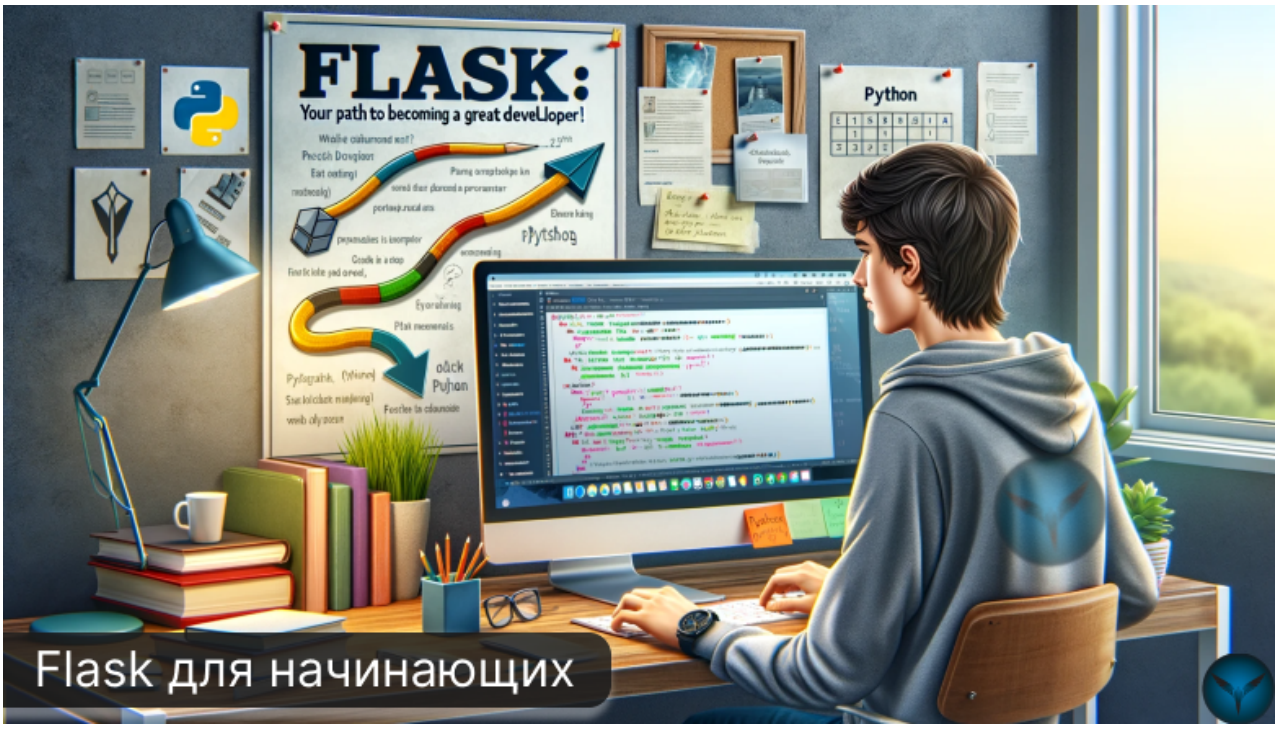
Flask для начинающих / Хабр

habr.com/ru/articles/783574

m_nikitin_dev December 27, 2023



Здравствуйте! Меня зовут Михаил, и я пишу эту статью специально для начинающих программистов, желающих изучить основы веб-разработки с использованием Flask. Я сам активно занимаюсь разработкой проектов на Flask и хочу поделиться своими знаниями и опытом, чтобы помочь вам начать свой путь в мире веб-разработки.

Flask — это прекрасный выбор для тех, кто уже знаком с основами Python и хочет применить эти знания в создании веб-приложений. Этот микро-фреймворк, написанный на Python, отличается своей простотой и гибкостью, делая его доступным для начинающих, в то же время предлагая мощные возможности для создания сложных приложений.

Требуемые навыки для чтения статьи:

- Основы ***Python***: понимание основных конструкций языка и способность писать простые программы.
- Знакомство с ***HTML***: базовое понимание тегов и структуры веб-страниц будет полезно.

Ориентировочное время чтения: Примерно 15-20 минут.

следующая статья / продолжение - Flask для начинающих — Часть 2 пишем landing page+admin panel с редактированием контента

Содержание

6. **База данных в Flask**
    - Обзор интеграции с базами данных (например, SQLite).
    - Пример простого CRUD-приложения (Создание, Чтение, Обновление, Удаление).
7. **Заключение**
    - Итоги о том, как Flask может быть использован для создания масштабируемых веб-приложений.
    - Рекомендации по дальнейшему изучению Flask и расширению знаний.

Установка и настройка Flask

Flask - это легкий и мощный веб-фреймворк для Python, который можно легко установить и настроить. В этом разделе мы покажем, как начать работу с Flask на вашем компьютере.

1. **Установка Python**

    Для работы с Flask вам понадобится Python версии 3.6 или выше. Если у вас еще не установлен Python, вы можете скачать его с официального сайта Python python.org.

2. **Установка Flask**
    - Flask можно установить с помощью инструмента управления пакетами Python, pip. Откройте терминал или командную строку и введите команду:

      ```
      pip install Flask
      ```

    - Эта команда установит Flask и все необходимые зависимости.

3. **Создание базового проекта Flask**

    - Создайте новую папку для вашего проекта и перейдите в нее в терминале.
    - Внутри папки проекта создайте файл, например, `app.py`, который будет основным файлом вашего веб-приложения.
    - Откройте `app.py` в текстовом редакторе и напишите следующий код:

      ```
      from flask import Flask
      app = Flask(name) @app.route('/')
      def hello_world():
        return 'Hello, World!' if name == 'main':
        app.run(debug=True)
      ```

    - Этот код создает базовое веб-приложение, которое отображает "Hello, World!" на главной странице.

4. **Запуск приложения**

    - Чтобы запустить приложение, вернитесь в терминал и выполните команду:

      ```
      python app.py
      ```

    - После этого откройте веб-браузер и перейдите по адресу `http://127.0.0.1:5000/`. Вы должны увидеть сообщение "Hello, World!".

Эти шаги демонстрируют, как легко начать работу с Flask. Вы установили Flask, создали простое приложение и успешно запустили его на локальном сервере.

Основы Flask

Flask предлагает гибкий и интуитивно понятный способ создания веб-приложений. Основой Flask является маршрутизация запросов и генерация ответов. В этом разделе мы рассмотрим эти ключевые концепции.

1. **Маршрутизация**

   - Маршрутизация в Flask отвечает за соединение функций с конкретными URL-адресами. Это позволяет вашему приложению реагировать на разные URL, например, отображать разные страницы.

   - Пример маршрута:

     ```
     @app.route('/about')
     def about():
       return 'This is the about page'
     ```

   - В этом примере, если пользователь перейдет по адресу `http://127.0.0.1:5000/about`, он увидит сообщение "This is the about page".

2. **Представления и Функции**

   - Представления в Flask – это функции Python, которые обрабатывают запросы и возвращают ответы. Как правило, ответ представляет собой HTML-страницу, но это также может быть текст, JSON или XML.

   - Пример функции представления:

     ```
     @app.route('/user/<username>')
     def show_user_profile(username):
       return f'User {username}'
     ```

   - Эта функция возвращает пользовательское имя, которое передается в URL.

3. **Использование Шаблонов**

- Flask использует систему шаблонов Jinja2, которая позволяет создавать HTML-страницы с динамическим содержимым.
- Пример использования шаблона:

```
from flask import Flask
from flask import render_template app =
Flask(name) @app.route('/hello/<name>')
def hello(name):
  return render_template('hello.html', name=name) if name == 'main':
  app.run(debug=True) # debug true задаем специально для разработки (в
данном случае при обновление/изменение кода приложение автоматически
само обновит данные на сайте)
```

- Здесь мы передаем переменную `name` в шаблон `hello.html`, который может динамически отображать разное содержимое в зависимости от переданного значения.
- Конечно, давайте добавим пример кода для шаблона `hello.html`, который будет использоваться в нашем примере с Flask. Этот шаблон будет простым HTML-документом, который динамически отображает имя пользователя, переданное через URL.

  Пример кода для hello.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="UTF-8">
   <title>Hello Page</title>
  </head>
  <body>
    <h1>Hello, {{ name }}!</h1>
    <p>Welcome to our simple Flask app.</p>
  </body>
</html>
```

  В этом шаблоне используется Jinja2 синтаксис для вставки переменной `name` в HTML. Когда Flask обрабатывает этот шаблон, он заменяет `{{ name }}` на значение, переданное в функцию `hello(name)`.

  Этот шаблон должен быть сохранен в папке `templates` в корне вашего проекта Flask. Flask автоматически ищет шаблоны в папке с именем `templates`, поэтому важно соблюдать эту структуру для корректной работы приложения

- **Пример структуры проекта**

```
/your-app
   /static
   /templates
   app.py
```

1. **app.py**: Это основной файл вашего приложения, где вы определяете маршруты, представления и бизнес-логику. В этом файле вы создаете экземпляр приложения Flask и запускаете сервер.

2. **/templates**: Это директория, в которой хранятся все HTML-шаблоны. Flask автоматически ищет шаблоны в папке с именем `templates`. Примеры шаблонов могут включать `hello.html`, `index.html`, `about.html` и так далее.

3. **/static**: Эта папка используется для хранения статических файлов, таких как CSS-стили, JavaScript-скрипты, изображения и другие файлы, которые не изменяются в процессе работы приложения. Flask автоматически обрабатывает запросы к этим файлам.

Эта структура является довольно стандартной для небольших проектов на Flask. Для более сложных приложений вы можете добавить дополнительные папки и файлы, например, для моделей базы данных, утилит, тестов и так далее. Однако даже в сложных проектах обычно сохраняется базовая организация с папками `static` и `templates`.

Теперь у вас есть полный пример, как использовать шаблоны в Flask для динамического отображения данных. Это базовый пример, но он хорошо демонстрирует мощь и гибкость шаблонизатора Jinja2 в сочетании с Flask.

Эти основные концепции являются ключевыми для понимания и создания веб-приложений с помощью Flask. Они позволяют вам строить структурированные и масштабируемые веб-приложения.

Работа с данными в Flask

1. **Обработка входящих запросов**

    - Flask обеспечивает простой способ обработки входящих запросов, будь то GET или POST запросы. Это делается через маршрутизацию и функции представлений.

    - Пример обработки GET запроса:

      ```
      @app.route('/greet/<name>')
      def greet(name):
        return f'Hello, {name}!'
      ```

      В этом примере, Flask обрабатывает запрос к URL `/greet/<name>` и возвращает приветствие с именем пользователя.

    - Пример обработки POST запроса:

      ```
      from flask import request @app.route('/submit', methods=['POST'])
      def submit():
        name = request.form['name']
        return f'Hello, {name}'
      ```

      Здесь мы обрабатываем POST запрос и извлекаем данные из формы.

2. **Работа с формами**

    - Для работы с формами в Flask часто используется объект `request`, который позволяет получить доступ к данным, отправленным пользователем.

    - Пример HTML-формы:

      ```
      <form method="post" action="/submit">
        <input type="text" name="name">
        <input type="submit" value="Submit">
      </form>
      ```

      Эта форма отправляет данные на маршрут `/submit`, который мы обрабатываем в нашем Flask приложении.

3. **Отправка данных**

    - Flask позволяет не только принимать данные, но и отправлять их обратно пользователю. Это может быть в виде HTML, JSON, XML и других форматов.

    - Пример отправки JSON:

      ```
      from flask import jsonify @app.route('/data')
      def data():
        return jsonify({'key': 'value'})
      ```

      Этот маршрут возвращает данные в формате JSON.

Эти примеры демонстрируют основные способы работы с данными в Flask. От обработки простых GET запросов до более сложных операций с формами и отправки данных, Flask предлагает гибкий и мощный инструментарий для веб-разработчиков.

Интеграция с базами данных в Flask

1. **Выбор базы данных**

    - Flask поддерживает интеграцию с множеством различных баз данных, включая SQL и NoSQL. Выбор базы д

    - анных зависит от нужд вашего приложения. Для простых приложений популярным выбором является SQLite из-за его легкости и неприхотливости в настройке.

2. **Использование SQLAlchemy**

    - SQLAlchemy - это ORM (Object-Relational Mapping) библиотека для Python, которая облегчает работу с базами данных SQL. В Flask ее можно использовать для упрощения операций с базой данных.

    - Пример подключения к базе данных с использованием SQLAlchemy:

      ```
      from flask import Flask
      from flask_sqlalchemy import SQLAlchemy app = Flask(name) # подключение
      базы данных
      app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
      db = SQLAlchemy(app)
      ```

      Здесь мы конфигурируем подключение к SQLite базе данных.

3. **Создание моделей**

    - Модели в SQLAlchemy представляют таблицы баз данных. Они позволяют взаимодействовать с данными на более высоком уровне абстракции.

    - Пример создания модели:

```
class User(db.Model):
  id = db.Column(db.Integer, primary_key=True)
  username = db.Column(db.String(80), unique=True, nullable=False)
  email = db.Column(db.String(120), unique=True, nullable=False)   def
__repr__(self):
        return f'<User {self.username}>'
```

4. **Операции CRUD**

    - CRUD означает Создание (Create), Чтение (Read), Обновление (Update), Удаление (Delete). SQLAlchemy упрощает выполнение этих операций.

    - Пример создания нового пользователя:

```
new_user = User(username='newuser', email='email@example.com')
db.session.add(new_user)
db.session.commit()
```

    - Пример чтения пользователя:

```
User.query.filter_by(username='newuser').first()
```

5. **Миграции**

    Для управления изменениями в структуре базы данных можно использовать расширение Flask-Migrate. Оно облегчает процесс создания и применения миграций.

Этот раздел статьи дает представление о том, как начать работу с базами данных в Flask. Использование ORM, такого как SQLAlchemy, может значительно упростить разработку и поддержку ваших веб-приложений.

Заключение

В этой статье мы рассмотрели основные аспекты работы с Flask, включая

- установку,
- основы маршрутизации и представлений,
- работу с шаблонами,
- обработку данных и интеграцию с базами данных.

Flask демонстрирует свою силу как легкий, но мощный инструмент для создания веб-приложений на Python. Его простота и гибкость делают его отличным выбором как для начинающих, так и для опытных разработчиков.

Если у вас остались или появились какие-то вопросы готов помочь разобраться начинающим программистам по работе с Flask

Спасибо за ваше уделенное время и поддержку!

следующая статья / продолжение - Flask для начинающих — Часть 2 пишем landing page+admin panel с редактированием контента