Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и технологий

**Высшая школа интеллектуальных систем и суперкомпьютерных технологий**

**КУРСОВАЯ РАБОТА**

**Rest Api сервер для базы данных музыкального сервиса**

по дисциплине «Базы данных»

Выполнил студент гр.3530901/70201 Белов Е. А.

Преподаватель Мяснов А. В.

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург 2020

Санкт-Петербургский государственный политехнический университет

**ЗАДАНИЕ НА ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ**

студенту группы 3530901/70201 Белову Егору Александровичу

***1. Срок сдачи законченного проект****а* 13.06.2020

***2. Исходные данные к проекту***: Задание для курсовой работы.

***3. Содержание пояснительной записки*** (перечень подлежащих разработке вопросов): техническое задание, реализация, вывод.

***Дата получения задания***: «10» мая 2020 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_ А. В. Мяснов (подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А.Белов (подпись студента)

\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Оглавление

[Техническое задание 4](#_Toc43423969)

[Постановка задачи 4](#_Toc43423970)

[Возможности сервера 4](#_Toc43423971)

[План разработки 4](#_Toc43423972)

[Ход выполнения работы 5](#_Toc43423973)

[RESTful сервер 5](#_Toc43423974)

[Ссылки 9](#_Toc43423975)

[Вывод 9](#_Toc43423976)

# Техническое задание

## Постановка задачи

Разработать Rest Api сервер, позволяющий пользователям музыкального сервиса прослушивать треки, просматривать исполнителей и музыкальные группы, просматривать награды, для песен, на основе песен составлять плэйлисты. В качестве хранилища данных использовать базу данных, созданную в течение семестра на лабораторных работах.

## Возможности сервера

* Регистрация, аутенфикация пользователей
* Http запросы на выдачу данных о песнях, артистах, группах, альбомах, плейстах
* Возможность пользователями добавлять в избранное понравившиеся альбомы, треки, группы, исполнители.

## План разработки

* Принятие решения о функциональности приложения и его архитектуре;
* Разработка сервиса для взаимодействия с базой данных;
* Выводы о проделанной работе и полученном результате.

# Ход выполнения работы

В качестве базы данных для данного проекта была взята база, разработанная в ходе лабораторных работ. Схема базы данных представлена на Рис.1.

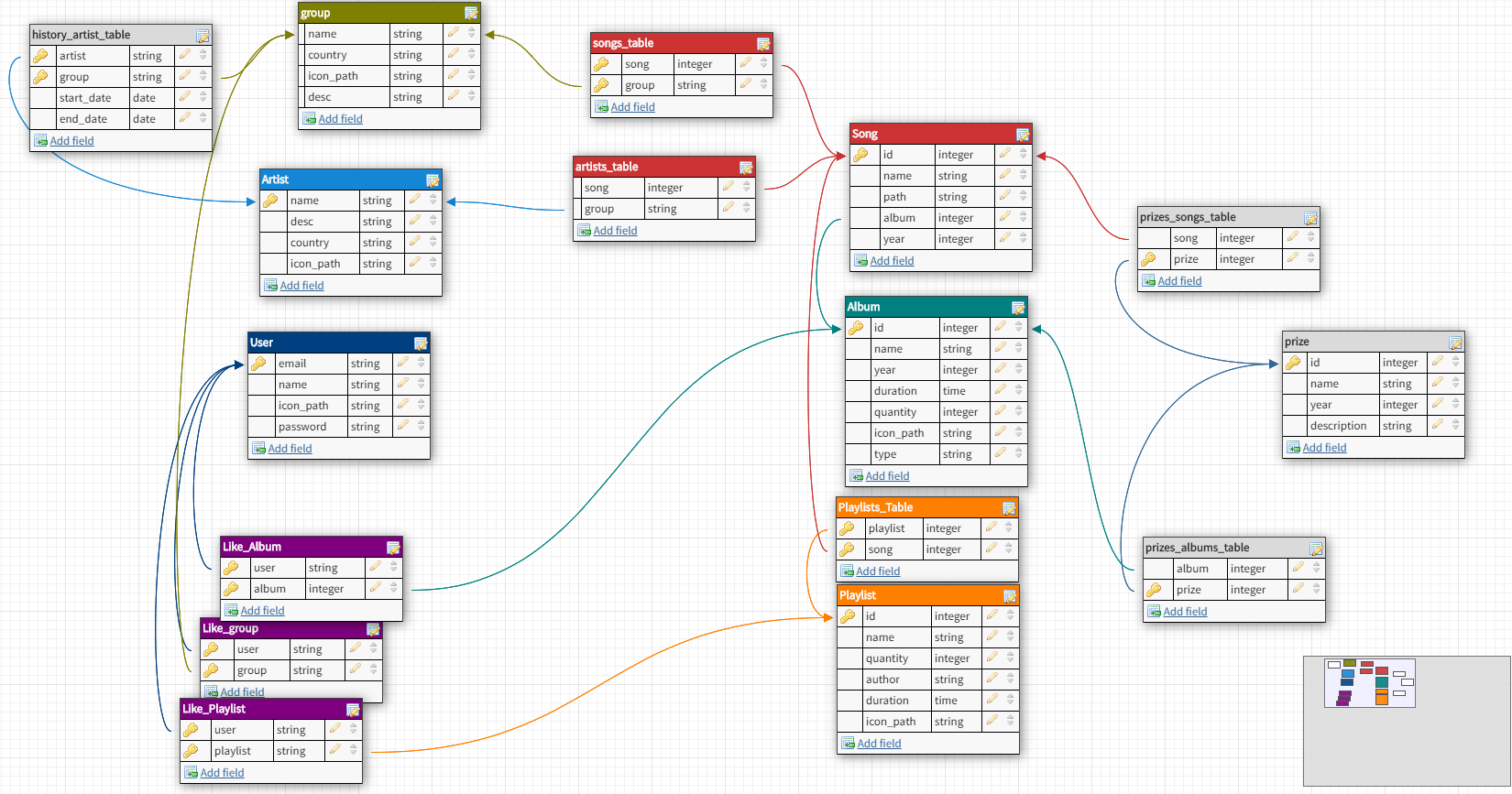


Рис.1. Схема базы данных

Для таблицы с пользователем User были добавлены дополнительные атрибуты для хранения токена token и для хранения время жизни токена token\_expiration. Пароль не храниться в явном виде, а храниться его хэш.

## RESTful сервер

Для разработки сервиса используются фреймворк Flask для языка Python.

REST (Representational State Transfer) определяет набор стандартов для веб-сервисов. API (Application Programming Interface) – интерфейс, с помощью которого различные программы коммуницируют друг с другом. Таким образом, RESTful API – это API, которых соответстует архитектурному стилю REST. REST системы не сохраняют состояния, масштабируются, кэшируются и имеют единый интерфейс Рассматриваемый RESTful API использует HTTP-запросы. Используются методы GET, POST, PUT и DELETE.

Входной точкой сервера является файл run.py. В нем запускается модуль app и настраивается контекст для flask shell. Flask Shell позволяет запустить интерпретатор Python в контексте приложения. В модуле app, в инициализирующем его файле создается и инстанцируется главный модуль Flask, инициализируется такие расширения Flask как LoginManager, OpenId, SqlAlhemy, Migrate. Эти расширения нужны для логирования и идентифицированния пользователей сервиса, работы с базой данных. Migrate позволяет удобно работать и проводить миграции базы данных.

Листинг.1. app/\_\_init\_\_.py

|  |
| --- |
| import os from flask\_login import LoginManager from flask\_openid import OpenID from config import basedir from config import Config from flask import Flask, jsonify, request from flask\_sqlalchemy import SQLAlchemy from flask\_migrate import Migrate  app = Flask(\_\_name\_\_) app.config.from\_object(Config) lm = LoginManager() lm.init\_app(app) oid = OpenID(app, os.path.join(basedir, 'tmp')) db = SQLAlchemy(app) migrate = Migrate(app, db)  from app import models, users, songs, albums, creators |

В файле models.py описаны отношения базы данных, методы преобразования из объектов базы данных в объекты удобного для представления в языке программирования, а также методы обработки данных (проверка паролей, добавление в избранное, выдача данных). Пример отношения пользователь User из файла models.py:

Листинг 2.

|  |
| --- |
| class User(UserMixin, db.Model):  email = db.Column(db.String(255), primary\_key=True)  name = db.Column(db.String(255), nullable=False)  icon\_path = db.Column(db.String(255), nullable=True)  password\_hash = db.Column(db.String(128), nullable=False)  token = db.Column(db.String(32), index=True, unique=True)  token\_expiration = db.Column(db.DateTime)   def \_\_repr\_\_(self):  return '<User {}>'.format(self.name)   def get\_token(self, expires\_in=3600):  now = datetime.utcnow()  if self.token and self.token\_expiration > now + timedelta(seconds=60):  return self.token  self.token = base64.b64encode(os.urandom(24)).decode('utf-8')  self.token\_expiration = now + timedelta(seconds=expires\_in)  db.session.add(self)  return self.token   def revoke\_token(self):  self.token\_expiration = datetime.utcnow() - timedelta(seconds=1)   @staticmethod  def to\_collection\_dict():  resources = User.query.paginate(1, 20, False)  data = {  'users': [item.to\_dict() for item in resources.items]  }  return data   @staticmethod  def check\_token(token):  user = User.query.filter\_by(token=token).first()  if user is None or user.token\_expiration < datetime.utcnow():  return None  return user   def from\_dict(self, data, new\_user=False):  for field in ['email', 'name', 'icon\_path']:  if field in data:  setattr(self, field, data[field])  if new\_user and 'password' in data:  self.set\_password(data['password'])   def to\_dict(self, include\_email=False):  data = {  'email': self.email,  'name': self.name,  'icon\_path': self.icon\_path  }  if include\_email:  data['email'] = self.email  return data   def set\_password(self, password):  self.password\_hash = generate\_password\_hash(password)   def check\_password(self, password):  return check\_password\_hash(self.password\_hash, password) |

В файлу config.py задаются константные значения и значения для конфигурации базы данных и сервиса в целом. Например, переменная SQLALCHEMY\_DATABASE\_URI хранит путь к файлу с базой данных.

Листинг 3.

|  |
| --- |
| import os  basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))   class Config(object):  CSRF\_ENABLED = True  SECRET\_KEY = 'you-will-never-guess'   SQLALCHEMY\_DATABASE\_URI = 'postgresql://postgres:9368@localhost/EgMusic'  SQLALCHEMY\_TRACK\_MODIFICATIONS = False |

Конечные точки для запросов описаны в файлах users.py, creators.py, songs.py, albums.py, auth.py. отвечающий каждый за свое предназначение.

Листинг 4. Albums.py

|  |
| --- |
| from flask import jsonify  from app import app from app.auth import token\_auth from app.models import Album   @app.route('/albums/<int:id>', methods=['GET']) @token\_auth.login\_required def get\_album(id):  return jsonify(Album.query.get\_or\_404(id).to\_dict())   @app.route('/albums', methods=['GET']) @token\_auth.login\_required def get\_albums():  data = Album.to\_collection\_dict()  return jsonify(data) |

В файле errors.py обрабатываются ошибки сервиса и формируется ответы с удобным форматом для чтения ошибок.

Листинг 5. Errors.py

|  |
| --- |
| from flask import jsonify from werkzeug.http import HTTP\_STATUS\_CODES   def error\_response(status\_code, message=None):  payload = {'error': HTTP\_STATUS\_CODES.get(status\_code, 'Unknown error')}  if message:  payload['message'] = message  response = jsonify(payload)  response.status\_code = status\_code  return response   def bad\_request(message):  return error\_response(400, message) |

В каталоге tests лежат файлы с тестами запросов. Файлы имеют формат .htpp. Формат этих файлов поддерживает встроенный в Pycharm http client IntelijIdea. Http клиент позволяет удобно тестировать запросы к серверу. В каталоге migrations лежат нужные файлы для расширения Flask Migrations, которые используются для миграций баз данных.

## Пагинация

Под пагинацией понимают постраничный вывод информации, то есть показ ограниченной части информации на одной (веб)-странице (например, 10 результатов поиска или 20 постов). Она повсеместно используется в веб-приложениях для разбиения большого массива данных на страницы и включает в себя навигационный блок для перехода на другие страницы — пагинатор. Также пагинация используется при так называемой бесконечном скроллинге. Это механизм, позволяющий пользователям просматривать массивный поток контента как единое целое, без видимой финишной черты. Приложение подгружает новую страницу с данными, по мере того как пользователь прокручивает страницу вниз.

**Преимущества**

* Пользователю не нужно ждать пока загрузится большой объем данных, это может занять большое количество времени. За место этого данные выдаются пользователю маленькими блоками, это занимает мало времени и не заставляет пользователя долго ждать.
* Со стороны сервера оптимальнее выдавать данные маленькими порциями.

**Задачи пагинации**

* ускоряет загрузку страницы.
* делает просмотр и поиск нужных элементов проще и удобнее;
* придаёт дизайну сайта аккуратный и законченный вид.
* Иными словами, пагинация позволяет находить нужную информацию на сайте быстро и с комфортом.

## Ссылки

Проект можно посмотреть по адресу:

<http://gitlab.icc.spbstu.ru/Egorius/db_2020>

# Вывод

В ходе разработки данного сервиса были получены навыки по разработке rest api сервисов, навыки формирования http запросов, преобразования данных из базы данных в json формат, навыки работы с базой данных в рамках фреймворка Flask .