬКО

ОДНОЙ

ГРУППЫ.

Уточнение - таблица запросов и возвращаемых результатов, используемых на сервере в текущем примере. Если в вашей работе запросы отличаются - не забудьте их изменить.

Запрос	Результат
http://localhost:8080/students	JSON со всеми студентам
http://localhost:8080/students/id id > 0	JSON студентов группы с номером, равном id
http://localhost:8080/groups	JSON групп (номер - название)

```
def get_group(id = None):
    data = []
    if not id or id=='0':
        data = requests.get("http://localhost:8080/students").json()
    else:
        try:
            data = requests.get("http://localhost:8080/students/{}".format(id)).json()
        except Exception as e:
            pass
    return data

def get_group_list():
    data = requests.get('http://localhost:8080/groups').json()
    d = [(str(i['id']),str(i['Name'])) for i in data]
    d = [('0', 'Все')] + d
    return d

class SelectForm(FlaskForm):
    group = SelectField('Группа', choices=get_group_list())
    ok = SubmitField("Выбрать")

@app.route('/')
def index():
    form = SelectForm()
    data = None
    if (form.group.data!="None"):
        data = get_group(form.group.data)
    else:
        data = get_group()
    return render_template("index.html", students = data, form = form)
```

Рис.13 Измененный модуль routes.py

Функция **get_group_list** - возвращает список пар id-группы : название группы

Функция **get_group** - возвращает список студентов заданной группы, или если группа не задана - список всех студентов

Класс **SelectForm** - шаблон класса. SelectedField - выпадающий список. SubmitField - кнопка отправляющая запрос.

Теперь мы в функции index(), передаем не только список студентов, но и форму, которую создаём. Важно не забыть изменить шаблон, см. Приложение 1.

Теперь в зависимости от того, какой из элементов выпадающего меню выбран, будут отображаться студенты соответствующей группы.

Группа

Все

ID	Имя	Фамилия	Отчество	ID группы
1	Иван	Иванов И.И.	Иванов	1
2	Марина	Зубкова	Геннадьевна	3
3	Юрий	Воронцов	Алексеевич	1
4	Руслан	Гафланов	Ингарович	1
5	Vic	Paperno	Alex	2
6	Александр	Звонников	Сергеевич	1
7	Марина	Зубкова	Геннадьевна	1
8	Данара	Манджиева	Альбертовна	1
9	Валерий	Мацарский	Александрович	1
10	Виктор	Паперно	Александрович	4
11	Антон	Мусин	Игоревич	1
12	Виктор	Паперно	Александрович	1
13	Андрей	Прохоров	Валерьевич	1
14	Юрий	Сергеев	Дмитриевич	1
15	Владислав	Штендер	Андреевич	1
20	Бригада_6	Студент_2	Тест	4
50	Vic	Paperno	Alex	5
777	test	test	test	1
1111	Владислав	Штендер	Андреевич	5
2222	Антон	Мусин	Игоревич	5

Рис.14 Результат работы

13. Теперь у вас есть все необходимые навыки для создания клиентского приложения любой сложности. Но если возникнут вопросы, можно обратить своё внимание на статьи на сайте Habr.com <https://habr.com/ru/post/346306/> . Там очень подробно и с примерами рассказывается о создании веб-сервисов с использованием Flask.
14. В дальнейшем, приложение написанное на Flask может быть выложено на сервер. Рекомендую бесплатный сервис <https://pythonanywhere.com/> - он даже предоставляет свой бесплатный домен, то есть ваше приложение будет сразу доступно пользователям интернета для тестирования.

Приложение 1

```
{% import 'bootstrap/wtf.html' as wtf %}

<!-- Шаблон HTML документа -->

{% block content %}
  <div class="container">
    <div class="row">
      <div class="col-md-2">
        {{ wtf.quick_form(form) }}
      </div>
    </div>
    <br>
    <table class="table table-bordered">
      <tr>
        <th class="text-center">Имя</th>
        <th class="text-center">Фамилия</th>
      </tr>
      {% for elem in elems %}
      <tr>
        <td class="text-center">{{ elem[0] }}</td>
        <td class="text-center">{{ elem[1] }}</td>
      </tr>
      {% endfor %}
    </table>
  </div>

{% endblock %}
```