

## Введение

В данном обзоре мы будем обозревать такой веб фреймворк как Flask.

Данный фреймворк предназначен для разработки веб приложений, использующих язык программирования Python. Сам же Flask использует набор инструментов Werkzeug и шаблонизатор Jinja2. Совместим же данный фреймворк с Python версии 2.7, 3.3 и выше.

## Плюсы и минусы

### Плюсы

#### Легкий для изучения

Flask имеет простую структуру и понятный синтаксис, также фреймворк позволяет программисту полностью контролировать процесс разработки.

#### Гибкий

Это свойство говорит о том, что программисту доступна возможность редактирования инструментов фреймворка под свои нужды.

#### Хорошие инструменты для тестирования

Flask имеет предустановленные инструменты для тестирования и отладки: unit тесты, встроенный отладчик и обработчик запросов.

### Минусы

#### Не поддерживает асинхронность

Flask обрабатывает все запросы в один поток. Это значит что каждый запрос будет блокировать поток до окончания выполнения, а затем будет выполнен следующий запрос.

#### Недостаток возможностей

Flask не подходит для разработки больших веб приложений.

## Использование

Flask стоит использовать в следующих случаях:

- Если у веб приложения будет небольшой бэкенд
- Если требуется большой контроль над используемыми компонентами
- Если вы не знакомы с асинхронностью

- Если вы хотите получить больше опыта и возможностей для обучения

## Модули для Flask

### Flask

В самом фреймворке есть функции, которые вам могут пригодиться при обучении. Например:

**Flash** – функция, используемая для вывода всплывающих сообщений. Допустим пользователь вошел в систему и вы хотите вывести ему сообщение, что он авторизовался. Это будет выглядеть примерно так:

```
flash('Вы успешно авторизовались')
```

**Urf\_for** – функция для создание URL. Данная функция принимает конечную точку и преобразует ее в URL в виде строки.

Использование:

```
url_for('index')
```

**Redirect** – функция для перенаправления пользователя на другую страницу.

Использование:

```
return redirect(url_for('*Страница*'))
```

### Flask-Login

Данный модуль предназначен для добавления в ваше приложение авторизации.

Чтобы установить этот модуль требуется выполнить команду:

```
Pip install flask-login
```

Использовать можно так:

```
from flask_login import login_required, current_user
```

**login\_required** – используется для проверки того, что пользователь авторизован. То есть при использовании этой функции, пользователь должен быть авторизован для доступа к какой-нибудь странице вашего приложения.

**Current\_User** – функция, с помощью которой приложение понимает какой пользователь сейчас использует приложение. Это используется при разграничении доступа, например, к административной панели в вашем приложении.

## **Flask\_bcrypt**

Модуль для хэширования данных.

Установка:

```
pip install flask-bcrypt
```

Использование:

```
from flask.ext.bcrypt import Bcrypt
```

```
bcrypt = Bcrypt(app)
```

```
pw_hash = bcrypt.generate_password_hash('hunter2')
```

## **Как начать работать, используя Flask**

Для начала, требуется скачать Python совместимой с фреймворком версией.

Затем нужно создать терминал, в котором нужно воспользовавшись командой

```
Pip install flask
```

Установить фреймворк.

Затем нужно в этой же папке создать файл с расширением .py

В нашей ситуации мы рассмотрим минимальное приложение на Flask.

Пример кода есть в официальной документации. И будет он выглядеть так:

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

Далее, чтобы запустить наше приложение, в ранее созданном терминале нужно прописать команду

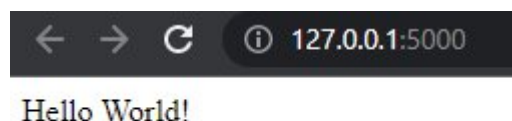
`Python *Имя_файла*.py`

Данная команда запустит наш проект.

При запуске вы увидите примерно следующее:

```
* Serving Flask app 'example' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

На данном адресе будет запущено наше приложение. Так оно выглядит в первоизданном виде:



The image shows a web browser address bar with the URL `127.0.0.1:5000`. Below the address bar, the text `Hello World!` is displayed in a monospaced font.

## Пример взаимодействия Flask и Jinja2

В нашем python файле находится следующий код.

```
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route("/")
5  def template_test():
6      return render_template('template.html', my_list=[0,1,2,3,4,5])
7
8  if __name__ == '__main__':
9      app.run(debug=True)
```

Так выглядит html файл с использованием шаблонизатора jinja2.

```
<!DOCTYPE html>
</body>
</html>
<html>
  <head>
    <title>Flask Template Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css" rel="stylesheet" media="screen">
    <style type="text/css">
      .container {
        max-width: 500px;
        padding-top: 100px;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <p>Loop through the list:</p>
      <ul>
        {% for n in my_list %}
        <li>{{n}}</li>
        {% endfor %}
      </ul>
    </div>
    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js"></script>
  </body>
</html>
```

Так будет выглядеть наша страница после запуска проекта.

Loop through the list:

- 0
- 1
- 2
- 3
- 4
- 5