

Министерство науки и высшего образования РФ  
ФГАОУ ВПО  
Национальный исследовательский технологический университет «МИСиС»

---

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

**Проект**  
по дисциплине «Объективно-ориентированное программирование»

Выполнил:  
студент группы БИВТ-24-5

Черных Богдан

Проверил:  
Стучилин В. В.

Москва, 2025

## Содержание

Введение	3
Постановка задачи	4
Описание пользовательских классов	5
Реализация компонентных функций	7
Реализация основной программы	8
Примеры экранных форм и диалогов	9
Примеры результатов работы программы	10
Программная документация	14
Листинг программы	16
Заключение	32

## **Введение**

Проект представляет собой веб-приложение, разработанное на веб-фреймворке Flask, который использовался для создания серверной части на Python. Код построен с использованием ООП, если быть точнее то с использованием методов, что позволяет легко масштабировать и поддерживать приложение. Фронтенд реализован с применением HTML и CSS. Для управления данными используется связка SQLAlchemy и PostgreSQL: первая обеспечивает ORM для работы с бд, а вторая выступает самой базой данных. Для проверки валидности данных используется код на С#, используя библиотеки для работы с PostgreSQL. Реализуется все это через модуль subprocess в Python. В будущем можно будет связать проект с АПИ Spotify, осуществляя более подробные данные о каждой части и автоматически подтягивать например обложку песни, альбома или самого артиста. [12]

# **Постановка задачи**

## **Цель проекта:**

Создать удобное и масштабируемое веб-приложение для оценки и анализа музыкальных произведений, альбомов и исполнителей с возможностью интеграции внешних сервисов (например, API Spotify) для обогащения информации.

## **Задачи проекта:**

- **CRUD-операции для музыкальных данных:** Реализовать добавление, редактирование, удаление и отображение записей о песнях, альбомах и исполнителях через веб-интерфейс.
- **Надёжное хранение и управление данными:** Организовать структуру базы данных на PostgreSQL с использованием ORM SQLAlchemy для эффективного доступа к данным.
- **Бизнес-логика и аналитика:** Разработать методы для вычисления характеристик треков и альбомов (например, общее время прослушивания, определение хита, расчёт рейтинга) с применением принципов ООП.
- **Валидация данных:** Интегрировать проверку корректности вводимых данных с помощью валидаторов, написанных на C# и запускаемых из Python через модуль subprocess.
- **Интеграция и расширяемость:** Создать гибкую архитектуру, предусматривающую возможность дальнейшей интеграции с внешними сервисами (например, с API Spotify).

## Описание пользовательских классов

В проекте реализованы основные классы для работы с данными, представляющими ключевые сущности приложения:

### Класс Song

- **Компонентные данные:**

- **id:** Уникальный идентификатор песни.
- **title:** Название песни.
- **streams:** Количество прослушиваний.
- **artist:** Исполнитель.
- **album:** Альбом, к которому принадлежит песня.
- **date:** Дата релиза.
- **vibe, beat, flow, re\_listening, energy, overall:** Оценочные

параметры.

- **country:** Страна происхождения.

- **Компонентные функции:**

- **get\_total\_play\_time(avg\_duration=3):** Вычисляет общую продолжительность прослушивания.

- **is\_hit():** Определяет, является ли песня хитом, исходя из оценок.
- **get\_song\_summary():** Формирует краткую сводку о треке.

### Класс Artist

- **Компонентные данные:**

- **id:** Уникальный идентификатор исполнителя.
- **artist:** Имя исполнителя.
- **nickname:** Псевдоним.
- **streams\_per\_month:** Прослушивания в месяц.
- **year\_was\_born:** Год рождения.
- **potential:** Потенциал артиста.
- **favourite\_songs:** Любимые песни (строкой с разделителями).
- **best\_album:** Лучший альбом.

- **vibe, annoyance, overall:** Оценочные параметры.
- **country:** Страна.
- **Компонентные функции:**
  - **get\_popularity\_status():** Определяет статус популярности.
  - **get\_top\_songs():** Возвращает топ-5 любимых песен.
  - **recommend\_similar\_artists():** Рекомендует похожих исполнителей.

## Класс Album

- **Компонентные данные:**
  - **id:** Уникальный идентификатор альбома.
  - **album\_title:** Название альбома.
  - **tracks\_num:** Количество треков.
  - **release\_date:** Дата релиза.
  - **likes\_percent:** Процент лайков.
  - **best\_songs:** Список лучших песен.
  - **views\_genius:** Количество просмотров или другая метрика.
  - **styles:** Жанры/стили.
  - **wanna\_relisten:** Вероятность повторного прослушивания.
  - **overall:** Общая оценка альбома.
  - **country:** Страна.
- **Компонентные функции:**
  - **calculate\_rating():** Вычисляет рейтинг альбома на основе нескольких параметров.
  - **recommend\_similar\_albums():** Рекомендует похожие альбомы.
  - **get\_top\_tracks(top\_n=5):** Выбирает топ-треки.

## **Реализация компонентных функций**

Компонентные функции классов обеспечивают:

- **Анализ и агрегирование данных:** Например, расчет общего времени прослушивания песни с учетом среднего времени проигрывания.
- **Определение характеристик:** Функция `is_hit()` в классе `Song` анализирует оценки для определения хита.
- **Сравнение и рекомендации:** Функции типа `recommend_similar_artists()` и `recommend_similar_albums()` осуществляют выборку данных из базы для формирования рекомендаций.

Эти функции реализованы с использованием объектно-ориентированного подхода, что облегчает поддержку и расширение проекта.

## Реализация основной программы

Основной файл приложения (app.py) написан на Python с использованием фреймворка Flask. Его основные задачи:

- **Настройка подключения к базе данных:**

Использование SQLAlchemy для подключения к PostgreSQL через строку подключения:

```
app.config['SQLALCHEMY_DATABASE_URI'] =  
'postgresql://postgres:1234@localhost/musicspoti'
```

- **Маршрутизация:**

Определены маршруты для работы с сущностями (песни, артисты, альбомы) – добавление, редактирование, удаление, отображение, подробный просмотр.

- **Интеграция валидации:**

Через модуль subprocess запускается код на C# (validators.cs), который проверяет корректность данных.

- **Запуск приложения:**

Приложение запускается на локальном сервере (localhost) с возможностью отладки.



## Примеры экранных форм и диалогов

Приложение содержит HTML-шаблоны, используемые для взаимодействия с пользователем. Примеры экранных форм:

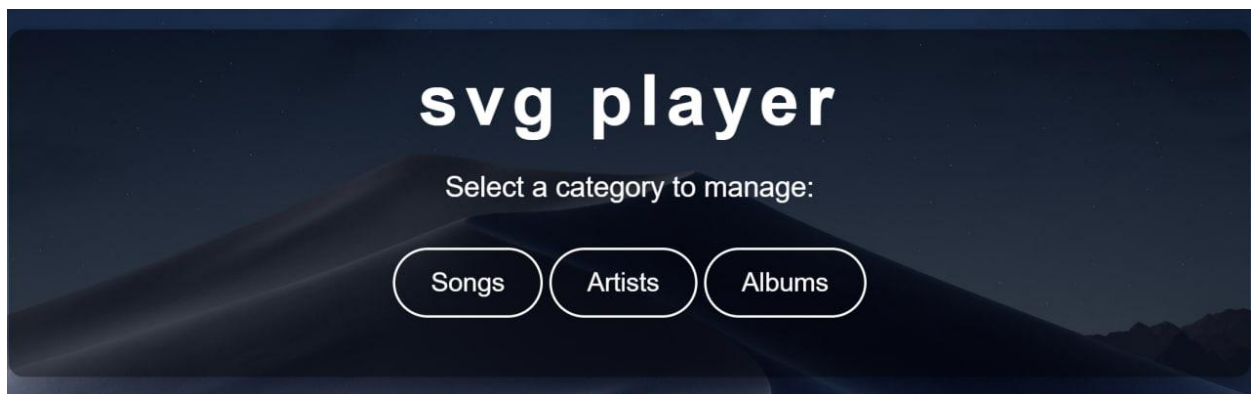
- **Главная страница (index.html):** Стартовая страница с навигацией.
- **Формы добавления:**
  - adder.html (общая форма для добавления новых данных),
  - addartist.html (форма для добавления исполнителя),
  - addalbums.html (форма для добавления альбома).
- **Формы редактирования:**
  - editor.html (общий редактор для песен),
  - editartist.html (редактирование исполнителя),
  - editalbum.html (редактирование альбома),
  - edit\_song\_details.html (редактирование деталей песни).
- **Страницы деталей:**
  - song\_details.html, artist\_details.html, album\_details.html – для отображения подробной информации об объектах.

## Примеры результатов работы программы

При запуске проекта пользователь может:

- Просматривать списки песен, альбомов и исполнителей с возможностью фильтрации по стране и сортировки по оценкам.
- Добавлять новые записи через удобные формы.
- Редактировать и удалять записи, после чего изменения автоматически сохраняются в базе данных.
- Получать рекомендации (например, похожие артисты или альбомы) и расчёты (например, рейтинг альбома, общее время прослушивания песни).


**Примеры:**



# Songs

Add New Song

Filter by Country: [All](#) [UK](#) [USA](#) [BRA](#) [RUS](#) [SPA](#) [EU](#)

Title	Artist	Album	Date	Streams	Vibe	Beat	Flow	Re-Listening	Energy	Overall 
<a href="#">Gata Only</a>	FloyyMenor ft. Cris Mj	no	02.02.2024	1269.74	10.0	10.0	8.0	10.0	9.0	10.0

Back to Home

## Add New Song

Song (Title):

Artist:

Album:

Date (DD.MM.YYYY):

Overall:

Country


Add Song

Back to Songs

# Artists

Add New Artist

Filter by Country: [All](#) [UK](#) [USA](#) [BRA](#) [RUS](#) [SPA](#) [EU](#)

Artist	Nickname	Year Born	Potential	Favourite Songs	Best Album	Vibe	Annoyance	Streams /Month	Overall 	Country
<a href="#">Yeat</a>	Yeat	2000	9.2	Fuk Tha Clout, Geek high, Already Rich	2 Alive	9.0	7.0	17.4	8.5	USA

## Add New Artist

Artist:

Nickname:

Year Born:

Potential:

# Add New Album

Album Title:

Tracks Num:


Date:

Likes Percent (%):

## Albums

Add New Album

Filter by Country: [All](#) [UK](#) [USA](#) [BRA](#) [RUS](#) [SPA](#) [EU](#)

Album Name	Tracks Num	Date	Likes Percent (%)	Best Songs	Views (Genius, in m)	Styles	Wanna Relisten (of 10)	Overall 	Coun
<a href="#">Dragonborn</a>	23.0	16.11.2018	87.0	dragonborn, coldfront, MILF, gucci kandelaki 2016, hard 2 kill	6.5	evolutional rap, russian trap, bassbosted	9.0	9.7	RU

# Программная документация

## Файловая структура проекта:

- **Папка** **static:**

Содержит статические файлы (CSS и изображения). В папке /images — фоновые изображения, файл styles.css отвечает за стилизацию веб-страниц.

- **Папка** **templates:**

Содержит HTML-шаблоны для рендеринга страниц (например, index.html, songs.html, albums.html, artist\_details.html и т.д.).

- **Основные файлы:**

- **app.py:** Основной файл приложения, где описана логика маршрутизации, подключение к базе данных и вызовы валидации.
- **prototypes.py:** Вспомогательный файл с прототипами функций и классов.
- **validators.cs:** Файл на C#, содержащий классы валидаторов для проверки корректности данных с использованием библиотеки Npgsql.

## Необходимые библиотеки и технологии:

- **Python (версии 3.7+):** Язык программирования для реализации серверной логики.
- **Flask:** Веб-фреймворк для организации маршрутизации и обработки HTTP-запросов.
- **SQLAlchemy:** Библиотека ORM для взаимодействия с PostgreSQL.
- **PostgreSQL:** Система управления базами данных.
- **C# и .NET SDK/Runtime:** Используются для запуска валидаторов (validators.cs) через модуль subprocess.
- **Npgsql:** Библиотека для работы с PostgreSQL в C#.
- **Subprocess (Python):** Для вызова кода на C# из Python.

## Инструкция пользователю для работы с проектом:

1. **Клонирование репозитория:**

git clone https://github.com/svgbogdnn/MusicRater.git

## 2. **Создание и активация виртуального окружения:**

- Для Windows:

```
python -m venv venv
```

```
venv\Scripts\activate
```

- Для macOS/Linux:

```
python3 -m venv venv
```

```
source venv/bin/activate
```

## 3. **Установка зависимостей:**

```
pip install -r requirements.txt
```

(Если requirements.txt отсутствует, установить Flask, SQLAlchemy, psycopg2 или psycopg2-binary)

## 4. **Настройка PostgreSQL:**

- Создать базу данных musicspoti и настроить строку подключения в файле app.py.

## 5. **Проверка наличия .NET и запуск валидаторов:**

- Убедиться, что установлен .NET SDK или .NET Runtime.
- Проверить работу кода C# через команду:

```
dotnet --version
```

## 6. **Запуск приложения:**

- Через команду:

```
flask run
```

или

```
python app.py
```

## 7. **Взаимодействие с приложением:**

- Открыть браузер и перейти по адресу <http://127.0.0.1:5000/>.

## Листинг программы

```
#svg does precious

'''Made on Flask as frame, code with Py, database in Postgre and to connect
with db

was used SQLAlchemy, web on html and half Flask, design on css'''

from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.dialects.postgresql import JSON

# launch
app = Flask(__name__)

# database in PostgreSQL
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://postgres:1234@localhost/musicspoti'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# model for songs
class Song(db.Model):
    __tablename__ = 'songs'

    id = db.Column(db.Integer, primary_key=True)

    title = db.Column(db.String(100), nullable=False)

    streams = db.Column(db.Numeric)

    artist = db.Column(db.String(100), nullable=False)

    album = db.Column(db.String(100))

    date = db.Column(db.String(10))

    vibe = db.Column(db.Numeric)

    beat = db.Column(db.Numeric)
```



```

flow = db.Column(db.Numeric)

re_listening = db.Column(db.Numeric)

energy = db.Column(db.Numeric)

overall = db.Column(db.Numeric)

country = db.Column(db.String(50))

'''O-O-P'''

# method for calculation duration of listening the track

def get_total_play_time(self, avg_duration=3):

    # overall duration in minutes

    return self.streams * avg_duration if self.streams else 0

# method to check hit or not that

def is_hit(self):

    # detect hit on rating

    return self.re_listening > 6 and self.vibe > 7 and self.beat > 7 and
self.flow > 8 and self.overall > 8.5

# main info about track

def get_song_summary(self):

    return f"{self.title} - {self.artist} ({self.date})"

#model for artists

class Artist(db.Model):

    __tablename__ = 'artists'

    id = db.Column(db.Integer, primary_key=True)

    artist = db.Column(db.String(100), nullable=False)

    nickname = db.Column(db.String(100), nullable=False)

    streams_per_month = db.Column(db.Numeric)

    year_was_born = db.Column(db.Integer)

    potential = db.Column(db.Numeric)

    favourite_songs = db.Column(db.String(200), nullable=False)

    best_album = db.Column(db.String(100), nullable=False)

    vibe = db.Column(db.Numeric)

    annoyance = db.Column(db.Numeric)

    overall = db.Column(db.Numeric)

```

```

country = db.Column(db.String(50))

'''O-O-P'''

# method to determine current popularity status
def get_popularity_status(self):
    if self.streams_per_month > 10:
        return "Star"
    elif self.streams_per_month > 2:
        return "Popular"
    elif self.streams_per_month > 0.5:
        return "Rising Star"
    else:
        return "Newcomer"

# method top-5 songs artist
def get_top_songs(self):
    # separation by comma ,
    top_songs = self.favourite_songs.split(",")[:5]
    return top_songs if top_songs else ["No songs available"]

# method to recommend similar artists
def recommend_similar_artists(self):
    similar_artists = Artist.query.filter(
        Artist.vibe.between(self.vibe - 1, self.vibe + 1),
        Artist.id != self.id
    ).all()
    return [artist.artist for artist in similar_artists] or ["No similar artists found"]

#model for albums
class Album(db.Model):
    __tablename__ = 'albums'

```

```

id = db.Column(db.Integer, primary_key=True)

album_title = db.Column(db.String(100), nullable=False)

tracks_num = db.Column(db.Numeric)

release_date = db.Column(db.String(100))

likes_percent = db.Column(db.Numeric)

best_songs = db.Column(db.String(100))

views_genius = db.Column(db.Numeric)

styles = db.Column(db.String(100))

wanna_relisten = db.Column(db.Numeric)

overall = db.Column(db.Numeric)

country = db.Column(db.String(50))

'''O-O-P'''

# method to calculate album rating Based on rating each song

def calculate_rating(self):

    track_ratings = [self.likes_percent, self.wanna_relisten,
self.views_genius]

    valid_ratings = [rating for rating in track_ratings if rating is not
None]

    return sum(valid_ratings) / len(valid_ratings) if valid_ratings else
0

# track_ratings = db.Column(JSON)

# def add_track_ratings(self, track_name, rating):

#     if not self.track_ratings:

#         self.track_ratings = {}

#         self.track_ratings[track_name] = rating

#         db.session.commit()

# def calculate_rating(self):

#     if self.track_ratings:

#         ratings = list(self.track_ratings.values())

#         return sum(ratings) / len(ratings) if ratings else 0

#     return 0

# method that recommend similar albums

def recommend_similar_albums(self):

```

```

        similar_albums = Album.query.filter(

            Album.styles == self.styles,

            Album.id != self.id

        ).all()

        return [album.album_title for album in similar_albums] or ["No
similar albums found"]

# method choosing top-5 tracks

def get_top_tracks(self, top_n=5):

    top_tracks = self.best_songs.split(",")[:top_n]

    return top_tracks if top_tracks else ["No popular tracks available"]

# main page for choosing

@app.route('/')

def index():

    return render_template('index.html')

'''SONGS'''

# routes for songs & filtration by country

@app.route('/songs')

def songs():

    country = request.args.get('country')

    sort_by_overall = request.args.get('sort', 'asc')

    query = Song.query

    if country:

        query = query.filter_by(country=country)

    if sort_by_overall == 'desc':

        query = query.order_by(Song.overall.desc())

    else:

        query = query.order_by(Song.overall.asc())

```

```

songs = query.all()

# Формируем список песен с методами
songs_with_methods = [
    {
        "song": song,
        "summary": song.get_song_summary(),
        "total_play_time": song.get_total_play_time(),
        "is_hit": song.is_hit()
    }
    for song in songs
]

return render_template('songs.html',
    songs_with_methods=songs_with_methods, country=country,
                        sort_by_overall=sort_by_overall)

@app.route('/songs/details/<int:id>', methods=['GET', 'POST'])
def edit_song_details(id):
    song = Song.query.get_or_404(id)

    if request.method == 'POST':
        song.duration_minutes = float(request.form['duration_minutes'])
        song.is_hit = request.form.get('is_hit') == 'on'
        song.main_info = request.form['main_info']

        db.session.commit()

        return redirect(url_for('songs'))

    return render_template('edit_song_details.html', song=song)

```

```

# route for display song details

@app.route('/songs/<int:id>')
def song_details(id):
    song = Song.query.get_or_404(id)

    song_info = {
        'summary': song.get_song_summary(),
        'total_play_time': song.get_total_play_time(),
        'is_hit': song.is_hit()
    }

    return render_template('song_details.html', song=song,
song_info=song_info)

# adder songs

@app.route('/songs/add', methods=('GET', 'POST'))
def add_song():
    if request.method == 'POST':
        title = request.form['title']
        artist = request.form['artist']
        streams = float(request.form['streams'])
        album = request.form['album']
        date = request.form['date']
        vibe = float(request.form['vibe'])
        beat = float(request.form['beat'])
        flow = float(request.form['flow'])
        re_listening = float(request.form['re_listening'])
        energy = float(request.form['energy'])
        overall = float(request.form['overall'])
        country = request.form['country']

        new_song = Song(
            title=title,
            artist=artist,
            streams=streams,

```

```

        album=album,

        date=date,

        vibe=vibe,

        beat=beat,

        flow=flow,

        re_listening=re_listening,

        energy=energy,

        overall=overall,

        country=country
    )

    db.session.add(new_song)

    db.session.commit()

    return redirect(url_for('songs'))

    return render_template('adder.html')

# edditer for songs

@app.route('/songs/edit/<int:id>', methods=('GET', 'POST'))
def edit_song(id):

    song = Song.query.get_or_404(id)

    if request.method == 'POST':

        song.title = request.form['title']

        song.artist = request.form['artist']

        song.streams = float(request.form['streams'])

        song.album = request.form['album']

        song.date = request.form['date']

        song.vibe = float(request.form['vibe'])

        song.beat = float(request.form['beat'])

        song.flow = float(request.form['flow'])

        song.re_listening = float(request.form['re_listening'])

        song.energy = float(request.form['energy'])

```

```

        song.overall = float(request.form['overall'])

        song.country = request.form['country']

        db.session.commit()

        return redirect(url_for('songs'))

    return render_template('editor.html', song=song)

# deleter for songs
@app.route('/songs/delete/<int:id>')
def delete_song(id):
    song = Song.query.get_or_404(id)
    db.session.delete(song)
    db.session.commit()
    return redirect(url_for('songs'))

'''ARTISTS'''

# routes for artists & filtration by country
@app.route('/artists')
def artists():
    country = request.args.get('country')
    sort_by_overall = request.args.get('sort', 'asc')

    query = Artist.query
    if country:
        query = query.filter_by(country=country)
    if sort_by_overall == 'desc':
        query = query.order_by(Artist.overall.desc())
    else:
        query = query.order_by(Artist.overall.asc())

```



```

artists = query.all()

artists_with_methods = [
    {
        "artist": artist,
        "popularity_status": artist.get_popularity_status(),
        "top_songs": artist.get_top_songs(),
        "similar_artists": artist.recommend_similar_artists()
    }
    for artist in artists
]

return render_template('artists.html',
artists_with_methods=artists_with_methods, country=country,
                        sort_by_overall=sort_by_overall)

# route to display artist details
@app.route('/artists/<int:id>')
def artist_details(id):
    artist = Artist.query.get_or_404(id)

    artist_info = {
        'popularity_status': artist.get_popularity_status(),
        'top_songs': artist.get_top_songs(),
        'similar_artists': artist.recommend_similar_artists()
    }

    return render_template('artist_details.html', artist=artist,
artist_info=artist_info)

# adder for artist
@app.route('/artists/add', methods=('GET', 'POST'))
def add_artist():
    if request.method == 'POST':
        artist = request.form['artist']
        nickname = request.form['nickname']

```

```

streams_per_month = float(request.form['streams_per_month'])

year_was_born = int(request.form['year_was_born'])

potential = float(request.form['potential'])

favourite_songs = request.form['favourite_songs']

best_album = request.form['best_album']

vibe = float(request.form['vibe'])

annoyance = float(request.form['annoyance'])

overall = float(request.form['overall'])

country = request.form['country']


new_artist = Artist(

    artist=artist,

    nickname=nickname,

    streams_per_month=streams_per_month,

    year_was_born=year_was_born,

    potential=potential,

    favourite_songs=favourite_songs,

    best_album=best_album,

    vibe=vibe,

    annoyance=annoyance,

    overall=overall,

    country=country

)


db.session.add(new_artist)

db.session.commit()

return redirect(url_for('artists'))

return render_template('addartist.html')


# edditor for artist

@app.route('/artists/edit/<int:id>', methods=('GET', 'POST'))

def edit_artist(id):

```

```

artist = Artist.query.get_or_404(id)

if request.method == 'POST':
    artist.artist = request.form['artist']
    artist.nickname = request.form['nickname']
    artist.streams_per_month = float(request.form['streams_per_month'])
    artist.year_was_born = int(request.form['year_was_born'])
    artist.potential = float(request.form['potential'])
    artist.favourite_songs = request.form['favourite_songs']
    artist.best_album = request.form['best_album']
    artist.vibe = float(request.form['vibe'])
    artist.annoyance = float(request.form['annoyance'])
    artist.overall = float(request.form['overall'])
    artist.country = request.form['country']

    db.session.commit()

    return redirect(url_for('artists'))

return render_template('editartist.html', artist=artist)

# deleter for artist
@app.route('/artists/delete/<int:id>')
def delete_artist(id):
    artist = Artist.query.get_or_404(id)
    db.session.delete(artist)
    db.session.commit()
    return redirect(url_for('artists'))

'''ALBUMS'''

# routes for albums & filtration by country
@app.route('/albums')
def albums():

```

```

country = request.args.get('country')

sort_by_overall = request.args.get('sort', 'asc')

query = Album.query

if country:
    query = query.filter_by(country=country)

if sort_by_overall == 'desc':
    query = query.order_by(Album.overall.desc())
else:
    query = query.order_by(Album.overall.asc())

albums = query.all()

return render_template('albums.html', albums=albums, country=country,
sort_by_overall=sort_by_overall)

# route for album details
@app.route('/albums/<int:id>')
def album_details(id):
    album = Album.query.get_or_404(id)

    album_info = {
        'rating': album.calculate_rating(),
        'similar_albums': album.recommend_similar_albums(),
        'top_tracks': album.get_top_tracks()
    }

    return render_template('album_details.html', album=album,
album_info=album_info)

# adder for album
@app.route('/albums/add', methods=('GET', 'POST'))
def add_album():
    if request.method == 'POST':
        album_title = request.form['album_title']
        tracks_num = float(request.form['tracks_num'])

```

```

release_date = str(request.form['release_date'])

likes_percent = float(request.form['likes_percent'])

best_songs = request.form['best_songs']

views_genius = float(request.form['views_genius'].replace(",", "."))

styles = request.form['styles']

wanna_relisten = float(request.form['wanna_relisten'])

overall = float(request.form['overall'])

country = request.form['country']


new_album = Album(
    album_title=album_title,
    tracks_num=tracks_num,
    release_date=release_date,
    likes_percent=likes_percent,
    best_songs=best_songs,
    views_genius=views_genius,
    styles=styles,
    wanna_relisten=wanna_relisten,
    overall=overall,
    country=country
)

db.session.add(new_album)

db.session.commit()

return redirect(url_for('albums'))

return render_template('addalbums.html')

# editor for album

@app.route('/albums/edit/<int:id>', methods=('GET', 'POST'))
def edit_album(id):
    album = Album.query.get_or_404(id)

```

```

if request.method == 'POST':

    album.album_title = request.form['album_title']

    album.tracks_num = float(request.form['tracks_num'])

    album.release_date = request.form['release_date']

    album.likes_percent = float(request.form['likes_percent'])

    album.best_songs = request.form['best_songs']

    album.views_genius = float(request.form['views_genius'])

    album.styles = request.form['styles']

    album.wanna_relisten = float(request.form['wanna_relisten'])

    album.overall = float(request.form['overall'])

    album.country = request.form['country']

    db.session.commit()

    return redirect(url_for('albums'))

return render_template('editalbum.html', album=album)

# deleter for album
@app.route('/albums/delete/<int:id>')
def delete_album(id):

    album = Album.query.get_or_404(id)

    db.session.delete(album)

    db.session.commit()

    return redirect(url_for('albums'))

# run app
if __name__ == '__main__':

    with app.app_context():

        db.create_all()

        app.run(debug=True)

# C# checker

```

```

import subprocess

dotnet_path = "dotnet"

program_path = r"C:\Users\Lenovo\SpotifyApp\bin\Debug\net8.0\SpotifyApp.dll"

arguments = ["arg1", "arg2", "arg3"]

try:
    result = subprocess.run(
        [dotnet_path, program_path] + arguments,
        capture_output=True,
        text=True,
        check=True
    )
    print("Output from C# program:")
    print(result.stdout)
    if result.stderr:
        print("Errors from C# program:")
        print(result.stderr)
except subprocess.CalledProcessError as e:
    print(f"Error during execution: {e}")
    print("Output:", e.output)
    print("Error Output:", e.stderr)

#test for git

```

## **Заключение**

Проект представляет собой надежное и масштабируемое веб-приложение для оценки музыкальных произведений, альбомов и исполнителей. Используя Flask, SQLAlchemy и PostgreSQL, система обеспечивает эффективное управление данными, а интеграция с валидаторами на C# гарантирует корректность ввода. Возможность подключения к API Spotify открывает перспективы для дальнейшего расширения функционала, делая проект полезным как для любителей музыки, так и для специалистов по анализу данных.