VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Title: Multiuser Music Database

Final Project review for
Database management System

Session: 2019-20
Slot: D1+TD1
Proffessor: Dr. Geetha Mary A.

Team:
Abhisar Shukla (18BCE0110)
Yashaswi Shivank (18BCE0162)
Anshul Tripathi (18BCE0148)

Abstract:

In the new world of growing music culture and online music, need for online music streaming services arise as all the songs cannot be stored on the user's mobile device. This requires a service that will store the songs and all the preferences of users and somewhere and should be able to serve user's request just in time. At the core, it requires a database to store all the relative information and therefore a database management system. This project will fulfill those needs by providing a system that will manage all the related data and provide the users with a front end to interact intuitively with the system. We develop a bare-bone basic streaming service for showcasing the database management techniques used.

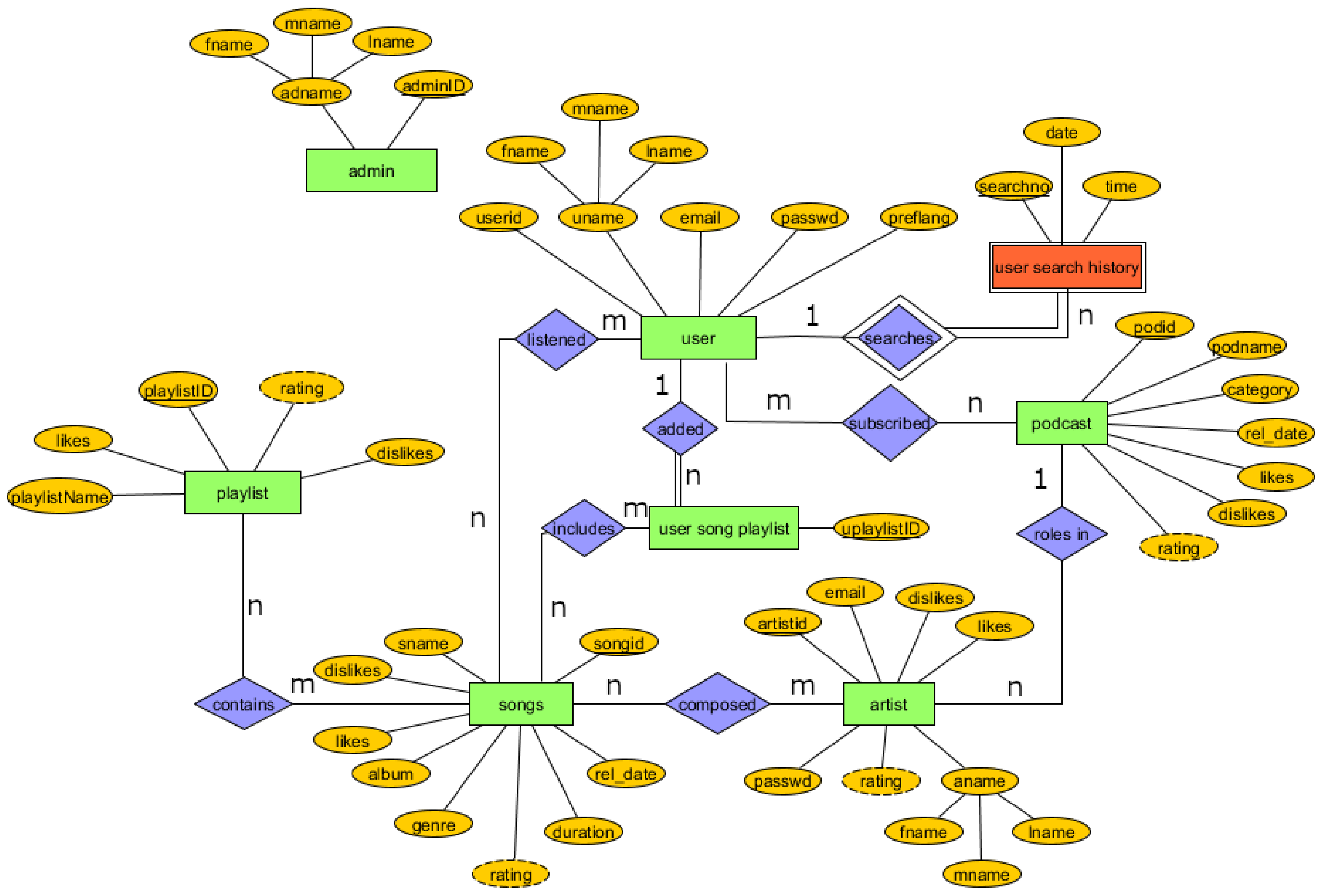
Functional requirements:

User: Can login, listen to songs, make a playlist, rate songs, rate playlists.
Admin: Add songs, remove songs, remove users, change meta data of songs.
Artist: Add songs, remove songs.
Song: Contains details about songs.
Podcast: Contains details about podcast.
Search: user and artists should be able to search the database.
Rating: users can rate the songs, artists, podcasts or playlists they listen to.
History: database contains the user history for quickly finding the songs they listened recently.

Non-Functional requirements:

Processor: i3 core processor
Clock speed: 2.5GHz
RAM: 1GB
Security: Database should be secure so that it cannot be hacked or suffer from any other vulnerability.
Redundancy: Two songs with same meta data cannot exist in the database.
Data Bandwidth: High data bandwidth to support a large user base.
Server: WSGI(Web Server Gateway Interface)
Database: sqlite
Frameworks: flask and jinja2

ERD Diagram:



fname
mname
lname
adname
adminID
admin
mname
fname
lname
date
searchno
time
userid
uname
email
passwd
preflang
user search history
listened
m
user
1
searches
n
podid
podname
category
rel_date
likes
dislikes
rating
podcast
1
added
m
subscribed
n
n
1
roles in
playlistID
rating
likes
dislikes
playlistName
playlist
n
includes
m
user song playlist
uplaylistID
n
n
email
dislikes
artistid
likes
sname
songid
dislikes
m
contains
songs
n
composed
m
artist
n
likes
album
rel_date
genre
duration
rating
passwd
rating
aname
fname
lname
mname

Relational Schema:

Admin:

adminID	fname	mname	lname

User:

userID	fname	mname	nname	email	passwd	preflang

Song:

songID	sname	album	likes	dislikes	relDate	genre	duration	rating

Artist:

artistID	fname	mname	lname	email	passwd	likes	dislikes	rating	podID

Podcast:

podID	podname	category	relDate	likes	dislikes	rating

Playlist:

playlistID	playlistName	rating	like	dislike

Listened:

songID	userID

Contains:

songID	playlistID

Composed:

songID	artistID

Subscribed:

userID	poscastID

User_shistory:

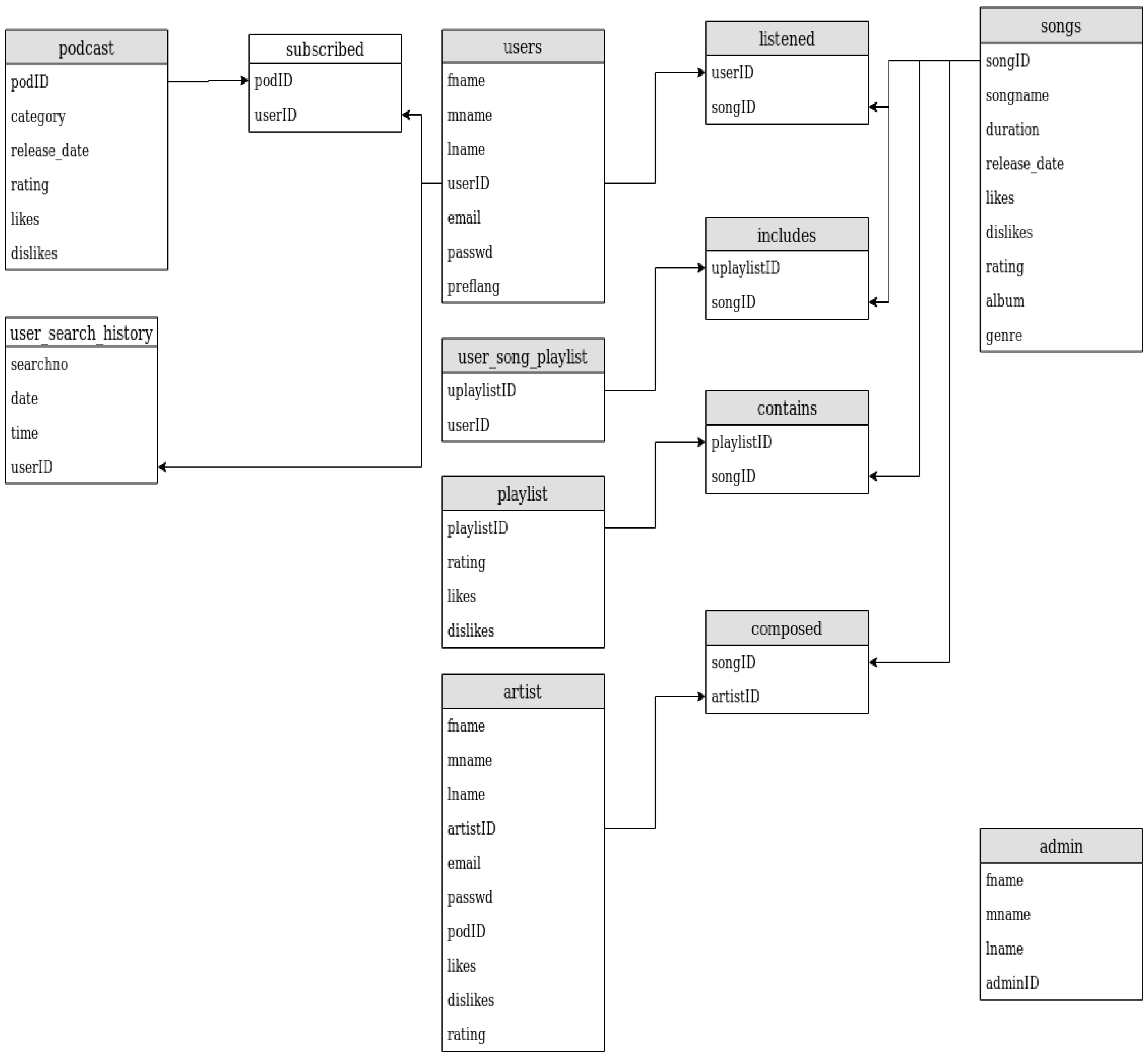
userID	searchNo	date	time

User_splaylist:

uplaylistID	userID

Includes:

uplaylistID	songID


podcast
podID
category
release_date
rating
likes
dislikes
subscribed
podID
userID
users
fname
mname
lname
userID
email
passwd
preflang
listened
userID
songID
songs
songID
songname
duration
release_date
likes
dislikes
rating
album
genre
includes
uplaylistID
songID
user_search_history
searchno
date
time
userID
user_song_playlist
uplaylistID
userID
contains
playlistID
songID
playlist
playlistID
rating
likes
dislikes
composed
songID
artistID
artist
fname
mname
lname
artistID
email
passwd
podID
likes
dislikes
rating
admin
fname
mname
lname
adminID

Sample codes:

DBMS

Database Classes declaration in SQLAlchemy

```
from datetime import datetime
from musicapp import db, login_manager
from flask_login import UserMixin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

listened = db.Table('listened',
            db.Column('user_id' ,db.Integer, db.ForeignKey('user.id')),
            db.Column('sond_id', db.Integer, db.ForeignKey('song.id'))
)

includes = db.Table('includes',
            db.Column('playlist_id', db.Integer, db.ForeignKey('user_song_playlist.id')),
            db.Column('song_id', db.Integer, db.ForeignKey('song.id'))
)

subscribed = db.Table('subscribed',
            db.Column('user_id', db.Integer, db.ForeignKey('user.id')),
            db.Column('podcast_id', db.Integer, db.ForeignKey('podcast.id'))
)

contains = db.Table('contains',
            db.Column('song_id', db.Integer, db.ForeignKey('song.id')),
            db.Column('playlist_id', db.Integer, db.ForeignKey('playlist.id'))
)

composed = db.Table('composed',
            db.Column('artist_id', db.Integer, db.ForeignKey('artist.id')),
            db.Column('song_id', db.Integer, db.ForeignKey('song.id'))
)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    fname = db.Column(db.String(75), nullable=False)
    lname = db.Column(db.String(75), nullable=False)
    username = db.Column(db.String(32), unique=True, nullable=False)
    email = db.Column(db.String(128), unique=True, nullable=False)
    image_file = db.Column(db.String(32), nullable=False, default='default.jpeg')
    #preflang = db.Column(db.String(20), nullable=False)
    password = db.Column(db.String(60), nullable=False)
    searches = db.relationship('user_search_history', backref=db.backref('user'))
    playlists = db.relationship('user_song_playlist', backref=db.backref('user'))
    songs = db.relationship('Song', secondary=listened)
    posts = db.relationship('Post', backref='author', lazy=True)
    podcasts = db.relationship('Podcast', secondary=subscribed)

    def __repr__(self):
        return f"User('{self.username}', '{self.fname}', '{self.lname}', '{self.email}', '{self.image_file}')"

class Podcast(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=True)
    title = db.Column(db.String(75), nullable=False)
    description = db.Column(db.String, default='No description')
    category = db.Column(db.String, nullable=False)
    release_date = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    file_location = db.Column(db.String(20), nullable=True)
    rating = db.Column(db.Integer, default=2)
    likes = db.Column(db.Integer, default=0)
    dislikes = db.Column(db.Integer, default=0)
    artists = db.relationship('Artist', backref=db.backref('podcast'))

    def __repr__(self):
        return f"Podcast('{self.name}', '{self.category}', '{self.release_date}"{self.rating}')"

class Song(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(75), nullable=False)
    title = db.Column(db.String(75), nullable=False)
    description = db.Column(db.String, default='No description')
    #duration = db.Column(db.Float, nullable=False)
    release_date = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    rating = db.Column(db.Integer, default=2)
    likes = db.Column(db.Integer, default=0)
    dislikes = db.Column(db.Integer, default=0)
    file_location = db.Column(db.String(20), nullable=False)
    album = db.Column(db.String(75), nullable=False)
    genre = db.Column(db.String(75), nullable=False)

    def __repr__(self):
        return f"Song('{self.name}', '{self.release_date}', '{self.rating}')"

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

    def __repr__(self):
        return f"Post('{self.title}', '{self.date_posted}')"
class Admin(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    fname = db.Column(db.String(75), nullable=False)
    fname = db.Column(db.String(75), nullable=False)
    lname = db.Column(db.String(75), nullable=False)

    def __repr__(self):
        return f"Admin('{self.fname}', '{self.lname}')"

class Artist(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(32), unique=True, nullable=False)
    email = db.Column(db.String(128), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpeg')
    likes = db.Column(db.Integer, default=0)
    dislikes = db.Column(db.Integer, default=0)
    rating = db.Column(db.Integer, default=2)
    podcast_id = db.Column(db.Integer, db.ForeignKey('podcast.id'))
    compose = db.relationship('Song', secondary=composed, backref=db.backref('composer', lazy='dynamic'))
    def __repr(self):
        return f"Artist('{self.fname}', '{self.lname}', '{self.rating}')"

class Playlist(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    likes = db.Column(db.Integer)
    dislikes = db.Column(db.Integer)
    rating = db.Column(db.Integer, default=2)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    songs = db.relationship('Song', secondary=contains)

    def __repr__(self):
        return f"Playlist('{self.id}', '{self.rating}')"

class user_song_playlist(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    songs = db.relationship('Song', secondary=includes)

class user_search_history(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    datetime = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    searchno = db.Column(db.Integer, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

<u>Forms for getting input into the database</u>

```
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed, FileRequired
from wtforms import StringField, SubmitField, PasswordField, BooleanField, TextAreaField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from musicapp.models import User
from musicapp import images, audios
from flask_login import current_user

class SignUpForm(FlaskForm):
    fname = StringField("First name",
                    validators=[DataRequired(), Length(min=2, max=75)])
    lname = StringField("Last name",
                    validators=[DataRequired(), Length(min=2, max=75)])
    email = StringField("Email",
                    validators=[DataRequired(), Email()])
    username = StringField("Username",
                    validators=[DataRequired(), Length(min=2, max=32)])
    password = PasswordField("Password",
                    validators=[DataRequired()])
    confirm_password = PasswordField("Confirm Password",
                    validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField("Sign Up")

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken please choose a different one.')
    def validate_email(self, email):
        email = User.query.filter_by(email=email.data).first()
        if email:
            raise ValidationError('That email is taken please choose a different one.')

class LoginForm(FlaskForm):
    email = StringField("Email",
                        validators=[DataRequired(), Email()])
    password = PasswordField("Password",
                        validators=[DataRequired()])
    remember = BooleanField("Remeber me")
    submit = SubmitField("Log In")

class UpdateAccountForm(FlaskForm):
    email = StringField("Email",
                        validators=[DataRequired(), Email()])
    username = StringField("Username",
                        validators=[DataRequired(), Length(min=2, max=32)])
    fname = StringField("First name",
                        validators=[DataRequired(), Length(min=2, max=75)])
    lname = StringField("Last name",
                        validators=[DataRequired(), Length(min=2, max=75)])
    picture = FileField("Update Profile Picture",
                        validators=[FileRequired(), FileAllowed(images, 'Upload only images!')])
    submit = SubmitField("Update")

    def validate_username(self, username):
        if username.data != current_user.username:
            user = User.query.filter_by(username=username.data).first()
            if user:
                raise ValidationError('That username is taken please choose a different one.')

    def validate_email(self, email):
        if email.data != current_user.email:
            email = User.query.filter_by(email=email.data).first()
            if email:
                raise ValidationError('That email is taken please choose a different one.')

class UploadPostForm(FlaskForm):
    title = StringField("Title",
                        validators=[DataRequired()])
    content = TextAreaField("Content",
                        validators=[DataRequired()])
    submit = SubmitField("Post")

class UploadSongForm(FlaskForm):
    title = StringField("Title",
                        validators=[DataRequired()])
    description = TextAreaField("Description",
                        validators=[DataRequired()])
    name = StringField("Song Title",
                        validators=[DataRequired()])
    album = StringField("Album",
                        validators=[DataRequired()])
    genre = StringField("Genre",
                        validators=[DataRequired()])
    artist = StringField("Artist Name",
                        validators=[DataRequired()])
    song = FileField("Browse song file",
                    validators=[FileRequired(), FileAllowed(audios, 'Upload only audio files!')])
    submit = SubmitField("Upload")

class UploadPodcastForm(FlaskForm):
    title = StringField("Title",
                    validators=[DataRequired()])
    description = TextAreaField("Description",
                    validators=[DataRequired()])
    name = StringField("Podcast Title",
                    validators=[DataRequired()])
    category = StringField("Category",
                    validators=[DataRequired()])
    artist = StringField("Artist Name",
                    validators=[DataRequired()])
    podcast = FileField("Browse podcast file",
                    validators=[FileRequired(), FileAllowed(audios, 'Upload only audio files!')])
    submit = SubmitField("Upload")

class SongSearchForm(FlaskForm):
    name = StringField("Song Title",
                    validators=[DataRequired()])
    album = StringField("Album")
    genre = StringField("Genre")
    submit = SubmitField("Search")

class PodcastSearchForm(FlaskForm):
    name = StringField("Podcast Title",
                    validators=[DataRequired()])
    category = StringField("Category")
    submit = SubmitField("Search")
```

Signup form code

```
@app.route("/signup", methods=['POST', 'GET'])
def signup():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = SignUpForm()
    if form.validate_on_submit():
        hashed_passw = bcrypt.generate_password_hash(form.password.data).decode('UTF-8')
        user = User(username = form.username.data, fname = form.fname.data,
                lname = form.lname.data, email = form.email.data,
                password = hashed_passw)
        db.session.add(user)
        db.session.commit()
        flash(f"Your account has been created and you can login", "success")
        return  redirect(url_for('login'))
    return render_template("signup.html", form=form, title='Register')
```

Login form code

```
@app.route("/login", methods=['POST', 'GET'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(url_for(next_page[1:])) if next_page else redirect(url_for('home'))
        else:
            flash(f"Incorrect password or email!", "danger")
    return render_template("login.html", form=form, title='login')
```

Code to upload a post

```
@app.route("/upload/post/new", methods=['GET', 'POST'])
@login_required
def new_post():
    form = UploadPostForm()
    if form.validate_on_submit():
        post = Post(title=form.title.data, content=form.content.data, author=current_user)
        db.session.add(post)
        db.session.commit()
        flash('Your post has been added successfully!', 'success')
        return redirect(url_for('home'))
    return render_template('upload_post.html', title='New Post', form=form)
```

Code to upload a song

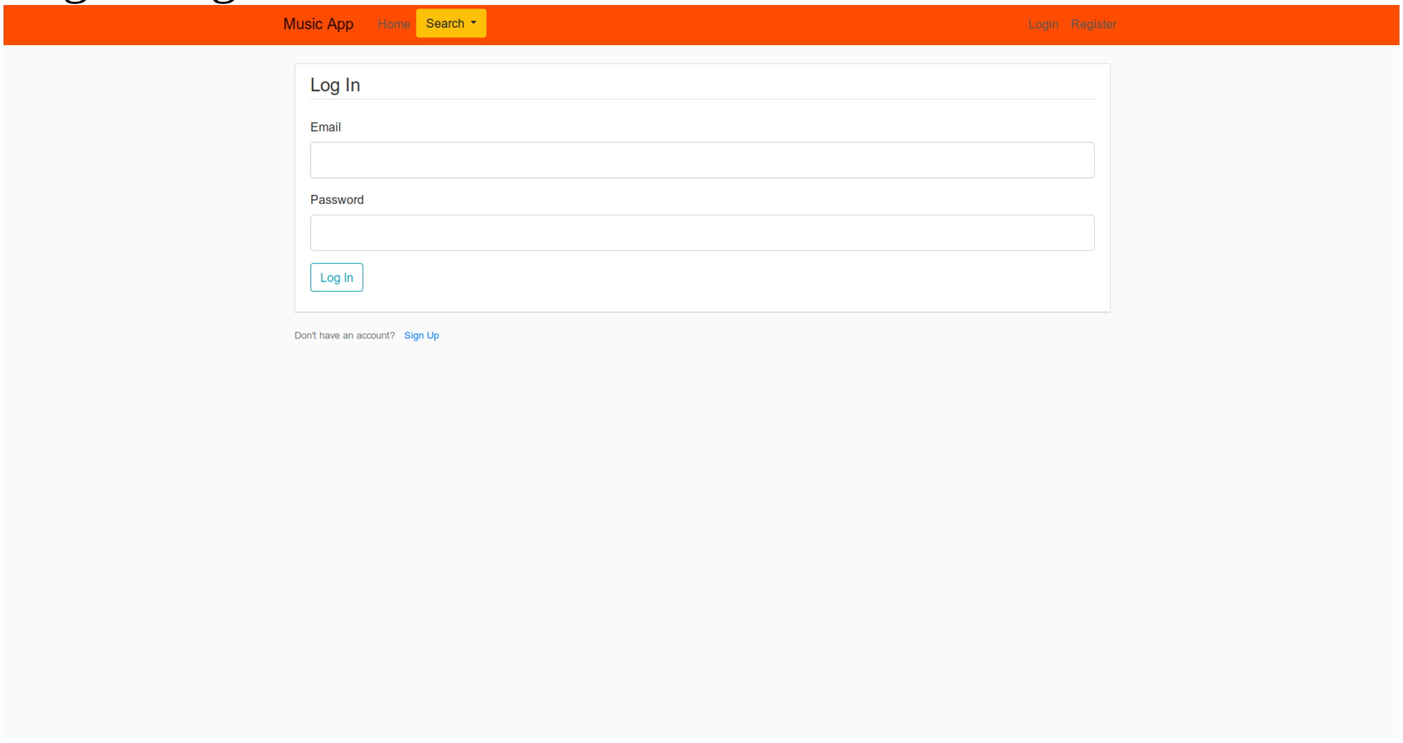
```
@app.route("/upload/song/new", methods=['GET', 'POST'])
@login_required
def new_song():
    form = UploadSongForm()
    if form.validate_on_submit():
        if request.method == 'POST':
            song_file = request.files['song']
            random_hex = secrets.token_hex(8)
            _, f_ext = os.path.splitext(song_file.filename)
            song_file.filename = 'm_' + random_hex + f_ext
            audio_file = audios.save(song_file)
        artist = form.artist.data
        artist = Artist.query.filter_by(name=artist).first()
        if artist:
            artist_id = artist.id
            song = Song(name=form.name.data, album=form.album.data, genre=form.genre.data, title=form.title.data,
description=form.description.data, file_location=audio_file)
            db.session.add(song)
            db.session.commit()
            song.composer.append(artist)
            db.session.commit()
            flash('Your song has been added successfully!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Entered artist does not exist', 'danger')
            return redirect(url_for('new_song'))
    return render_template('upload_song.html', title='New Song', form=form)
```

Code to upload a podcast

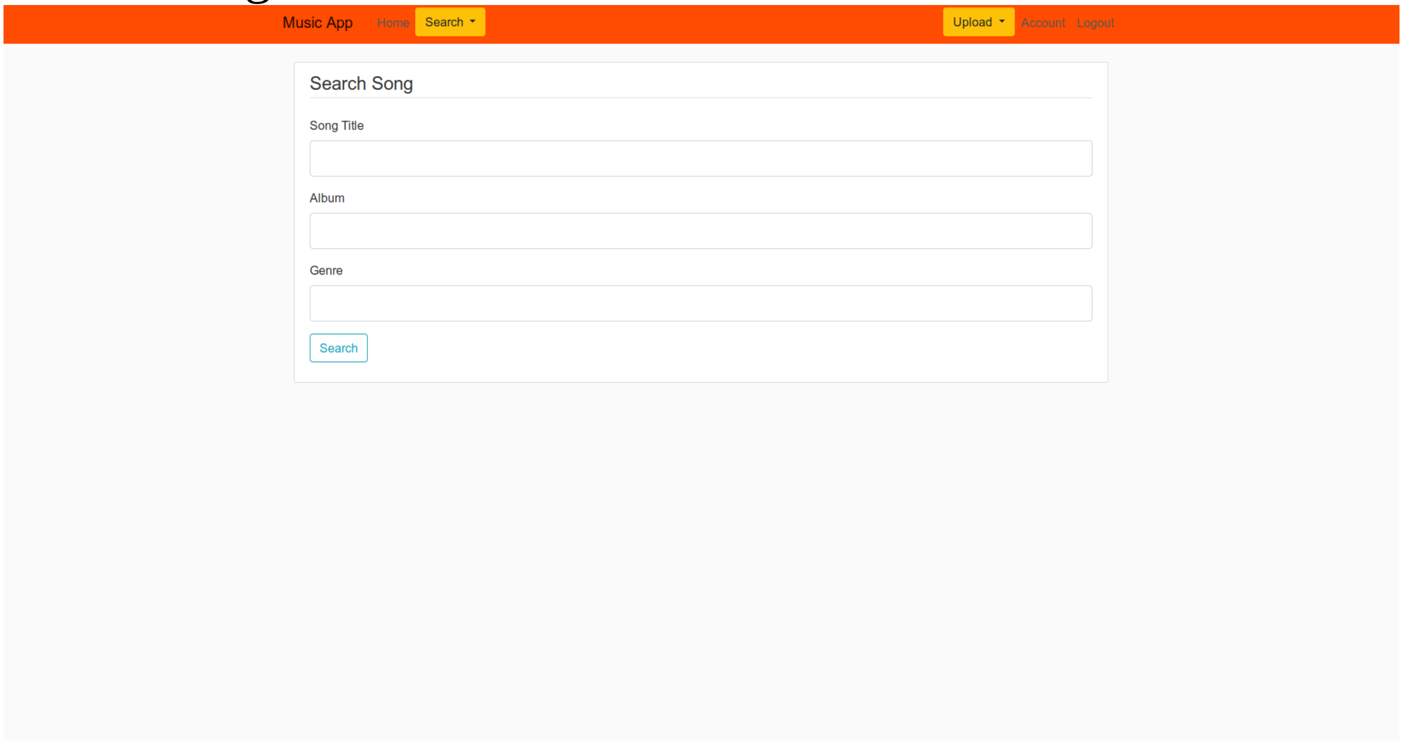
```
@app.route("/upload/podcast/new", methods=['GET', 'POST'])
@login_required
def new_podcast():
    form = UploadPodcastForm()
    if form.validate_on_submit():
        if request.method == 'POST':
            podcast_file = request.files['podcast']
            random_hex = secrets.token_hex(8)
            _, f_ext = os.path.splitext(podcast_file.filename)
            podcast_file.filename = 'p_' + random_hex + f_ext
            audio_file = audios.save(podcast_file)
        artist = form.artist.data
        artist = Artist.query.filter_by(name=artist).first()
        if artist:
            artist_id = artist.id
            podcast = Podcast(name=form.name.data, category=form.category.data, title=form.title.data,
description=form.description.data, file_location=audio_file, artists=artist)
            db.session.add(podcast)
            db.session.commit()
            flash('Your podcast has been added successfully!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Entered artist does not exist', 'danger')
            return redirect(url_for('new_song'))
    return render_template('upload_podcast.html', title='New Podcast', form=form)
```

Front End Screenshots:

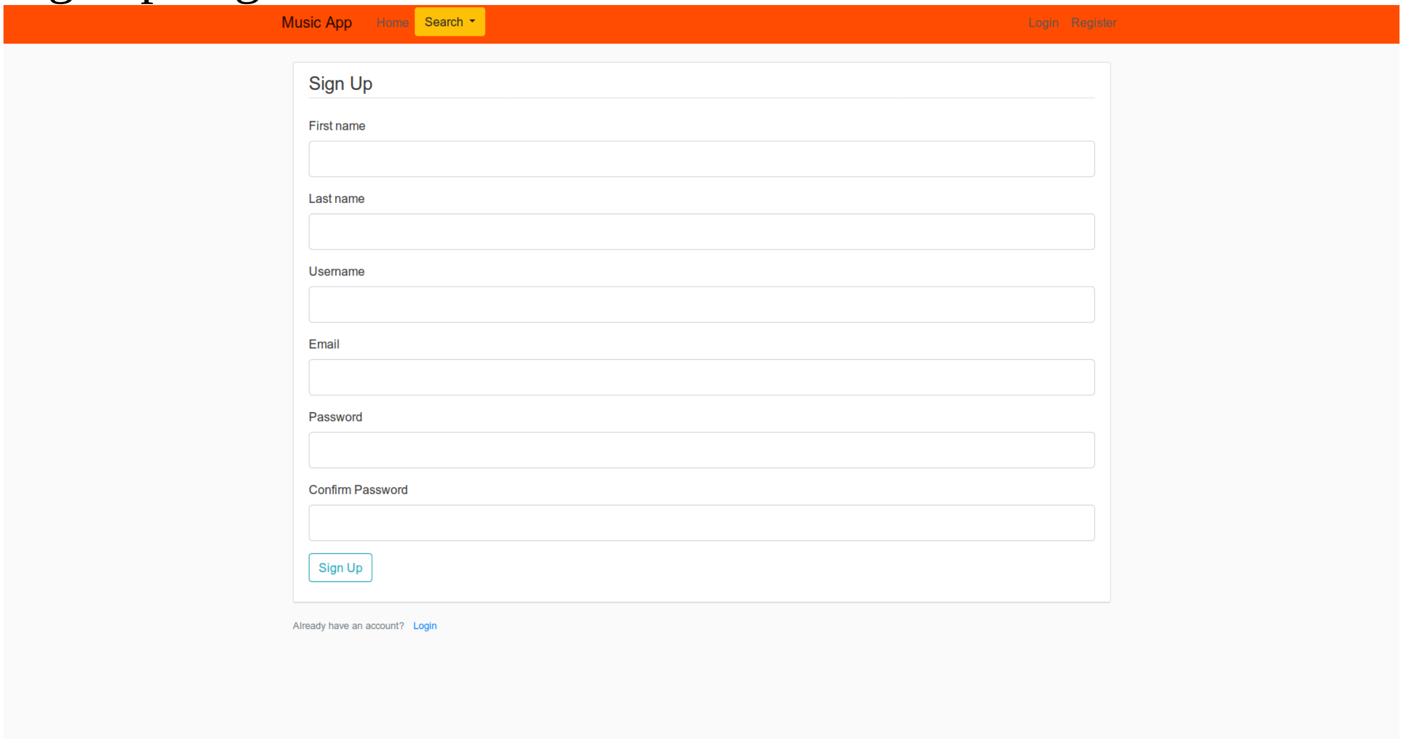
Log in Page:



Search Song:



Sign up Page:



Music App Home Search Login Register

Sign Up

First name

Last name

Username

Email

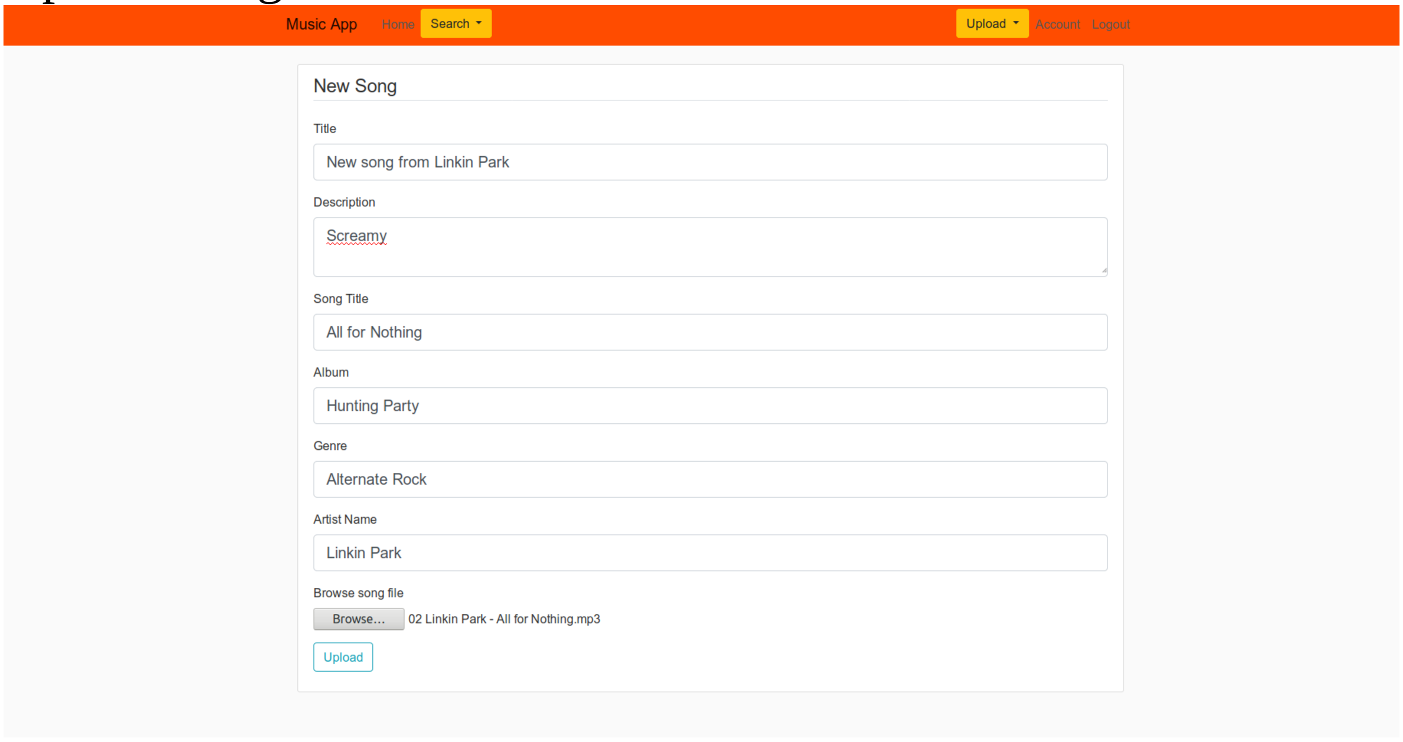
Password

Confirm Password

Sign Up

Already have an account? Login

Upload Song:



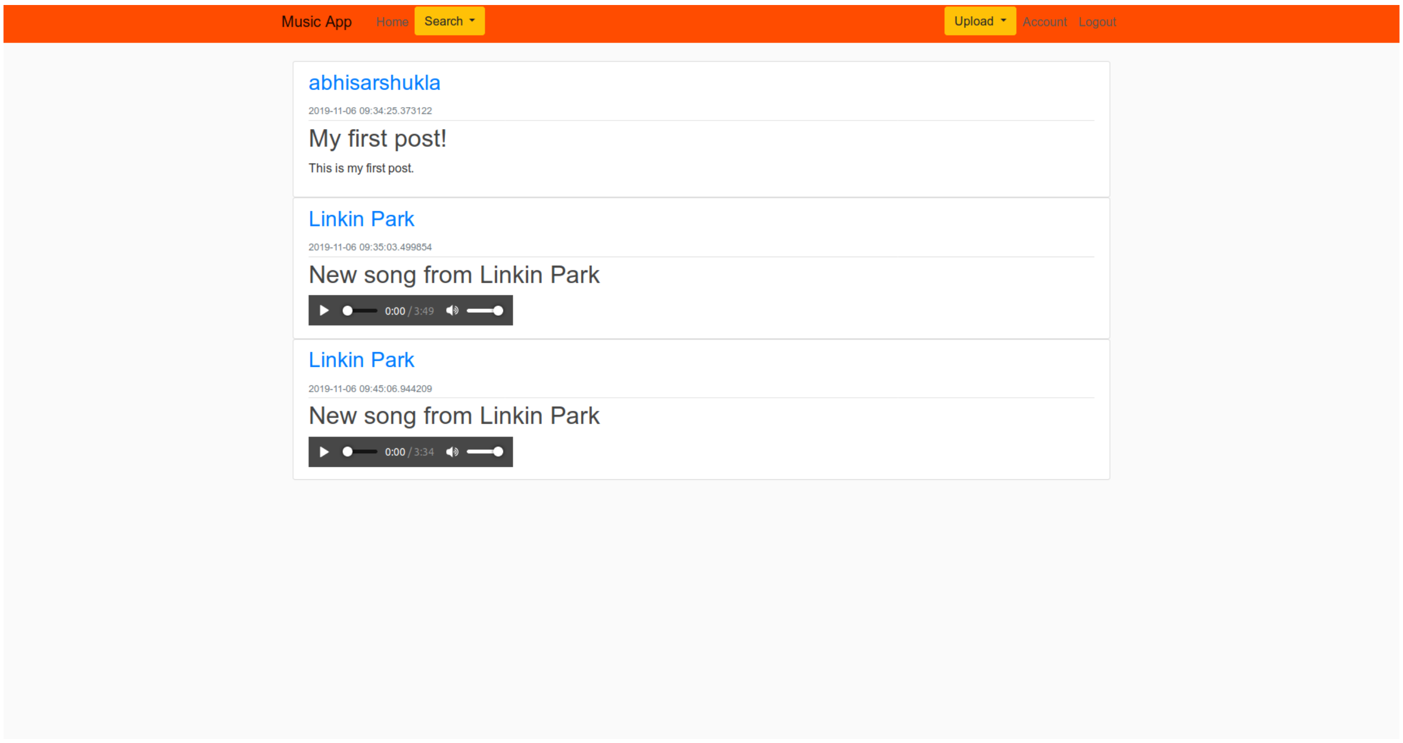
Music App Home Search Upload Account Logout

New Song

Title	New song from Linkin Park
Description	Screamy
Song Title	All for Nothing
Album	Hunting Party
Genre	Alternate Rock
Artist Name	Linkin Park
Browse song file	Browse... 02 Linkin Park - All for Nothing.mp3

Upload

Home Page:
1. Logged in:

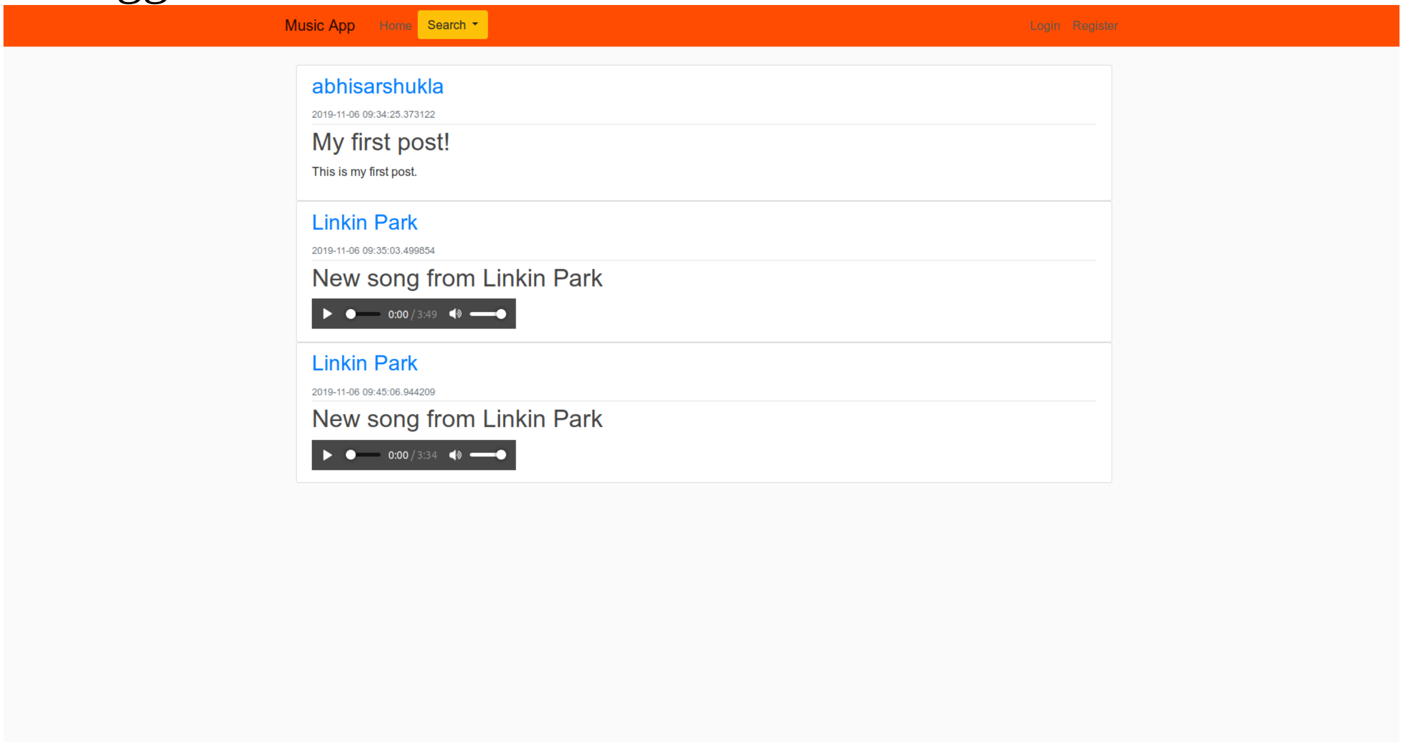


abhisarshukla
2019-11-06 09:34:25.373122
My first post!
This is my first post.

Linkin Park
2019-11-06 09:35:03.499854
New song from Linkin Park
0:00 / 3:49

Linkin Park
2019-11-06 09:45:06.944209
New song from Linkin Park
0:00 / 3:34

2. Logged out:



abhisarshukla
2019-11-06 09:34:25.373122
My first post!
This is my first post.

Linkin Park
2019-11-06 09:35:03.499854
New song from Linkin Park
0:00 / 3:49

Linkin Park
2019-11-06 09:45:06.944209
New song from Linkin Park
0:00 / 3:34