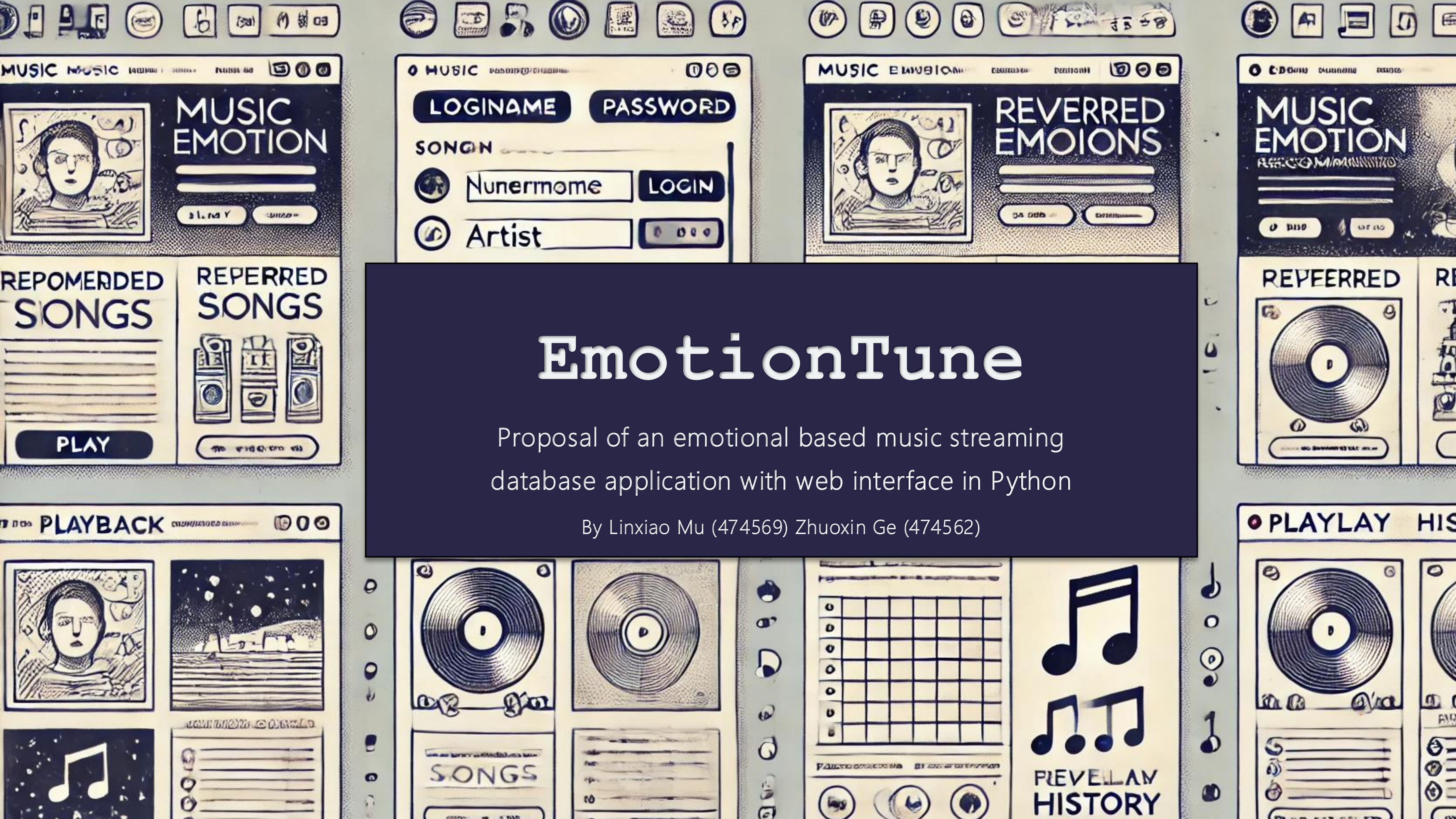


EmotionTune

Proposal of an emotional based music streaming database application with web interface in Python

By Linxiao Mu (474569) Zhuoxin Ge (474562)

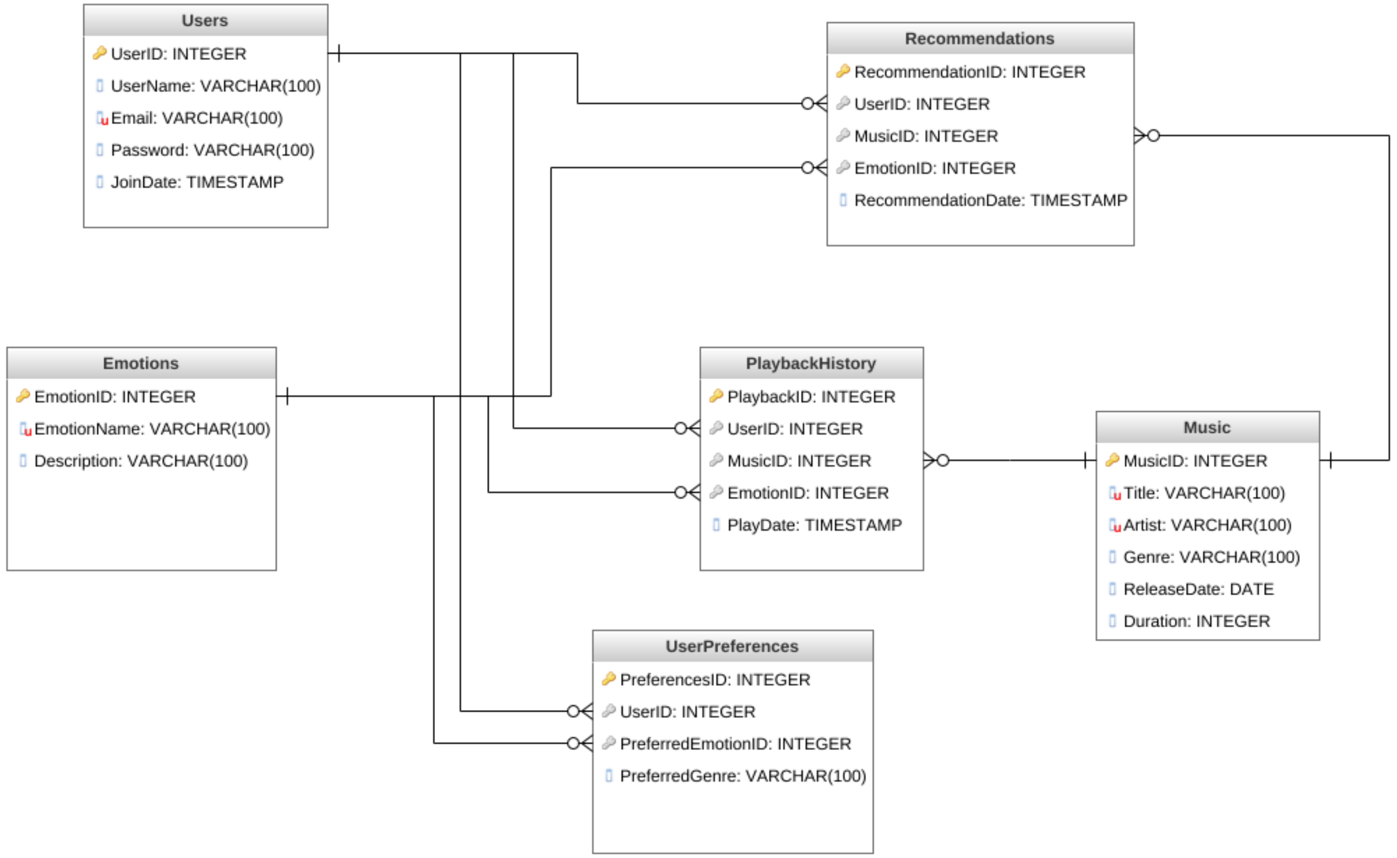
INTRODUCTION OF PROJECT

In the era of digital transformation, music streaming platforms have revolutionized how people access and experience music. However, the emotional connection between listeners and music remains underexplored. Our project, EmotionTune, bridges this gap by leveraging a database-driven application to recommend music tailored to users' emotional states. By analyzing user inputs and providing an intuitive interface, EmotionTune enhances user experience through personalized, emotion-driven recommendations.

The platform is built using Python and SQLite, featuring a web-based interface for ease of access. The database architecture is designed to manage complex relationships between users, emotions, and musical attributes, ensuring accurate and efficient recommendations. By integrating advanced SQL functionalities such as triggers, views, and complex queries, the system guarantees robust data handling and user satisfaction.

EmotionTune is not just a music recommendation tool; it is an innovation that connects emotions to melodies, creating an immersive and personalized musical journey.

DATABASE SCHEME



Indexes of tables :

Users – Email

Music -Title

Emotions-EmotionName

Recommendations-UserID, MusicID,

EmotionID

PlaybackHistory-UserID, MusicID,

EmotionID

UserPreferences-UserID,

PreferredEmotionID

DESCRIPTION OF SQL QUERIES

1. Complex SELECT Query

The goal is to retrieve detailed information about songs associated with a specific emotion, including user details, song titles, artist names, and playback dates.
Description: This query joins the Users, PlaybackHistory, Music, and Emotions tables to ensure accurate data relationships. It filters records based on a specified emotion (e.g., "Happy") and retrieves data linking users, the songs they played, and the associated emotions. By doing so, it offers a complete view of user interactions with emotionally tagged music.
Purpose: This query helps analyze user behavior and preferences for specific emotions, such as identifying users who frequently listen to "Happy" songs and when these interactions occurred.

2. Subquery

The goal is to identify the top three most frequently played songs on the platform to recognize trending tracks.
Description: The subquery aggregates playback counts for each song by grouping PlaybackHistory records based on MusicID. Using COUNT, it calculates the number of times each song was played. The outer query sorts these results in descending order of play counts and limits the output to the top three songs.
Purpose: This query is instrumental for identifying popular tracks, which can be used to enhance recommendations or display "Top Trending Songs" to users on the platform.

DESCRIPTION OF SQL QUERIES

3. Index Usage

The goal is to optimize queries that search for users by their email, such as during the login process.
Description: An index on the Email column in the Users table allows rapid lookup of user information. Without scanning the entire table, the database can quickly locate the record based on the indexed email. This drastically improves performance, especially for large datasets. Purpose: It enhances user authentication speed, making the login process seamless and efficient for the platform's users.

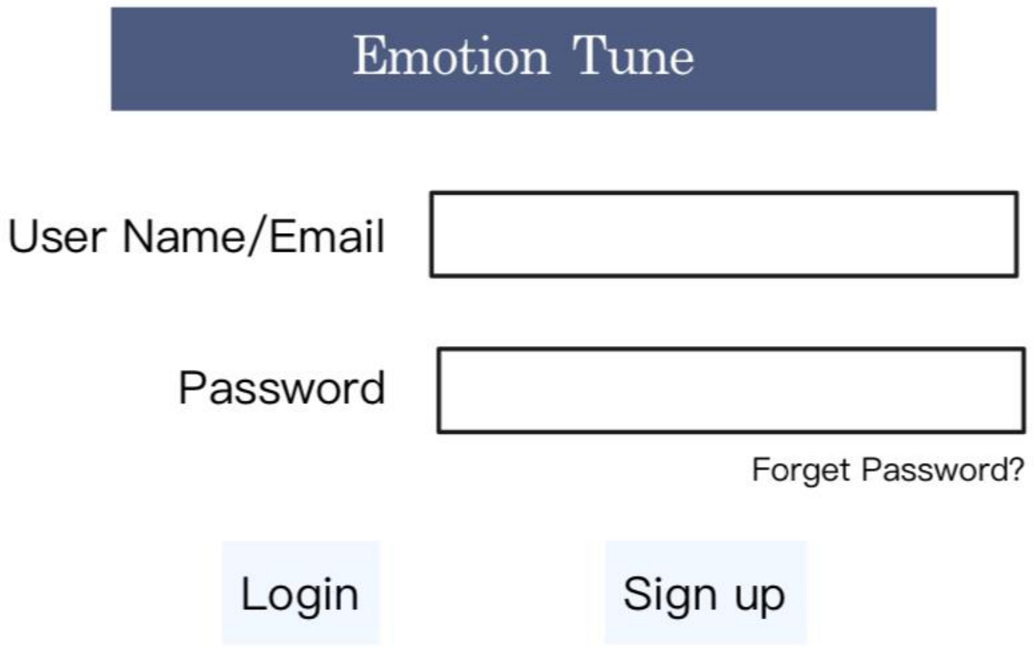
4. Trigger

The goal is to automatically update the LastPlayed field in the Music table whenever a song is played.
Description: A trigger is defined on the PlaybackHistory table to activate after a new playback record is inserted. The trigger ensures the corresponding LastPlayed field in the Music table reflects the most recent play date for that song. This automation guarantees data accuracy without requiring manual intervention. Purpose: It maintains real-time updates to the last-played information for songs, enabling accurate tracking and enhancing user experience by showing up-to-date data.

5. View

The goal is to create a simplified view that compares user preferences with the platform's recommendations. Description: This view combines data from UserPreferences, Recommendations, Music, and Emotions tables. It filters records to include only recommendations matching user preferences, such as preferred genres or emotions. The view provides a concise and easy-to-analyze dataset for evaluating the alignment of recommendations with user interests. Purpose: It aids administrators in analyzing the platform's recommendation effectiveness and identifying areas for improvement in matching user preferences with recommended content.

DETAILED DESCRIPTION OF THE SCREENS



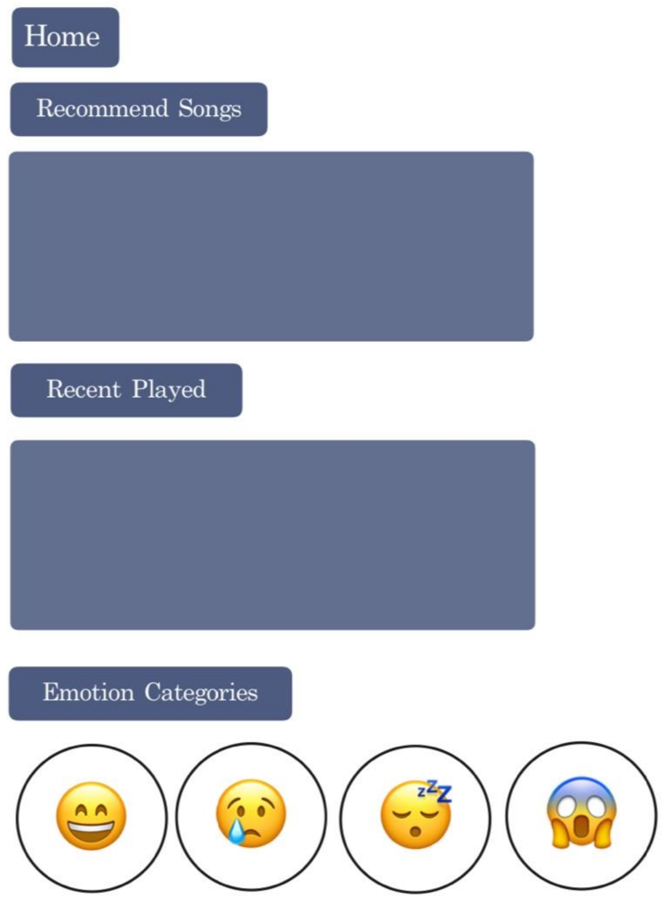
Login Page
Purpose: To authenticate users and provide secure access to the platform.
Features:

1. Username or email input field.
2. Password input field.
3. A "Login" button to submit credentials.
4. Links for "Sign Up" (new users) and "Forgot Password" (password recovery).

Functionality:

1. Verifies user credentials against the database.
2. Redirects authenticated users to the homepage.

DETAILED DESCRIPTION OF THE SCREENS

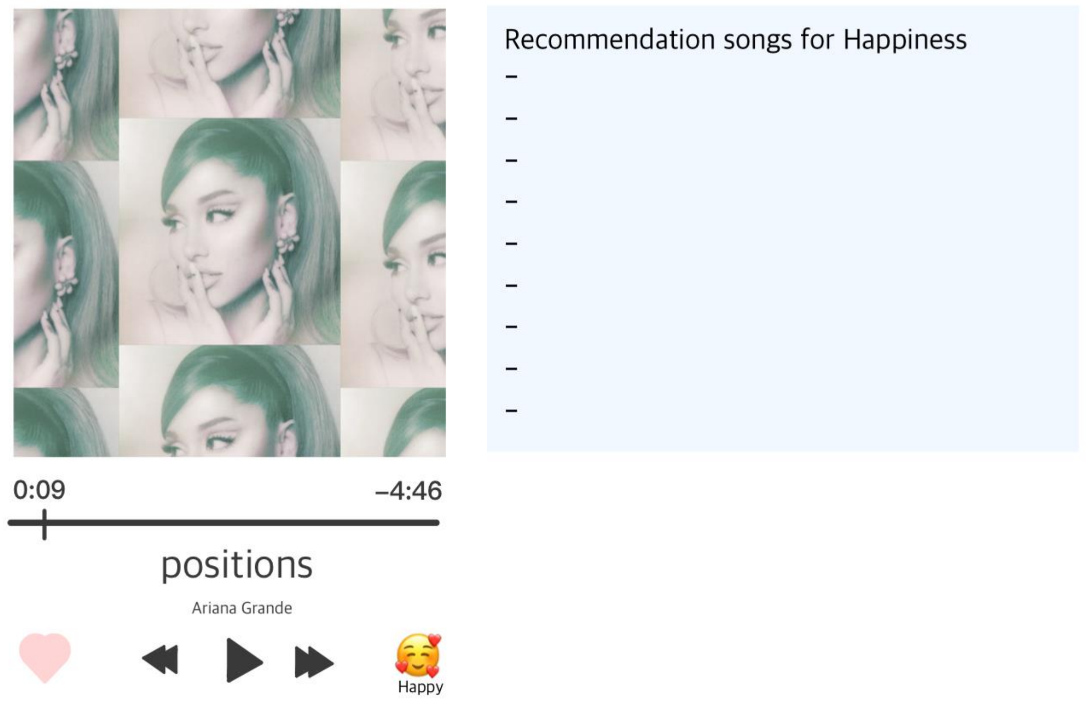


Homepage
Purpose: Serves as the central dashboard for users to explore content.
Features:

1. Navigation bar for easy access to other subpages.
2. Search bar to find songs, artists, or emotions.
3. Sections for:
    - **Recommended Songs**: Displays songs tailored to the user's preferences.
    - **Recently Played**: Lists tracks the user listened to recently.
    - **Emotion Categories**: Buttons or icons to filter songs by emotions (e.g., Happy, Sad, Calm).

Functionality: Dynamically updates content based on user preferences and activity.

DETAILED DESCRIPTION OF THE SCREENS



Music Detail Page

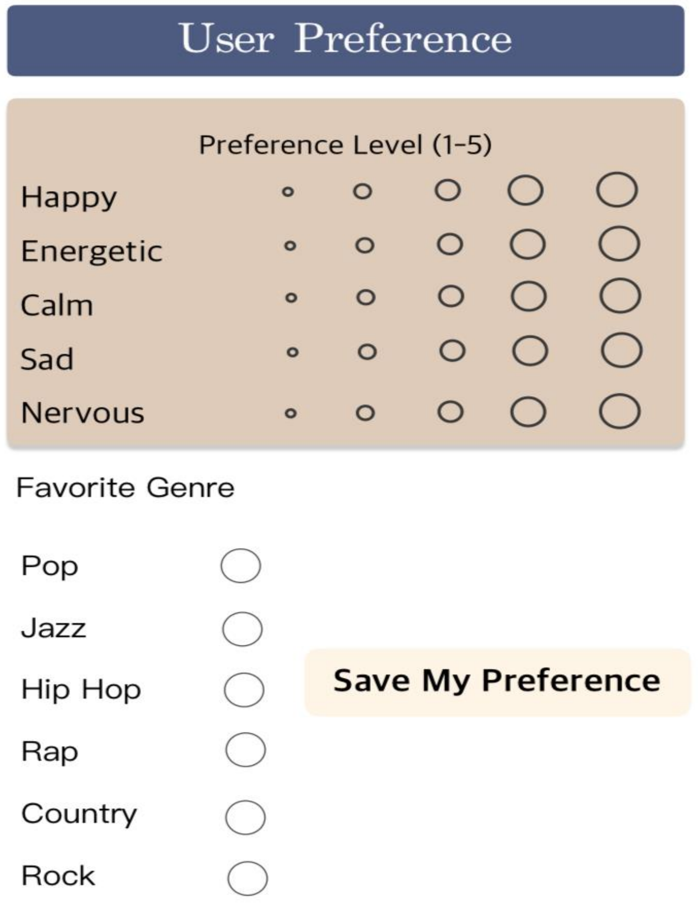
Purpose: Provides detailed information about a specific song.

Features:

1. Displays the song title, artist name, album, genre, and release date.
2. Shows emotion tags associated with the song.
3. Includes playback controls:
    - "Play" button for immediate playback.
    - "Add to Playlist" or "Favorite" button for saving the song.
    - Recommendations for similar songs based on emotion and genre.

Functionality: Allows users to interact with and learn more about individual tracks.

DETAILED DESCRIPTION OF THE SCREENS



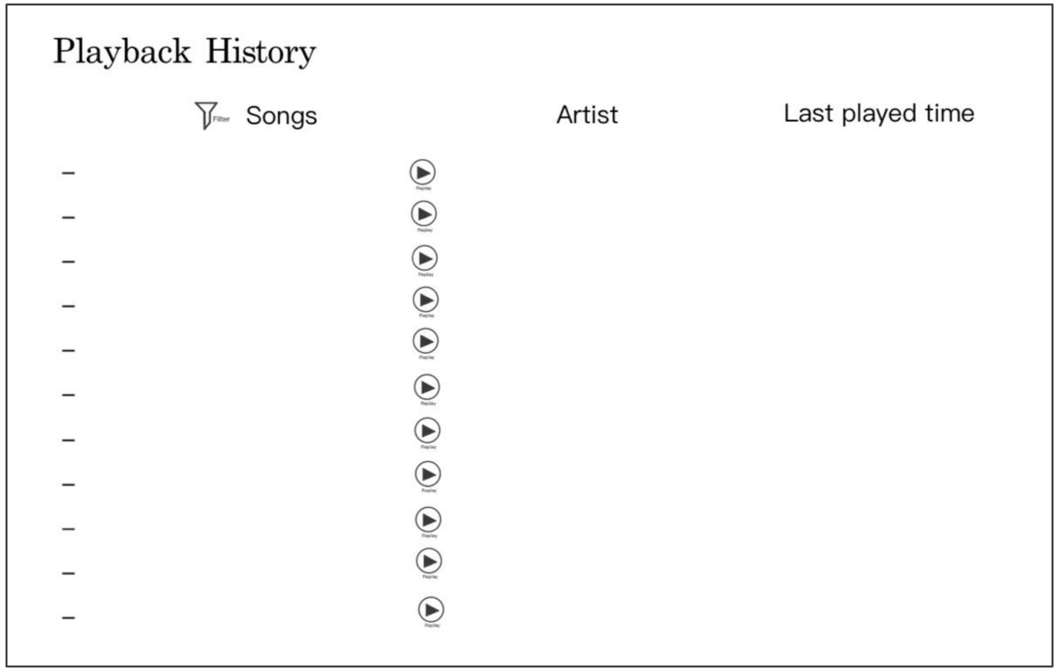
User Preferences Page
Purpose: Enables users to customize their preferences for emotions and genres.

Features:

- A section to select preferred emotions
- A section to choose favorite genres
- A "Save Preferences" button to store changes.

Functionality: Updates the database with the user's preferences to personalize recommendations.

DETAILED DESCRIPTION OF THE SCREENS



Playback History Page

Purpose: Displays the user's listening history.

Features:

- A table or list of songs the user has played, including:
  - Song title and artist.
  - Play date and time.
- Filters to sort or search history by date, genre, or emotion.
- A "Replay" button for each song to play it again.

Functionality:

- Provides a record of past interactions and allows users to revisit their listening history.

LIST OF PYTHON PACKAGE

- Flask: Create and manage routes for the web interface, Serve dynamic HTML pages for login, homepage, song details, user preferences, and playback history. Handle user requests and responses for song recommendations and playback interactions.
- Flask-SQLAlchemy: Manage SQLite database interactions for tables like Users, Music, PlaybackHistory, and Recommendations. Simplify queries related to song recommendations, user preferences, and playback history. Define relationships between database tables in Pythonic syntax.
- SQLite3: Store user data, playback history, song metadata, and emotion preferences. Handle structured data like user preferences, emotions, and song relationships efficiently.
- WTForms: Validate user input for login, signup, and preference settings. Ensure form fields (e.g., password strength, email format) are filled correctly before submission.

LIST OF PYTHON PACKAGE

- Jinja2: Populate HTML templates with data from the backend (e.g., recommended songs, playback history). Render user-specific content like personalized playlists or preferences dynamically.
- Pandas: Analyze user playback history and preferences to refine song recommendations. Aggregate and filter data for administrative insights into platform usage.
- Matplotlib: Generate graphical insights into user engagement, such as most played songs or emotional trends. Create visual dashboards for administrators to monitor platform performance.
- Flask-Bootstrap: Design visually appealing and responsive web pages for the platform. Use pre-built UI components like buttons, forms, and navigation bars to enhance user experience.
- Pillow: Handle and display album covers or other images in the song detail pages. Manage profile pictures for user accounts.
- Scikit-learn: Analyze user behavior to suggest songs based on past preferences. Develop emotion-based recommendation systems using clustering or classification.

TASK DISTRIBUTION

Task	Responsible Person	Notes
Database Design	Linxiao Mu	Design schema diagram, create table structures
Implementation of Recommendation Algorithm	Zhuoxin Ge	Develop SQL queries and triggers
Front-end Page Development	Linxiao Mu	Login, emotion recording, recommendation pages
Data Visualization	Zhuoxin Ge	Generate trend charts using Matplotlib/Plotly
Database and Front-end Integration	Both	Complete integration using Flask
Function Testing and Optimization	Both	Validate recommendation accuracy, interface compatibility
Documentation for Database	Linxiao Mu	Table structure explanation, SQL code documentation
Documentation for Front-end and User Manual	Zhuoxin Ge	Interface screenshots, feature descriptions
Project Presentation Slides	Both	Present overall workflow and outcomes