Титульный лист

МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение Московской области «Ногинский колледж»

Курсовой проект

по МДК.09.01 Проектирование и разработка веб-приложений

ПМ.09 Проектирование, разработка и оптимизация веб-приложений

Тема:

ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА FLASK В РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЯ «БРОНИРОВАНИЕ АВИАБИЛЕТОВ»

Разработчик: студент группы 3ИСПР2

Алимов Д.Ю.

(подпись)

Оценка защиты курсового проекта

« »

Дата защиты

« » 2024

Руководитель проекта:

преподаватель Степанов С.О.

(подпись)

Ногинск, 2024

Содержание

Оглавление

[ВВЕДЕНИЕ 3](#_Toc169471268)

[1. Основная часть 4](#_Toc169471269)

[1.1Flask: Знакомство и возможности области фреймворка 4](#_Toc169471270)

[1.2 Установка Flask 4](#_Toc169471271)

[2. База данных и форма регистрации/авторизации 6](#_Toc169471272)

[2.1 Models.py 7](#_Toc169471273)

[2.2 Create\_db.py: 8](#_Toc169471274)

[2.3 Login.html 8](#_Toc169471275)

[2.4 Register.html 9](#_Toc169471276)

[2.5 Base.html 11](#_Toc169471277)

[2.6 App.py 11](#_Toc169471278)

[3. Запросы GET и POST 12](#_Toc169471279)

[3.1 Register 13](#_Toc169471280)

[3.2 Index 14](#_Toc169471281)

[3.3 Profile 15](#_Toc169471282)

[ЗАКЛЮЧЕНИЕ 16](#_Toc169471283)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 17](#_Toc169471284)

[ПРИЛОЖЕНИЯ 18](#_Toc169471285)

# ВВЕДЕНИЕ

Темой курсового проекта является «Бронирование авиабилетов».

Целью данного проекта было изучение возможностей Flask в создании веб-сайта, а также практическое применение полученных знаний в создании полноценного функционального сайта.

Для достижения поставленной цели мне необходимо выполнить следующие задачи:

1. Изучить фреймворк Flask;

2. Установка Flask и специальных расширений;

3. Практическая разработка сайта с учетом недостатков.

Бронирование авиабилетов является одним из наиболее популярных видов онлайн-сервисов в современном мире. С каждым годом количество путешественников, пользующихся услугами онлайн-платформ для покупки билетов, увеличивается. Это обусловлено удобством и быстротой процесса бронирования, а также возможностью сравнения цен и услуг различных авиакомпаний.

Актуальность данной темы обусловлена постоянным ростом числа путешественников и повышением требований к качеству обслуживания

Современное состояние систем бронирования авиабилетов на рынке характеризуется разнообразием предложений от различных интернет-платформ, но не всегда они удовлетворяют все потребности пользователей. Поэтому разработка новой системы, учитывающей все актуальные требования и тенденции рынка, является важной задачей в сфере онлайн-туризма.

Этот фреймворк оказался для меня в новинку, так как я услышал о нем впервые. Он оказался достаточно популярным для современного мира, и большинство проектов таких как: Netflix, Reddit, Trivago, Samsung использовали Flask в качестве инструмента.

# 1. Основная часть

## 1.1Flask: Знакомство и возможности области фреймворка

Моё знакомство с Flask началось с изучение документации. Итак что же такое Flask? Это легковесный веб-фреймворк для языка Python, который предоставляет минимальный набор инструментов для создания веб-приложений. На нём можно сделать и лендинг, и многостраничный сайт с кучей плагинов и сервисов.

У Flask много преимуществ, которые выделяют его среди других фреймворков:

1. Простой синтаксис — это всё-таки Python;
2. Удобные шаблоны — можно быстро создавать прототипы веб-приложений;
3. Большое количество инструментов для гибкой настройки сайтов под любые нужды.

Первое что мне предстояло это подготовка перед установкой фреймворка.

Это новейшая версия Python и pip, чтобы в последствии не возникало ошибок. Моей средой разработки служила программа Visual Studio Code, так как это был более оптимальный и удобный для меня вариант.

1.2 Установка Flask

Я осуществлял установку через терминал Visual Studio Code, перед этим установив специальное расширение для Python, но перед этим нужно проверить актуальность pip и python а после этого устанавливать Flask.

pip –version

python --version

pip install Flask

Он сразу же начал загрузку и через секунду все было установлено. После этого я искал информацию о том что мне может пригодится, это привело меня к установке специальных расширений, таких как:

* Flask – login;
* Flask – admin;
* Flask – SQLAlchemy
* Flask – request;
* Flask – werkzeug.

И пр.

Далее была поставлена задана создать репозиторий в GitHub что я

успешно сделал и уже там я создал проект в котором хранился и дополнялся код.

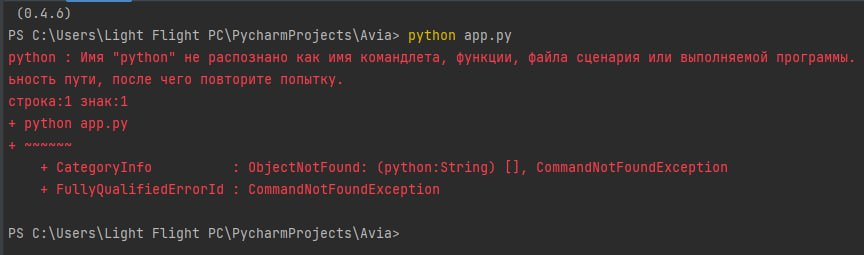
Всё началось с создание папок, были созданы такие папки как:

* app.py – в котором хранились все связи и все установленные расширения;
* models.py – для моделей страниц;
* templates – для html страниц;
* static – для стилей html страниц;
* instance – для хранения БД;
* create\_db.py – для создания БД.

В app.py я подключал расширения чтобы они связывались c остальным кодом и правильно функционировали. Большая часть времени уходила на то чтобы понять, а что именно мне нужно писать? В этом мне помогли видео уроки и теоретические материалы взятые из интернета. Первой моей целью было заставить сайт работать, т.к. было множество ошибок в коде которые я упускал из виду, и на которые потребовалось много времени для их решения:

Первой ошибкой с которой я столкнулся был сам Python. У меня был указан неправильный интерпретатор в настройках компьютера. Мне не составило много проблем решить эту проблему.



Второй ошибкой был pip который конфликтовал с версией Python. 

Третьей ошибкой была неправильная загрузка дополнений Flask из-за который постоянно вылезало оповещение о том, что интерпретатор не подключен. Эта ошибка висела довольно продолжительное время, но с помощью интернета я успешно ее устранил.

Четвертой ошибкой был неправильное подключение кода, из-за этого сайт долго не хотел высвечиваться, и постоянно жаловался на версию pip и Python, хотя в них вообще проблем небыло. Из-за этого ошибка также держалась долго пока я не понял что проблема в коде.

Пятой и последней ошибкой был неправильный синтаксис и некоторые значения кода, где в одном случае я писал user а в другом user, из-за этого код конфликтовал. А я не мог увидеть ошибку.

Наконец-то спустя продолжительное время сайт запустился, но на нем было пусто, так как я ничего еще не сделал.

## 2. База данных и форма регистрации/авторизации

Я использовал базу данных SQLAlchemy, сама БД представляет из себя Python-библиотеку, которая позволяет работать с реляционными базами данных с помощью ORM.

Реляционные базы данных хранят информацию в виде связанных между собой таблиц. К таким базам относят Oracle, Microsoft SQL Server, PostgreSQL и другие. Реляционные базы данных работают по принципам реляционной алгебры, но это тема для отдельной статьи.

Сила SQLAlchemy — в её **ORM**. Расшифровывается как **object relational mapper**, или **«объектно-реляционное отображение»**. ORM позволяет управлять базами данных с помощью методов объектов в коде и при этом не использовать SQL-запросы. На самом деле это очень удобно, так как позволяет писать привычный код, не переключаясь на SQL.

Эта бд показалась мне очень удобной т.к. у нее много хороших особенностей, таких например как то, что мы можем заменить базу данных в любой момент не ломая код. Но с ее подключением также возникло много трудностей. Таких как неправильно поданная информация из интернета которая давала неправильные указания. И мне потребовалось достаточно много времени чтобы понять как работают папки и что в них нужно писать .

Итак чтобы создать функциональную бд мне потребовалось 2 основных файла: models и create\_db.

Файл models отвечал за вид самой БД, что там будет и с какими параметрами.

Файл create\_db отвечал за функцию создания. То есть с каждым запуском из папки models брались данные и автоматически создавали БД. Не надо было ничего прописывать самостоятельно в терминале, это очень удобно. Но чтобы это понять у меня ушло очень много проб и ошибок.

2.1 Models.py:

from flask\_login import UserMixin

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(50), unique=True, nullable=False)

password = db.Column(db.String(500), nullable=True)

email = db.Column(db.String(120), unique=True, nullable=False)

phone = db.Column(db.Integer(), nullable=True)

def \_\_repr\_\_(self):

return f"<users {{self.username}}>"

### 2.2 Create\_db.py:

from flask import Flask

from models import User, db

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///users.db'

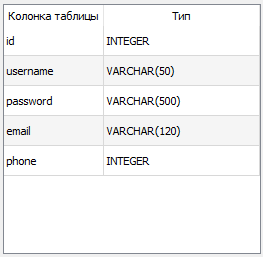
db.init\_app(app)

if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

print('Создана база данных')



Далее мне надо было примерно представить как должен будет выглядеть сайт, поэтому я принялся писать шаблоны для сайтов.

Все началось с формы Регистрации/Авторизации. Я четко знал что там будет имя и номер телефона, поэтому я искал информацию как правильно писать на flask соблюдая синтаксис во избежании ошибок. Flask использует не стандартный синтаксис используя {% %}, тем самым осуществляя подключение которое зависит от base.html. Без него сайт не будет видеть шаблоны и сайт будет выдавать ошибки. Что и случалось у меня пару раз, но я сразу исправил все недочеты. Мой шаблон авторизации выглядел так:

2.3 Login.html

{% extends "base.html" %}

{% block title %}Вход{% endblock %}

{% block content %}

{% if message %}

<p> {{message}} </p>

{% endif %}

<form action = "" method="post">

<p>

<label for="username">Ваше имя: </label>

<input type="text" name="username">

</p>

<p>

<label>Номер телефона: </label>

<input type="text" name="phone" value="" requierd>

</p>

<p>

<label for="password">Пароль: </label>

<input type="password" name="password">

</p>

<p>

<input type="submit">

</p>

</form>

{% endblock %}

Из данного кода мы можем сделать выводы что на сайте логина у нас должны отображаться ячейки имени, телефона и пароля. Довольно стандартное окно логина.

Теперь давайте перейдем к регистрации, она не менее важна тут так как через нее осуществляется занесение данных в базу данных. Вот так он выглядит:

2.4 Register.html

{% extends "base.html" %}

{% block title %}Регистрация{% endblock %}

{% block content %}

<form action="" method="post" class="form-contact">

<p>

<label for="username">Ваше имя: </label>

<input type="text" name="username">

</p>

<p>

<label>Пароль: </label>

<input type="password" name="password" value="" requierd>

</p>

<p>

<label>Email: </label>

<input type="text" name="email" value="" requierd>

</p>

<p>

<label>Номер телефона: </label>

<input type="text" name="phone" value="" requierd>

</p>

<p>

<input type="submit" name="Войти" >

</p>

</form>

{% endblock %}

Как мы можем заметить в обоих фрагментах кода присутствует строчка {% extends "base.html" %}. Именно она осуществляет подключение к base.html, самому важному файлу в папке templates. Именно через этот файл осуществляется подключение к шаблонам html.

Я не буду показывать полностью код base.html так как он очень большой, но я выделю пару строчек и подробно распишу их. Начнем со строчек

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"> - эта строчка кода подключает CSS-файл Bootstrap версии 5.2.3 к моему HTML-документу, что позволяет использовать все стили и компоненты, предоставляемые этим CSS-фреймворком.

2.5 Base.html

<div class="col-md-3 text-end">

<a href="{{ url\_for('login') }}" class="btn btn-outline-primary me-2">Вход</a>

<a href="{{ url\_for('register') }}" class="btn btn-primary">Регистрация</a>

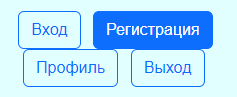
{% if current\_user.is\_authenticated %}

<a href="{{ url\_for('profile') }}" class="btn btn-outline-primary me-2">Профиль</a>

<a href="{{ url\_for('logout') }}" class="btn btn-outline-primary me-2">Выход</a>

{% endif %}

В этом коде мы можем увидеть подключение которое предоставляет доступ к кнопкам навигации, их всего 4 но функционал выполняют как часы. Это был функционал HTML-шаблонов.



Чтобы наш login и register работали нужно было прописать им соответсвующие пути и осуществить подключение к БД. На этом этапе у меня было достаточно много ошибок так как все источники давали разную информацию, приходилось обращаться к видеоурокам которые также были не сильно эффективными. Поэтому я засел на довольно продолжительное время собирая информацию и пробуя различный код. Спустя время я собирая по частичкам понял строение функции и у меня получилась связь которая заработала. Вот так выглядит связь в login:

### 2.6 App.py

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

if user and check\_password\_hash(user.password, password):

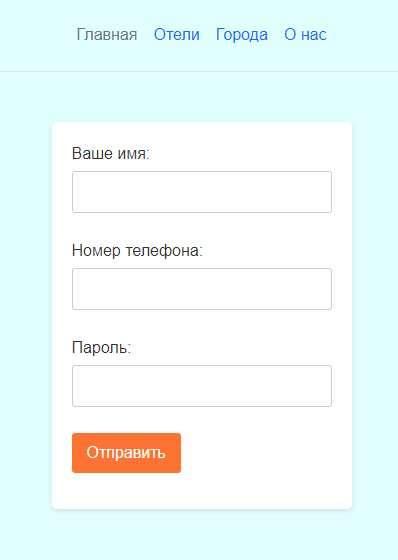
login\_user(user)

return redirect(url\_for('index'))

else:

flash('НЕПРАВИЛЬНО', 'error')

return render\_template('login.html')



Можно довольно быстро разобрать данную часть кода. Но для начала разберем что такое запросы GET и POST.

## 3. Запросы GET и POST

GET — метод для чтения данных с сайта. Например, для доступа к указанной странице. Он говорит серверу, что клиент хочет прочитать указанный документ. На практике этот метод используется чаще всего, например, в интернет-магазинах на странице каталога. Фильтры, которые выбирает пользователь, передаются через метод GET.

POST — метод для отправки данных на сайт. Чаще всего с помощью метода POST передаются

@app.route('/login', methods=['GET', 'POST']) - это декоратор, который определяет маршрут "/login" в веб-приложении. Этот маршрут будет обрабатывать два типа HTTP-запросов: GET и POST.

def login(): - это функция, которая будет вызываться, когда клиент обращается к маршруту "/login". Внутри этой функции будет реализована логика обработки запросов.

if request.method == 'POST': - это условие, которое проверяет, был ли текущий запрос отправлен методом POST. Это означает, что пользователь отправил форму на этот маршрут.

Таким образом, этот код определяет маршрут "/login", который будет обрабатывать как GET-запросы (например, для отображения формы входа), так и POST-запросы (для обработки отправленной формы). Когда пользователь отправляет форму на этот маршрут, код внутри условия if request.method == 'POST': будет выполнен.

### 3.1 Register

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

def register():

if current\_user.is\_authenticated:

return redirect(url\_for('profile'))

if request.method == "POST":

username = request.form['username']

password = request.form['password']

email = request.form['email']

phone = request.form['phone']

user = User(username=username, password=generate\_password\_hash(password), email=email, phone=phone)

try:

db.session.add(user)

db.session.commit()

return redirect('/')

except:

return "Ты где-то Ошибся :)"

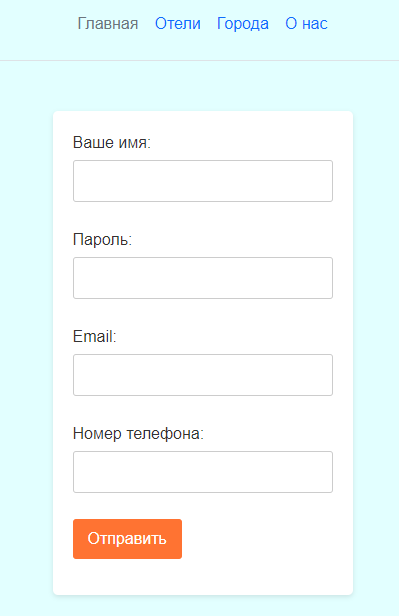
return render\_template ("register.html")

username = request.form['username'], password = request.form['password'], email = request.form['email'], phone = request.form['phone']: - Эти строки извлекают данные, введенные пользователем в форму регистрации.

User = User(username=username, password=generate\_password\_hash(password), email=email, phone=phone): - Создается новый объект User с введенными пользователем данными.

Пароль пользователя хэшируется с помощью generate\_password\_hash() для безопасного хранения.

try: ... except:: - Этот блок пытается добавить нового пользователя в базу данных и сохранить изменения. Если возникает ошибка, будет возвращено сообщение "Ты где-то Ошибся :)".



### 3.2 Index

@app.route('/')

def index():

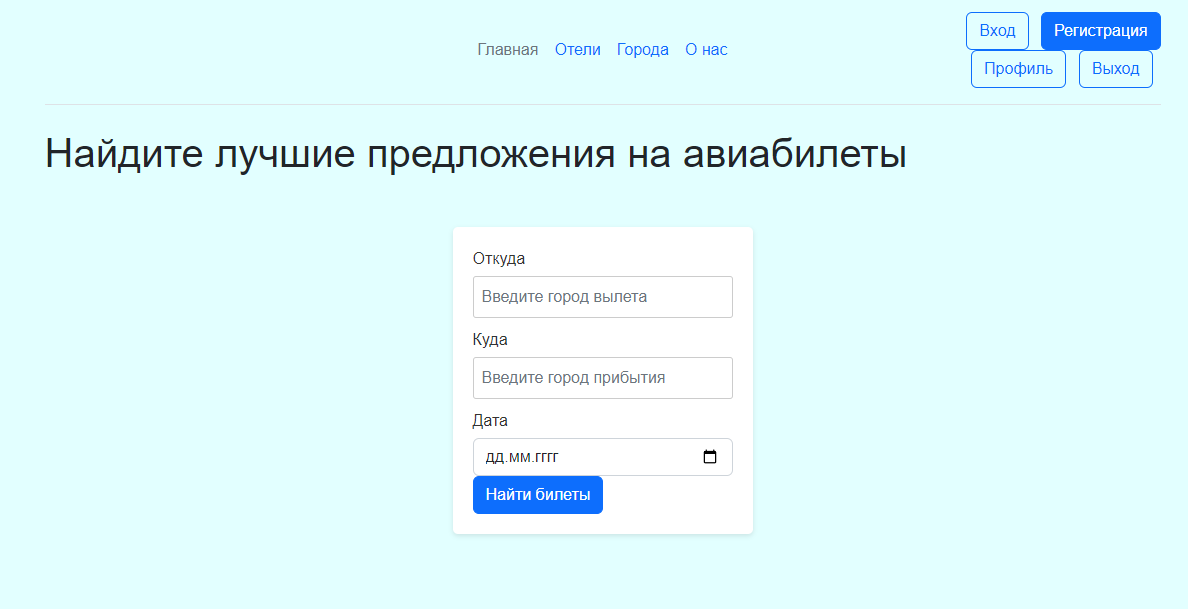
if current\_user.is\_authenticated:

return render\_template('index.html', user=current\_user)

else:

return redirect(url\_for('login'))

Данная часть кода отвечает за связь с главной страницей.



### 3.3 Profile

@app.route('/profile')

def profile():

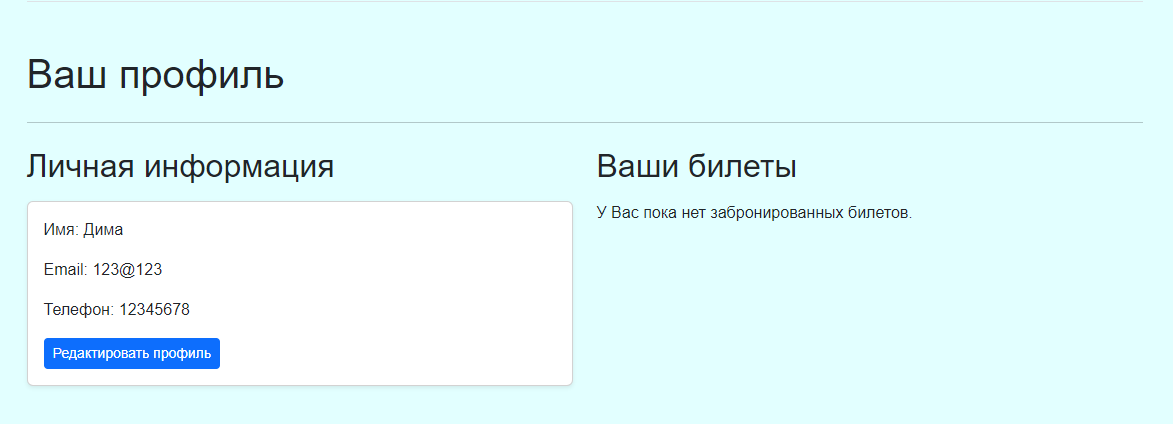
if current\_user.is\_authenticated:

return render\_template('profile.html', user=current\_user)

else:

return redirect(url\_for('login'))

Этот код отвечает за связь с профилем пользователя.



# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

# ПРИЛОЖЕНИЯ