В данной статье я создам свое первое веб-приложения на Flask, а также попрактикуюсь в использовании шаблонов.

Но прежде чем начать, давайте разберемся что такое вообще Flask. Flask– это легкая веб‑платформа Python, которая предоставляет полезные инструменты и функции для создания веб‑приложений на языке Python. Это дает разработчикам гибкость и является доступной средой для новых разработчиков, поскольку вы можете быстро создать веб‑приложение, используя только один файл Python.

**Создание и запуск своего первого веб-приложения**

Работать я буду в PyCharm. Поэтому, первым делом я создаю свой проект и далее устанавливаю Flask через встроенный терминал с помощью команды **pip install flask**. Далее, я создал файл **app1.py**, где будет написано простое веб-приложение, отображающее HTML-код в браузере.

Открыв файл, я написал следующий код:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return '<h1>Hello, World!</h1>'

В нем мы сначала импортируем объект Flask из пакета flask. Затем используем его для создания экземпляра приложения Flask, присвоив ему имя app. Мы передаем специальную переменную \_\_name\_\_, которая содержит имя текущего модуля Python. Это имя сообщает экземпляру, где он находится; это нужно, потому что Flask устанавливает некоторые пути за кулисами.

Создав экземпляр приложения, мы можем использовать его для обработки входящих веб‑запросов и отправки ответов пользователю. @app.route — это декоратор, который превращает обычную функцию Python в функцию просмотра Flask, который преобразует возвращаемое значение функции в HTTP-ответ, отображаемый HTTP-клиентом, например, веб‑браузером. Мы передаем значение ‘/‘ в @app.route (), чтобы указать, что эта функция будет отвечать на веб‑запросы для URL-адреса /, который является основным URL-адресом.

Функция просмотра hello() возвращает строку ‘<h1>Hello, World!</h1>‘ в качестве ответа HTTP.

Теперь у нас есть простое приложение Flask в файле Python с именем **app1.py.**

**Далее, необходимо его запустить чтобы увидеть результат** функции просмотра hello(), отображаемой в веб‑браузере. Как это сделать? Во встроенном терминале я прописываю команду **set FLASK\_APP=app1**, благодаря которой я сообщаю Flask, где найти приложение app1. После, прописываю **set FLASK\_ENV=development**, которая говорит о том, что я хочу запустить приложение в режиме разработки (чтобы можно было использовать отладчик для обнаружения ошибок). Наконец, я прописываю **flask run**. После запуска, вы увидите следующее:

\* Serving Flask app 'app1' (lazy loading)

\* Environment: development

\* Debug mode: on

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 188-110-549

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Эти выходные данные содержат несколько частей информации:

* Название приложения, которое вы запускаете (**«app1»**).
* Среда, в которой выполняется приложение (**development**).
* **Debug mode** (Режим отладки): **on** означает, что отладчик Flask запущен. Это полезно при разработке, поскольку предоставляет подробные сообщения об ошибках, когда что-то идет не так, что упрощает поиск и устранение неисправностей.
* Приложение работает локально по URL-адресу **http://127.0.0.1:5000/**. **127.0.0.1** — это IP-адрес, который представляет локальный хост вашего компьютера, а: **5000** — это номер порта.

Далее, открываем браузер и введим URL-адрес **http://127.0.0.1:5000**/. Вы увидите текст “**Hello, World!**” в качестве ответа. Это подтверждает, что ваше приложение успешно работает.

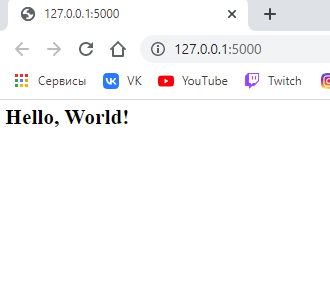


Рисунок 1 – Результат работы приложения

Кроме этого, можно добавить несколько маршрутов для отображения разных страниц в зависимости от запрошенного URL.

Маршрут — это URL-адрес, который вы можете использовать, чтобы определить, что пользователь получает, когда посещает ваше веб‑приложение в своем браузере. Например, http://127.0.0.1:5000/ — это основной маршрут, который может использоваться для отображения главной страницы. URL-адрес http://127.0.0.1:5000/about может быть другим маршрутом, используемым для страницы «About», которая дает посетителю некоторую информацию о вашем веб‑приложении.

Для этого, в **app1.py** прописываем следующее:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return '<h1>Hello, World!</h1>'

@app.route('/about/')

def about():

return '<h3>This is a Flask web application.</h3>'

Здесь, мы добавили новую функцию about(). Эта функция украшена декоратором @app.route(), который преобразует ее в функцию представления, которая обрабатывает запросы к конечной точке **http://127.0.0.1:5000/about**. Для просмотра результата переходим по этому URL-адресу.

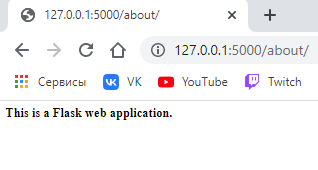


Рисунок 2 – Результат использования маршрута

Но можно использовать и несколько маршрутов для одной функции просмотра. Для этого, изменяем **app1.py** следующим образом:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/index/')

def hello():

return '<h1>Hello, World!</h1>'

@app.route('/about/')

def about():

return '<h3>This is a Flask web application.</h3>'

После добавления этого нового декоратора вы можете получить доступ к странице индекса как по адресу **http://127.0.0.1:5000/**, так и по адресу **http://127.0.0.1:5000/index**.

Теперь вы понимаете, что такое маршруты, как использовать их для создания функций просмотра и как добавлять новые маршруты в ваше приложение. Затем вы воспользуетесь динамическими маршрутами, чтобы позволить пользователям управлять ответом приложения.

Кроме этого, хотелось бы разобрать динамические маршруты, то есть мы создадим маршрут, в котором слова, переданные через URL, будут написаны с заглавной буквы, и маршрут, который складывает два числа и отображает результат.

Для этого, открываем наш **app1.py** и добавляем:

from markupsafe import escape

from flask import Flask

# ...

@app.route('/capitalize/<word>/')

def capitalize(word):

return '<h1>{}</h1>'.format(escape(word.capitalize()))

Этот новый маршрут имеет переменную секцию. Это говорит Flask взять значение из URL-адреса и передать его функции просмотра. Переменная URL передает аргумент ключевого слова функции просмотра capitalize(). Аргумент имеет то же имя, что и переменная URL (в данном случае слово). При этом вы можете получить доступ к слову, переданному через URL-адрес, и ответить его версией с заглавной буквы, используя метод capitalize() в Python.

Мы используем функцию escape (), которую мы импортировали, чтобы отобразить строку слова как текст. Это важно, чтобы избежать атак с использованием межсайтовых сценариев (XSS). Если пользователь отправляет вредоносный код JavaScript вместо слова, escape() отобразит его как текст, а браузер не запустит его, сохраняя безопасность вашего веб‑приложения.

Чтобы отобразить слово с заглавной буквы внутри заголовка HTML <h1>, используйте метод Python format().

Запустив сервер разработки, открыв браузер и передав по следующим URL-адресам. Вы можете заменить выделенные слова любым словом по вашему выбору. Например:

http://127.0.0.1:5000/capitalize/hello

http://127.0.0.1:5000/capitalize/flask

http://127.0.0.1:5000/capitalize/python

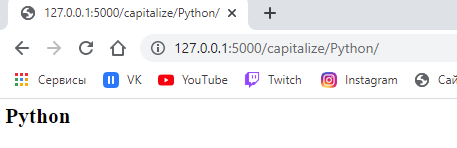


Рисунок 3 – Результат динамического маршрута

Также можно использовать несколько переменных в маршруте. Чтобы продемонстрировать это, необходимо добавить маршрут, который складывает два положительных целых числа и отображает результат.

Откроем файл app1.py для редактирования и добавим в конец файла следующий маршрут:

# ...

@app.route('/add/<int:n1>/<int:n2>/')

def add(n1, n2):

return '<h1>{}</h1>'.format(n1 + n2)

В этом маршруте вы используете специальный преобразователь int с переменной URL (/add///), который принимает только положительные целые числа. По умолчанию переменные URL-адреса считаются строками и рассматриваются как таковые.

Запустив сервер разработки, открыв браузер и перейдя по следующему URL-адресу:

http://127.0.0.1:5000/add/5/5/

мы увидим следующий результат:

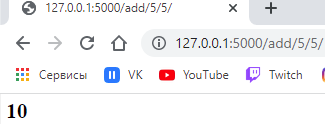


Рисунок 4 – Использование нескольких переменных в маршруте

При разработке веб‑приложения вы часто будете сталкиваться с ситуациями, когда приложение отображает ошибку вместо ожидаемого поведения. Вы можете неправильно написать переменную или забыть определить или импортировать функцию. Чтобы упростить решение этих проблем, Flask предоставляет отладчик при запуске приложения в режиме разработки. На этом шаге вы узнаете, как исправить ошибки в вашем приложении с помощью отладчика Flask.

Чтобы продемонстрировать, как обрабатывать ошибки, вы создадите маршрут, который приветствует пользователя из списка имен пользователей.

Откроем файл app1.py для редактирования и добавим в конец файла следующий маршрут:

# ...

@app.route('/users/<int:user\_id>/')

def greet\_user(user\_id):

users = ['Bob', 'Jane', 'Adam']

return '<h2>Hi {}</h2>'.format(users[user\_id])

В приведенном выше маршруте функция просмотра greet\_user() получает аргумент user\_id из переменной URL user\_id. Мы используем преобразователь int, чтобы принимать положительные целые числа. Внутри функции у нас есть список Python под названием users, который содержит три строки, представляющие имена пользователей. Функция просмотра возвращает строку, которая создается в зависимости от предоставленного user\_id. Если user\_id равен 0, ответом будет Привет, Боб в теге <h2>, потому что Боб является первым элементом в списке (значение users[0]).

Запустив сервер разработки, откроем браузер и перейдем по следующим URL-адресам:

http://127.0.0.1:5000/users/0

http://127.0.0.1:5000/users/1

http://127.0.0.1:5000/users/2

Мы получим следующие ответы:

Hi Bob

Hi Jane

Hi Adam

Пока всё работает хорошо, но может пойти не так, если мы запросим приветствие для пользователя, которого не существует. Чтобы продемонстрировать, как работает отладчик Flask, наберем следующий URL-адрес:

http://127.0.0.1:5000/users/3

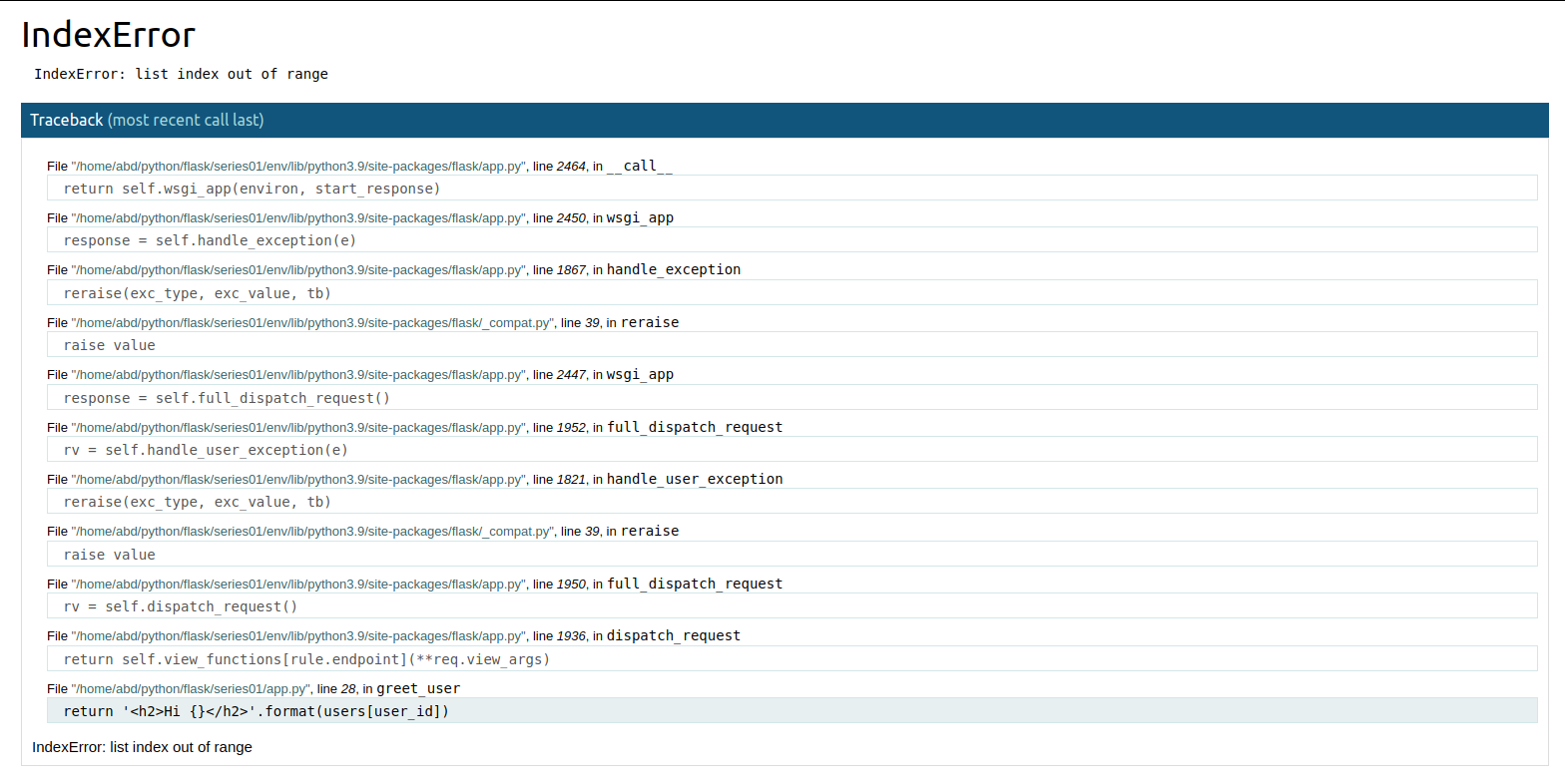


Рисунок 5 – Стандартная ошибка

Вверху на странице указано имя исключения Python, которым является IndexError, что указывает на то, что индекс списка (в данном случае 3) находится за пределами диапазона списка (который составляет только от 0 до 2, потому что в списке всего три элемента). В этом случае вы можете использовать простую команду try ... except, чтобы исправить эту ошибку. Если запрошенный URL-адрес имеет индекс за пределами диапазона списка, пользователь получит ошибку 404 Not Found, которая является ошибкой HTTP, которая сообщает пользователю, что страница, которую он ищет, не существует.

Откроем файл **app1.py** для редактирования. Чтобы ответить с ошибкой HTTP 404, нам понадобится функция Flask abort(), которую можно использовать для создания ответов об ошибках HTTP. Изменим вторую строку файла, чтобы также импортировать эту функцию:

from markupsafe import escape

from flask import Flask, abort

Затем отредактируем функцию просмотра greet\_user(), чтобы она выглядела следующим образом:

# ...

@app.route('/users/<int:user\_id>/')

def greet\_user(user\_id):

users = ['Bob', 'Jane', 'Adam']

try:

return '<h2>Hi {}</h2>'.format(users[user\_id])

except IndexError:

abort(404)

Мы использовали попытку выше, чтобы проверить возвращаемое выражение на наличие ошибок. Если ошибки не было, то есть user\_id имеет значение, соответствующее индексу в списке пользователей, приложение ответит соответствующим приветствием. Если значение user\_id выходит за пределы диапазона списка, возникает исключение IndexError,  
и мы используем except, чтобы поймать ошибку и ответить ошибкой **HTTP 404** с помощью вспомогательной функции Flask abort().

Теперь, когда сервер разработки запущен, снова посетим тот URL-адрес. На этот раз мы увидии стандартную страницу с ошибкой 404, информирующую пользователя о том, что страница не существует.

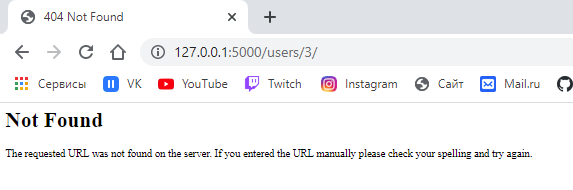


Рисунок 6 – Ошибка 404

Проделав все эти действия, мы имеем общее представление о том, что такое Flask, как его установить и как использовать для написания веб‑приложения, как запустить сервер разработки и как использовать маршруты и функции просмотра для отображения различных веб‑страниц, обслуживающих определенные целей. Мы также узнали, как использовать динамические маршруты, чтобы позволить пользователям взаимодействовать с вашим веб‑приложением через URL-адрес, и как использовать отладчик для устранения ошибок.

**Как использовать шаблоны в приложении Flask**

## 1. Визуализация шаблона и использование переменных

## Создав просто веб-приложение на Flask, я решил попробовать использование шаблонов. Для этого, я создал файл app2.py.

## В него добавляем следующее:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return render\_template('index.html')

В этом блоке кода мы импортируем класс Flask и функцию render\_template() из пакета flask. Мы используем класс Flask для создания экземпляра приложения Flask с именем app. Затем мы определяем функцию просмотра (которая является функцией Python, которая возвращает ответ [HTTP](http://ru.wikipedia.org/wiki/Special:Search/HTTP" \o "wikipedia: HTTP" \t "wikipedia)[એ](http://ru.wikipedia.org/wiki/Special:Search/HTTP" \o "wikipedia: HTTP" \t "wikipedia)) с именем hello(), используя [декоратор](https://chel-center.ru/python-yfc/2020/03/08/dekoratory-v-python/) app.route(), который преобразует обычную функцию в функцию просмотра. Эта функция представления использует функцию render\_template() для визуализации файла шаблона с именем **index.html**.

Затем нам нужно будет создать файл шаблона **index.html** в каталоге с именем **templates** внутри каталога **flask\_app**. Flask ищет шаблоны в каталоге **templates**, который называется шаблонами, поэтому имя важно. Для этого выполним следующую команду, чтобы создать каталог шаблонов:

mkdir templates

Затем откроем файл с именем **index.html** и добавим следующий HTML‑код:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FlaskApp</title>

</head>

<body>

<h1>Hello World!</h1>

<h2>Welcome to FlaskApp!</h2>

</body>

</html>

Здесь мы устанавливаем заголовок, добавляем Hello World! сообщение в виде заголовка <h1> и создаем сообщение Welcome to FlaskApp! как заголовок <h2>. Далее запускаем наше приложение. (как это сделать говорилось выше)

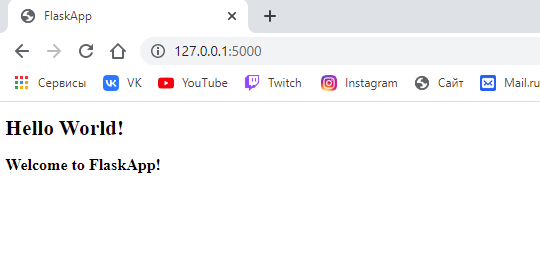


Рисунок 7 – Результат работы приложения

Оставим сервер запущенным и откроем файл **app2.py** для редактирования. Импортируем модуль datetime из стандартной библиотеки Python и отредактируем функцию index(), чтобы файл выглядел следующим образом:

import datetime

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return render\_template('index.html', utc\_dt=datetime.datetime.utcnow())

Здесь мы импортировали модуль datetime и передали в шаблон **index.html** переменную с именем utc\_dt со значением datetime.datetime.utcnow(), которое является текущей датой и временем в формате UTC.

Затем, чтобы отобразить значение переменной на странице индекса, откроем файл **index.html** для редактирования. Отредактируем файл, чтобы он выглядел следующим образом:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FlaskApp</title>

</head>

<body>

<h1>Hello World!</h1>

<h2>Welcome to FlaskApp!</h2>

<h3>{{ utc\_dt }}</h3>

</body>

</html>

Мы добавили заголовок <h2> со специальным разделителем {{...}} для печати значения переменной utc\_dt. Откроем браузер и перейдём на главную страницу:

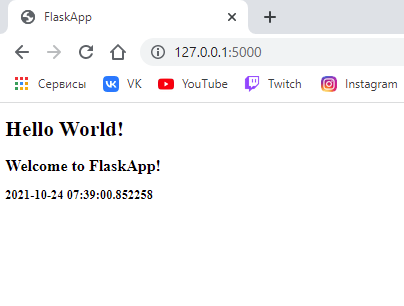


Рисунок 8 – Работа приложения с отображением переменной

Теперь мы создали страницу индекса с шаблоном HTML в приложении Flask, отрисовали шаблон, передали и отобразили значение переменной. Затем мы избежим повторения кода, используя наследование шаблонов.

## 2. Использование механизма наследования шаблонов

## На этом этапе мы создадим базовый шаблон с содержанием, которым можно будет поделиться с другими шаблонами. Мы отредактируем свой шаблон индекса, чтобы он унаследовал от базового шаблона. Затем мы создадим новую страницу, которая будет служить страницей «О приложении», где пользователи смогут найти дополнительную информацию о вашем приложении.

## Для этого, создадим **base.html и пропишем следующий код:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title %} {% endblock %} - FlaskApp</title>

<style>

nav a {

color: #d64161;

font-size: 3em;

margin-left: 50px;

text-decoration: none;

}

</style>

</head>

<body>

<nav>

<a href="#">FlaskApp</a>

<a href="#">About</a>

</nav>

<hr>

<div class="content">

{% block content %} {% endblock %}

</div>

</body>

</html>

Большая часть кода в этом файле представляет собой стандартный HTML‑код, заголовок, некоторые стили для ссылок навигации, панель навигации с двумя ссылками, одна для страницы индекса, другая для еще не созданной страницы «О программе», а также <div> для содержание страницы. (Ссылки пока не работают; на следующем шаге будет показано, как создавать ссылки между страницами). Однако, следующие выделенные части относятся к шаблонизатору Jinja:

* {% block title %} {% endblock %}: блок, который служит заполнителем для заголовка. Позже вы будете использовать его в других шаблонах, чтобы задать собственный заголовок для каждой страницы в приложении, не переписывая каждый раз весь раздел.
* {% block content %} {% endblock %}: другой блок, который будет заменен содержимым в зависимости от дочернего шаблона (шаблона, наследуемого от base.html), который заменит его.

Теперь, когда у нас есть базовый шаблон, мы можем воспользоваться им, используя наследование. Откроем файл **index.html** и затем заменим его содержимое следующим:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Index {% endblock %}</h1>

<h1>Hello World!</h1>

<h2>Welcome to FlaskApp!</h2>

<h3>{{ utc\_dt }}</h3>

{% endblock %}

Здесь мы используем тег {% extends %} для наследования от шаблона **base.html**. Затем мы расширяем его, заменяя блок содержимого в базовом шаблоне тем, что находится внутри блока содержимого в предыдущем блоке кода.

Этот блок содержимого содержит тег <h1> с текстовым индексом внутри основной надписи,который, в свою очередь, заменяет исходную основную надпись в шаблоне **base.html** текстовым индексом, так что полный заголовок становится индексом — FlaskApp. Таким образом, мы можем избежать повторения одного и того же текста дважды, поскольку он работает как заголовок страницы, так и заголовок, который появляется под панелью навигации, унаследованной от базового шаблона. Затем у нас есть еще несколько заголовков: один заголовок <h1> с текстом Hello World!, заголовок <h2> и заголовок <h3>, содержащий значение переменной utc\_dt.

Наследование шаблонов дает нам возможность повторно использовать HTML‑код, который есть в других шаблонах (в данном случае **base.html**), без необходимости повторять его каждый раз, когда это необходимо. Сохраним и обновим страницу индекса в браузере. Страница будет выглядеть следующим образом:

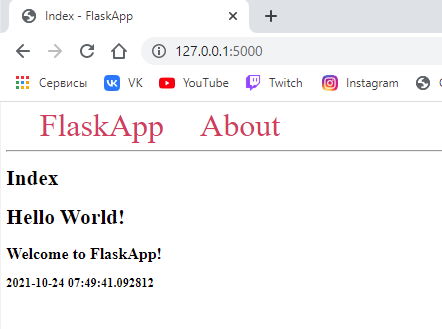


Рисунок 9 – Результат

Затем мы создаем страницу «About». Открываем файл app2.py, чтобы добавить новый маршрут и добавляем следующее:

# ...

@app.route('/about/')

**def** about():

**return** render\_template('about.html')

Здесь используется декоратор app.route() для создания функции просмотра с именем about(). В нем мы возвращаем результат вызова функции render\_template() с именем файла шаблона **about.html** в качестве аргумента.

Откроем файл шаблона с именем **about.html** для редактирования и добавьте следующий код:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} About {% endblock %}</h1>

<h3>FlaskApp **is** a Flask web application written **in** Python.</h3>

{% endblock %}

Здесь мы наследуем базовый шаблон с помощью тега extends, заменяем блок содержимого базового шаблона тегом <h1>, который также служит заголовком страницы, и добавляем тег <h3> с некоторой информацией о приложении.

Когда сервер разработки запущен, перейдем по следующему URL‑адресу в своем браузере:

http://127.0.0.1:5000/about

Мы увидим следующее:

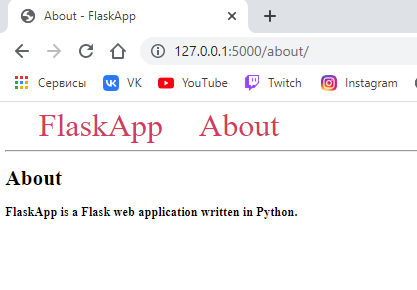


Рисунок 10 – Вывод страницы About

Мы создали базовый шаблон и использовали его на своей индексной странице и на странице сведений, чтобы избежать повторения кода.

## 3. Связывание страниц

На этом этапе мы свяжем страницы в шаблонах с помощью вспомогательной функции url\_for(). Мы добавим две ссылки на панель навигации в свой базовый шаблон, одну для главной страницы и другую для страницы «About».

Сначала откроем базовый шаблон для редактирования и внесите следующие изменения:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title %} {% endblock %} - FlaskApp</title>

<style>

nav a {

color: #d64161;

font-size: 3em;

margin-left: 50px;

text-decoration: none;

}

</style>

</head>

<body>

<nav>

<a href="{{ url\_for('hello') }}">FlaskApp</a>

<a href="{{ url\_for('about') }}">About</a>

</nav>

<hr>

<div **class**="content">

{% block content %} {% endblock %}

</div>

</body>

</html>

Здесь мы используем специальную функцию url\_for(), которая возвращает URL‑адрес функции просмотра, которую мы ему передаем. Первая ссылка ссылается на маршрут функции просмотра hello() (которая является страницей индекса). Вторая ссылка указывает на маршрут функции просмотра about(). Обратите внимание, что мы передаем имя функции просмотра, а не маршрут (/ или /about). Использование функции url\_for() для создания URL‑адресов помогает лучше управлять URL‑адресами. Если мы жестко запрограммируем URL‑адреса, наши ссылки сломаются, если мы изменим маршруты. С url\_for() мы можем редактировать маршруты и гарантировать, что ссылки будут работать должным образом. Функция url\_for() также заботится о других вещах, таких как экранирование специальных символов.

Теперь перейдем на главную страницу и попробуем ссылки на панели навигации. Мы увидим, что они работают.

## 4. Использование условных выражений и циклов

На этом этапе мы будем использовать операторы if в своих шаблонах, чтобы управлять тем, что отображать в зависимости от определенных условий. Мы также будем использовать циклы for для просмотра списков Python и отображения каждого элемента в списке. Мы добавим новую страницу, на которой будут отображаться комментарии в виде списка. Комментарии с четным порядковым номером будут иметь синий фон, а комментарии с нечетным порядковым номером будут отображаться на сером фоне.

Сначала мы создадим маршрут для страницы комментариев. Откроем файл **app2.py** для редактирования и добавьте следующий маршрут в конец файла:

# ...

@app.route('/comments/')

**def** comments():

comments = ['This is the first comment.',

'This is the second comment.',

'This is the third comment.',

'This is the fourth comment.'

]

**return** render\_template('comments.html', comments=comments)

В приведенном выше маршруте у нас есть список Python, называемый комментариями, который содержит четыре элемента. Мы возвращаете файл шаблона с именем **comments.html** в последней строке, передача переменной с именем comments, содержащей список, в файл шаблона.

Затем создадим и откроем новый файл **comments.html** в каталоге шаблонов для редактирования и добавим следующий код:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% **for** comment **in** comments %}

<div style="padding: 10px; background-color: #EEE; margin: 20px">

<p style="font-size: 24px">{{ comment }}</p>

</div>

{% endfor %}

</div>

{% endblock %}

Здесь мы расширяем шаблон **base.html** и заменяем содержимое блока содержимого. Во-первых, мы используем заголовок <h1>, который также служит заголовком страницы.

Мы используем цикл Jinja for в строке {% for comment in comments %}, чтобы просмотреть каждый комментарий в списке комментариев (который сохраняется в переменной комментария). Комментарий отображается в теге <p style = "font-size: 24px"> {{comment}} </p> так же, как мы обычно отображаем переменную в Jinja. Мы сигнализируем о завершении цикла for с помощью ключевого слова {% endfor %}. Это отличается от способа построения циклов for в Python, поскольку в шаблонах Jinja нет специального отступа.

Когда сервер разработки запущен, откроем браузер и перейдем на страницу комментариев:

http://127.0.0.1:5000/comments

Мы увидим следующее:

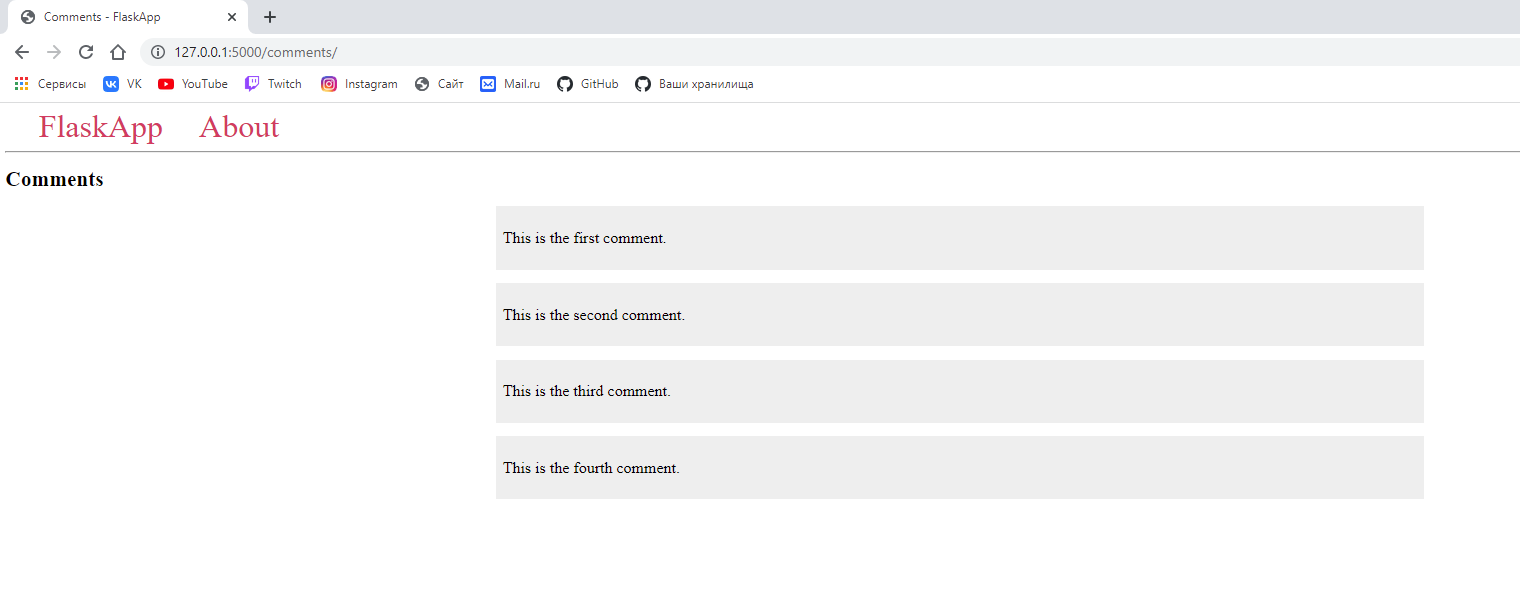


Рисунок 11 – Страница комментариев

Давайте откроем файл шаблона **comments.html** и отредактируйте его, чтобы он выглядел следующим образом:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% for comment in comments %}

{% if loop.index %2 == 0 %}

{% set bg\_color = '#e6f9ff' %}

{% else %}

{% set bg\_color = '#eee' %}

{% endif %}

<div style="padding: 10px; background-color: {{ bg\_color }}; margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment }}</p>

</div>

{% endfor %}

</div>

{% endblock %}

С помощью этого нового редактирования мы добавили оператор if в строку {% if loop.index %2 == 0 %}. Переменная цикла — это специальная переменная Jinja, которая дает нам доступ к информации о текущем цикле. Здесь мы используем loop.index, чтобы получить индекс текущего элемента, который начинается с 1, а не с 0, как в списках Python. Оператор if здесь проверяет, использует ли индекс даже оператор %. Он проверяет остаток от деления номера индекса на 2; если остаток равен 0, это означает, что номер индекса четный, в противном случае номер индекса нечетный. Тег {% set %} используется для объявления переменной с именем bg\_color. Если порядковый номер четный, мы устанавливаем его на голубоватый цвет, в противном случае, если номер индекса нечетный, мы устанавливаем переменную bg\_color на серый. Затем мы используем переменную bg\_color, чтобы установить цвет фона для тега <div>, содержащего комментарий. Над текстом комментария мы используем loop.index для отображения текущего номера индекса в теге <p>.

Сохраним и посмотрим результат:

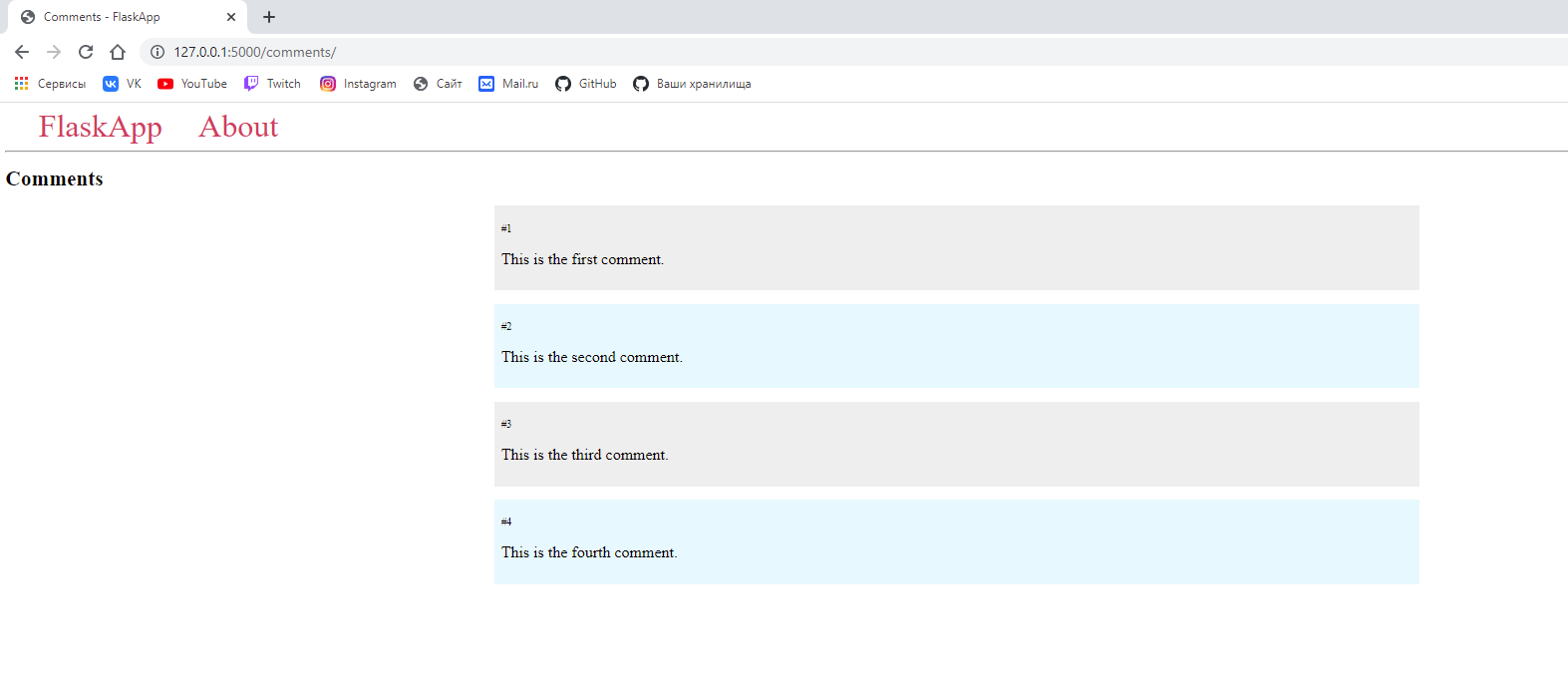


Рисунок 12 – Страница комментариев с синими блоками для четных

Кроме того, чтобы отобразить все комментарии, кроме второго, мы можем использовать оператор if с условием loop.index != 2, чтобы отфильтровать второй комментарий.

Откроем шаблон комментариев и отредактируем его, чтобы он выглядел следующим образом:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% **for** comment **in** comments %}

{% **if** loop.index != 2 %}

<div style="padding: 10px;

background-color: #EEE;

margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment }}</p>

</div>

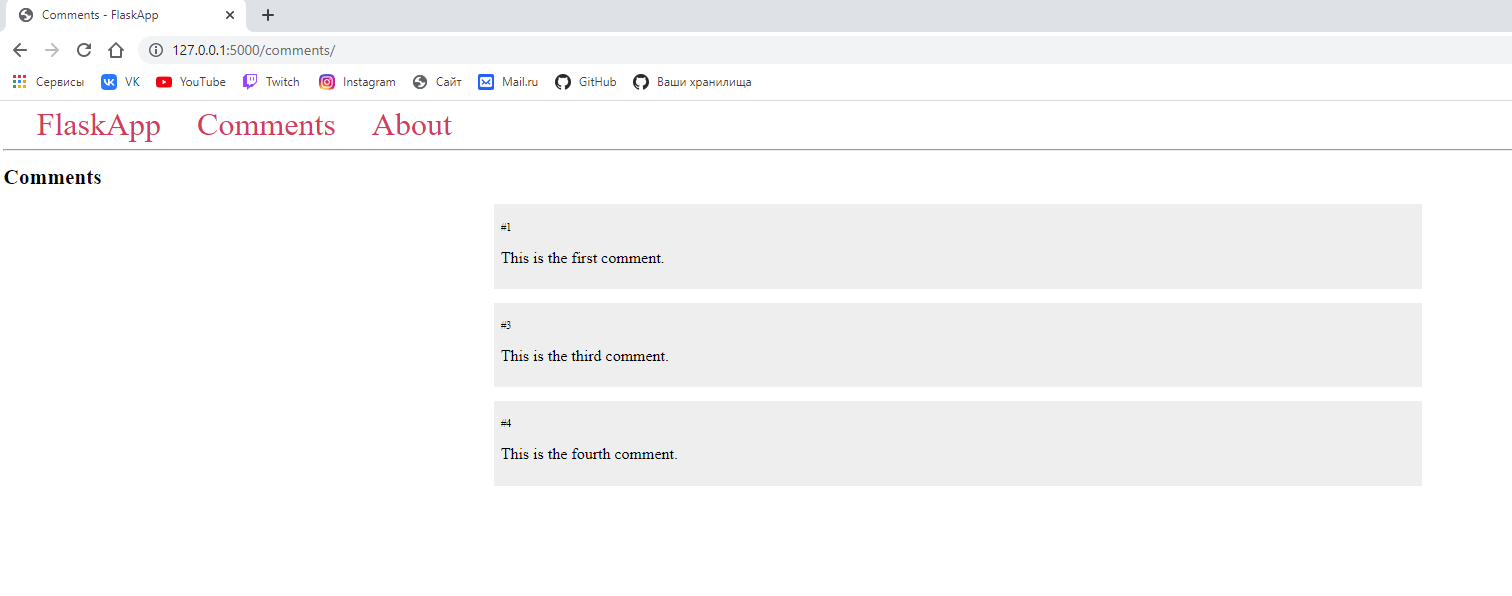
{% endif %}

{% endfor %}

</div>

{% endblock %}

Здесь мы используем {% if loop.index != 2 %}, чтобы отображать только те комментарии, которые не имеют индекса 2, что означает все комментарии, кроме второго. Мы также используем жестко запрограммированное значение для цвета фона вместо помощника loop.cycle(), чтобы упростить задачу, а остальное не изменяется. Мы завершаем оператор if, используя {% endif %}. Обновим страницу комментариев, и мы увидим, что второй комментарий не отображается.

Рисунок 13 – Результат без 2 комментария

Теперь нам нужно добавить ссылку, которая переводит пользователей на страницу комментариев на панели навигации. Откроем базовый шаблон для редактирования и изменим содержимое тега <nav>, добавив к нему новую ссылку <a>:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title %} {% endblock %} - FlaskApp</title>

<style>

nav a {

color: #d64161;

font-size: 3em;

margin-left: 50px;

text-decoration: none;

}

</style>

</head>

<body>

<nav>

<a href="{{ url\_for('hello') }}">FlaskApp</a>

<a href="{{ url\_for('comments') }}">Comments</a>

<a href="{{ url\_for('about') }}">About</a>

</nav>

<hr>

<div class="content">

{% block content %} {% endblock %}

</div>

</body>

</html>

Здесь мы используем помощник url\_for() для ссылки на функцию просмотра comments(). Теперь на панели навигации будет новая ссылка, которая ведет на страницу комментариев.

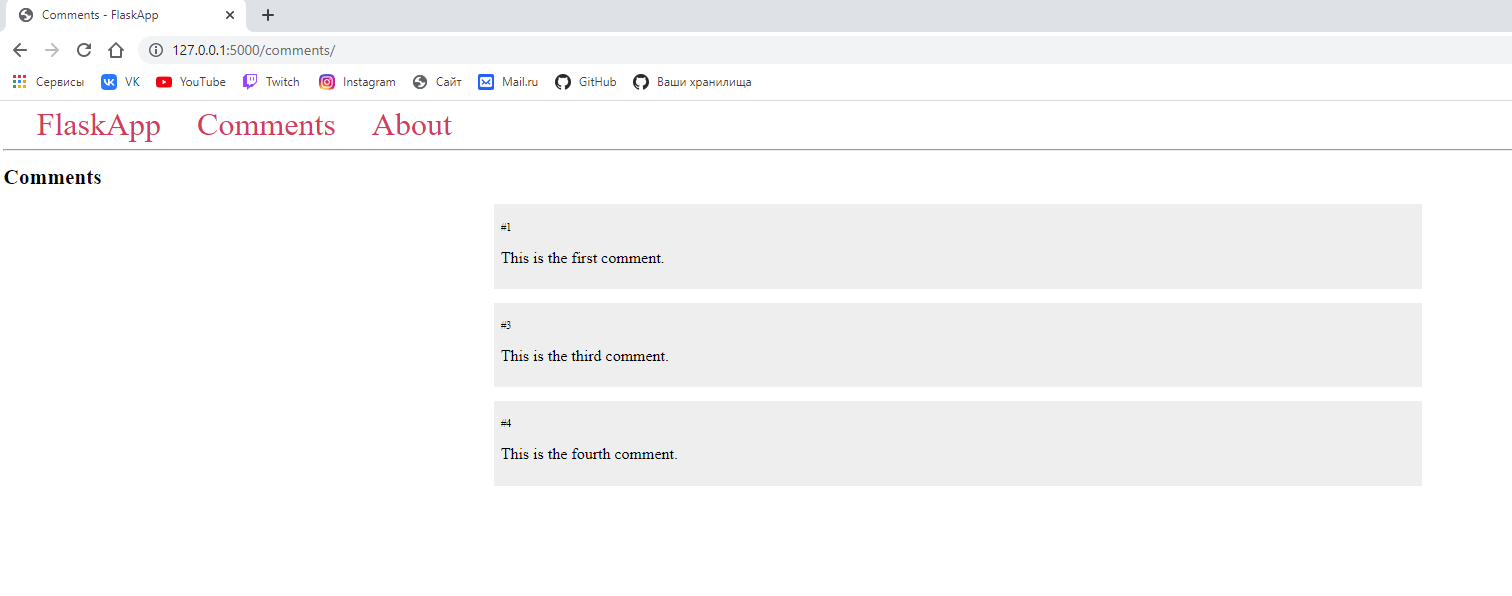


Рисунок 14 – Ссылка на комментарии

## 5. Использование фильтров

## На этом этапе мы узнаем, как использовать фильтры Jinja в своих шаблонах.

Сначала мы преобразуем комментарии на странице комментариев в верхний регистр. Откроем для редактирования шаблон **comments.html** и отредактируем его, чтобы он выглядел следующим образом:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% **for** comment **in** comments %}

{% **if** loop.index != 2 %}

<div style="padding: 10px;

background-color: #EEE;

margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment | upper }}</p>

</div>

{% endif %}

{% endfor %}

</div>

{% endblock %}

Здесь мы расширяем шаблон **base.html** и заменяем содержимое блока содержимого. Во-первых, мы используем заголовок <h1>, который также служит заголовком страницы.

Мы используем цикл Jinja for в строке {% for comment in comments %}, чтобы просмотреть каждый комментарий в списке комментариев (который сохраняется в переменной комментария). Комментарий отображается в теге <p style = "font-size: 24px"> {{comment}} </p> так же, как мы обычно отображаем переменную в Jinja. Мы сигнализируем о завершении цикла for с помощью ключевого слова {% endfor %}. Это отличается от способа построения циклов for в Python, поскольку в шаблонах Jinja нет специального отступа.

Когда сервер разработки запущен, откроем браузер и перейдем на страницу комментариев:

http://127.0.0.1:5000/comments

Фильтры также могут принимать аргументы в круглых скобках. Чтобы продемонстрировать это, воспользуемся фильтром объединения, чтобы объединить все комментарии в списке комментариев.

Откроем шаблон комментариев и отредактируем его, чтобы он выглядел следующим образом:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% **for** comment **in** comments %}

{% **if** loop.index != 2 %}

<div style="padding: 10px;

background-color: #EEE;

margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment | upper }}</p>

</div>

{% endif %}

{% endfor %}

<hr>

<div>

<p>{{ comments | join(" | ") }}</p>

</div>

</div>

{% endblock %}

Здесь мы добавили теги <hr> и <div>, где мы объединяем все комментарии в списке комментариев с помощью фильтра join(). Обновим страницу комментариев, и мы увидим страницу, подобную следующей:

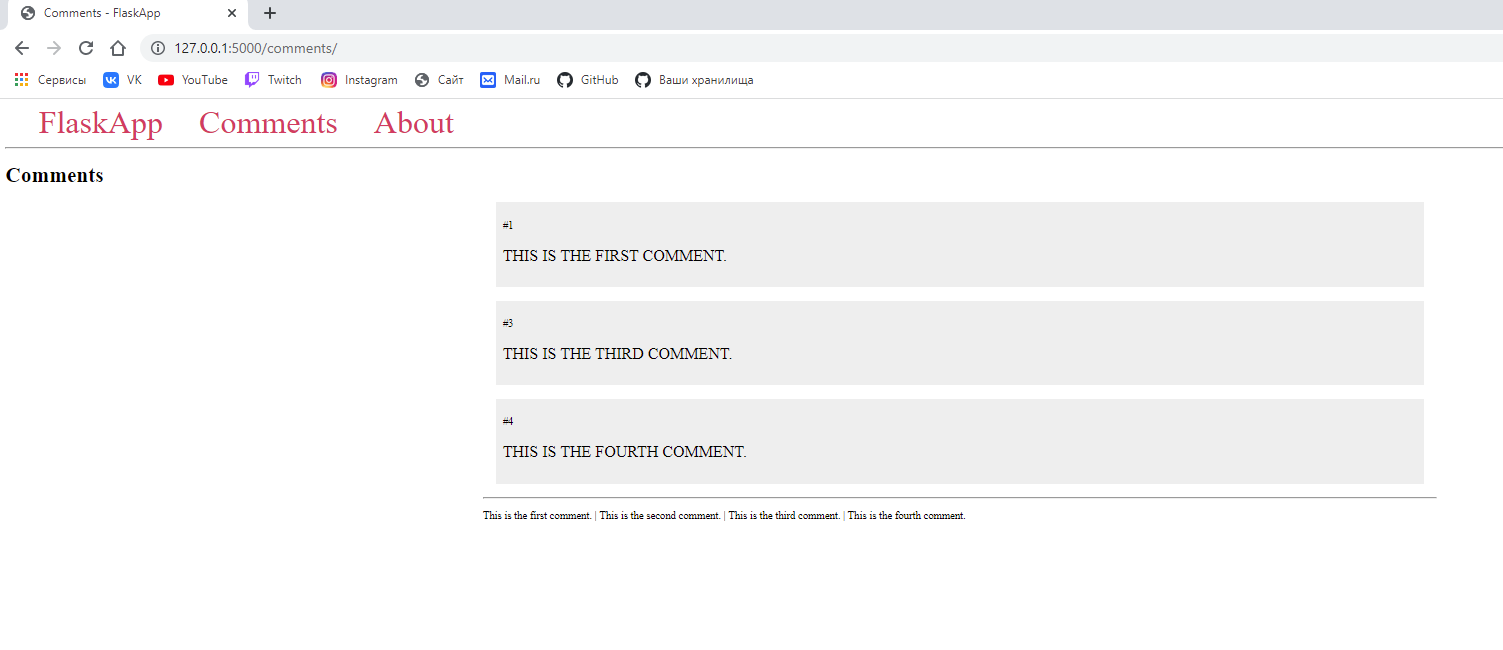


Рисунок 15 – Результат с использованием фильтров

## Как видите, список comments отображается с комментариями, разделенными вертикальной чертой, которую мы передали фильтру join().

## Другой важный фильтр — это безопасный фильтр, который позволяет отображать доверенный HTML‑код в браузере.  Для его реализации откроем шаблон комментариев и отредактируем его, чтобы он выглядел следующим образом:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% for comment in comments %}

{% if loop.index != 2 %}

<div style="padding: 10px;

background-color: #EEE;

margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment | upper }}</p>

</div>

{% endif %}

{% endfor %}

<hr>

<div>

{{ "<h1>COMMENTS</h1>" }}

<p>{{ comments | join(" <hr> ") }}</p>

</div>

</div>

{% endblock %}

Здесь мы добавили значение «<h1>COMMENTS</h1>» и изменили аргумент соединения на тег <hr>. Обновим страницу комментариев, и мы увидите страницу, подобную следующей:

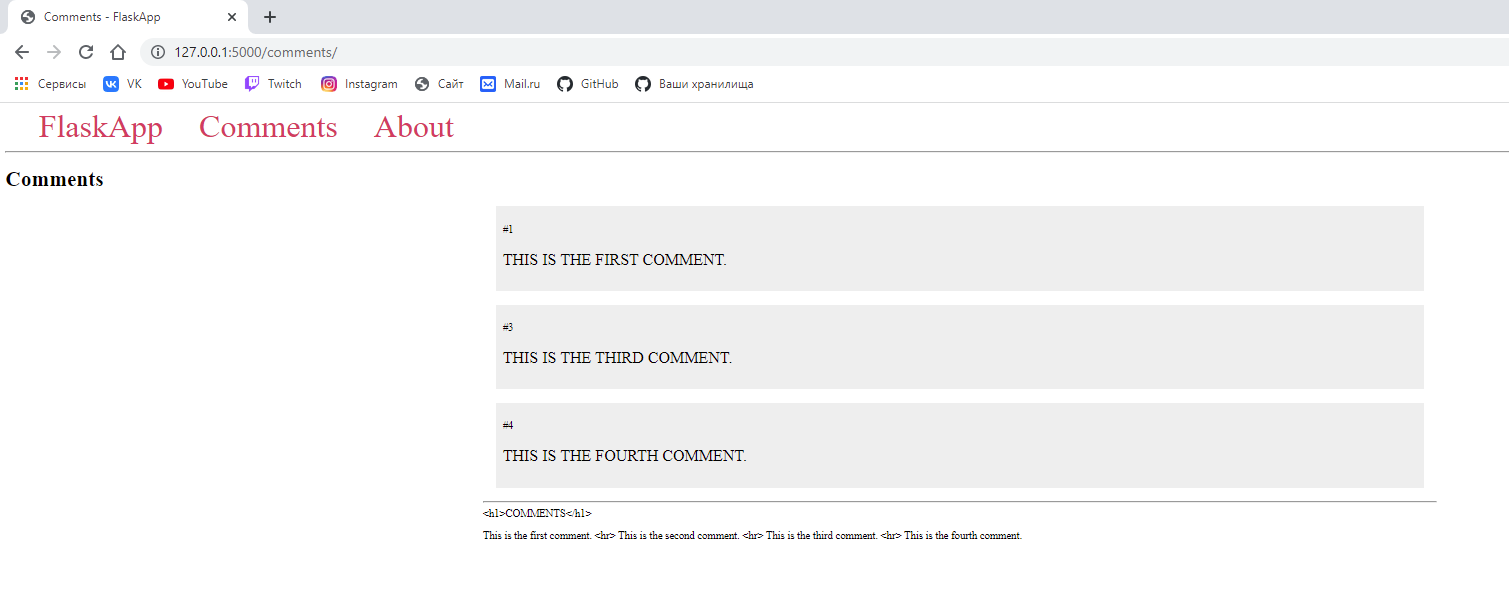


Рисунок 16 – Безопасный фильтр

Чтобы отобразить HTML‑теги выше, откроем файл шаблона комментариев и добавим безопасный фильтр:

{% extends 'base.html' %}

{% block content %}

<h1>{% block title %} Comments {% endblock %}</h1>

<div style="width: 50%; margin: auto">

{% **for** comment **in** comments %}

{% **if** loop.index != 2 %}

<div style="padding: 10px;

background-color: #EEE;

margin: 20px">

<p>#{{ loop.index }}</p>

<p style="font-size: 24px">{{ comment | upper }}</p>

</div>

{% endif %}

{% endfor %}

<hr>

<div>

{{ "<h1>COMMENTS</h1>" | safe }}

<p>{{ comments | join(" <hr> ") | safe }}</p>

</div>

</div>

{% endblock %}

Мы можем видеть, что мы также можем объединить фильтры в цепочку, как в строке <p>{{ comments | join("<hr>") | safe }}<hr>. Каждый фильтр применяется к результату предыдущей фильтрации. Обновим страницу комментариев, и мы увидим, что теги HTML теперь отображаются должным образом:

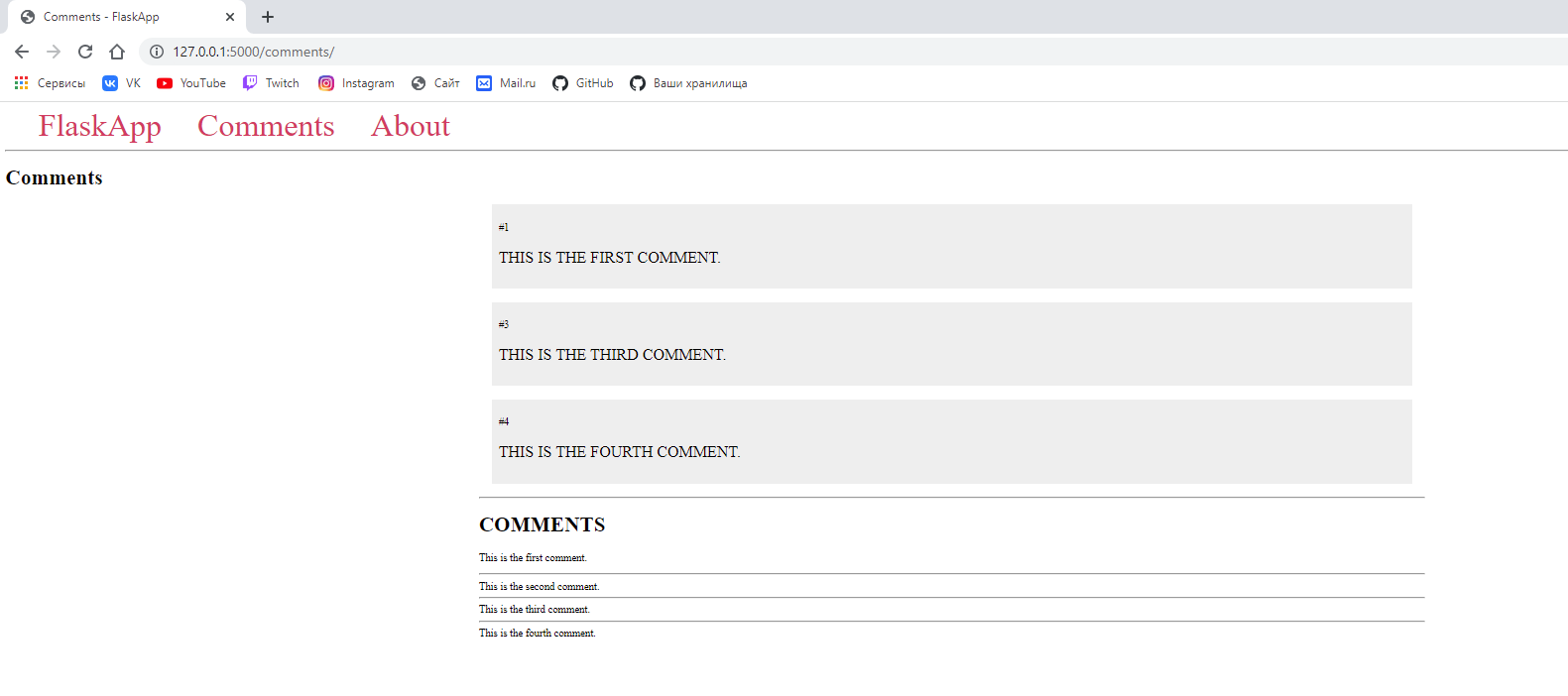


Рисунок 17 – Безопасный фильтр с отображенными тегами

## 6 — интеграция Bootstrap

## Сейчас мы узнаем, как использовать набор инструментов Bootstrap для стилизации нашего приложения. Добавим панель навигации Bootstrap в базовый шаблон, которая будет отображаться на всех страницах, унаследованных от базового шаблона.

Чтобы использовать Bootstrap, нужно добавить его в базовый шаблон, чтобы мы могли использовать его во всех других шаблонах. Откроем свой шаблон **base.html** для редактирования и запишем:

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- Bootstrap CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0p/We" crossorigin="anonymous">

<title>{% block title %} {% endblock %} - FlaskApp</title>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<div class="container-fluid">

<a class="navbar-brand" href="{{ url\_for('hello') }}">FlaskApp</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link" href="{{ url\_for('comments') }}">Comments</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{{ url\_for('about') }}">About</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container">

{% block content %} {% endblock %}

</div>

<!-- Optional JavaScript -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVVFIhvj" crossorigin="anonymous"></script>

</body>

</html>

Большая часть приведенного выше кода — это шаблон Bootstrap, необходимый для его использования. У нас есть несколько метатегов, ссылка на файл CSS Bootstrap в разделе <head>, а внизу у нас есть ссылка на необязательный JavaScript. Выделенные части кода содержат код Jinja, объясненный в предыдущих шагах.  
Обратите внимание, как мы используем определенные теги и классы CSS, чтобы указать Bootstrap, как отображать каждый элемент.

В теге <nav> выше у нас есть тег <a> с классом navbar-brand, который определяет ссылку бренда на панели навигации. Внутри тега <ul class = "navbar-nav"> у нас есть обычные элементы панели навигации внутри тега <a> в теге <li>.

Когда сервер разработки запущен, откроем страницу индекса в своем браузере:

http://127.0.0.1:5000/

Мы увидим страницу, похожую на следующую:

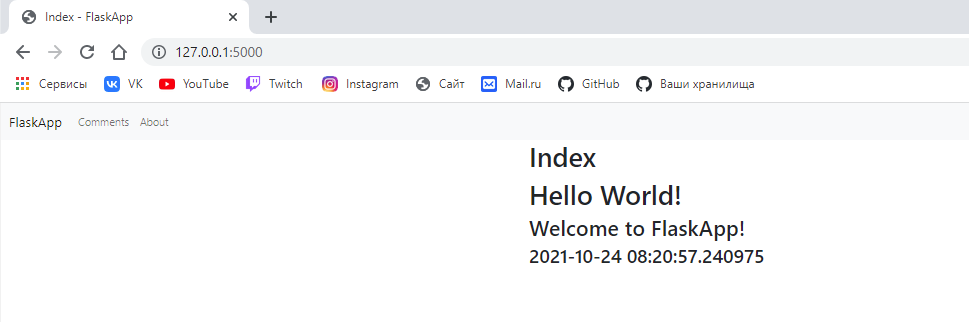


Рисунок 18 – Результат интеграции Bootstrap

## Теперь мы знаем, как использовать HTML‑шаблоны в своем веб-приложении Flask. Мы использовали переменные для передачи данных с сервера в шаблоны, чтобы избежать повторения HTML‑кода мы использовали наследование шаблонов, встроили такие элементы, как условные выражения if и циклы for, а также ссылки между разными страницами. Мы узнали о фильтрах для изменения текста и отображения надежного HTML, а также интегрировали Bootstrap в свое приложение.