1 RecomMuse

Our Music Recommendation System utilizes the algorithms mentioned here to provide personalized music recommendations to users. Here's how the system suggests music to users:

1.1 User Interaction

User Login: When a user logs into our music platform, their preferences and interactions are tracked to provide tailored recommendations.

Interactive Music Player: The user interacts with the interactive music player web page generated by our system (as described in Algorithm 1). This web page allows users to explore and listen to songs.

1.2 Recommendation Generation

Song Similarity Calculation: In the background, our system employs the Song Similarity and Ranking Algorithm (Algorithm 2) to calculate the similarity between songs based on various factors, including genre, language, and user liking. The combined similarity score is a key metric that quantifies how closely two songs match each other.

Collaborative Filtering: The system considers the user's interactions, such as liked and unliked songs, to determine their preferences. It identifies other users who have similar preferences and music history.

Personalized Recommendations: Using collaborative filtering and the calculated song similarities, our system generates a list of personalized song recommendations for the user. These recommendations are ranked based on their combined similarity scores, ensuring that the most relevant songs appear at the top of the list.

1.3 User Feedback

User Feedback Loop: As the user listens to and interacts with the recommended songs, the system continues to collect feedback. Users can like or dislike songs, and these actions are recorded in the "liked_songs" table.

1.4 Continuous Improvement

Adaptive Recommendations: Over time, the system adapts to the user's evolving preferences. It continuously refines its recommendations by incorporating the user's feedback, ensuring that the music suggestions become more accurate and aligned with the user's taste.

By following this process, our Music Recommendation System creates a dynamic and personalized music discovery experience for users, offering them songs that match their preferences and allowing them to explore new music that suits their taste.

Algorithm 1 Algorithm for Music Recommendation System Web Page

Require: Database tables: "users," "songs," "liked_songs" (with song_id, user_id, liked, and unliked columns)

Ensure: Interactive music player web page

Include necessary PHP libraries and stylesheets.
Start a PHP session.
Fetch music data from the database:
  - Select song details including id, name, artist, album, music file, cover image, genre, language, and combined similarity score from the "songs" table.
  - Calculate combined similarity score using a subquery that calculates genre, language, and liking similarity components for song pairs.
Initialize variables for likedStatus and unlikedStatus.
if Music data is not empty and user is logged in **then**
  Fetch liked and unliked status for the currently playing song from the "liked_songs" table.
    - Execute a SQL query to retrieve liked and unliked status for the current user and song.
    - Update the likedStatus and unlikedStatus variables based on the query result.
end if
Create the HTML structure for the music player web page.
for all Songs in the music data **do**
  Create a list item for the song with the song's key.
    - Display song details including name and artist.
    - Add an event handler to play the selected song when clicked.
end for
Initialize an Audio object for playing music.
Initialize variables for currentMusicKey and isPlaying.
Define a function playMusic(key) to play the selected music by its key:
  - Set the audio source to the selected music file.
  - Update the current music information.
  - Reset the like and unlike button colors.
  - Fetch and update liked and unliked status for the currently playing song.
  - Play the audio.
Define a function fetchLikedStatus() to fetch liked and unliked status for the currently playing song from the server.
Define a function to handle pause/play button clicks:
  - Toggle between pausing and playing the audio.
  - Update the button appearance accordingly.
Define functions to handle previous and next button clicks:
  - Play the previous or next song in the playlist if available.
Update the timeline as the audio plays:
  - Calculate and display current time, end time, and the progress bar.
Define a function to handle timeline click events:
  - Calculate the seek time based on the click position and update the audio playback position.
Define functions to update button colors based on liked and unliked status:
  - Update thumbs-up and thumbs-down button colors.
Define functions to handle thumbs-up and thumbs-down button clicks:
  - Toggle the liked and unliked status.
  - Update button colors.
  - Update liked and unliked status in the database through an AJAX request.
Define a function to update liked and unliked status in the database through an AJAX request.

Algorithm 2 Song Similarity and Ranking Algorithm

Require: Database tables: "users," "songs," "liked_songs" (with song_id, user_id, and liked columns)

Ensure: List of songs ranked by combined similarity score

1: Initialize an empty list to store song objects with their respective combined similarity scores.
2: **for** each unique pair of songs (s1, s2) from the "songs" table where s1.id ¡ s2.id **do**
3: Calculate genre similarity component (genre_similarity):
4: - Count the number of common genres between s1 and s2.
5: - Calculate the total number of unique genres between s1 and s2.
6: - Calculate $genre\_similarity = \frac{CommonGenres}{TotalUniqueGenres}$.
7: Calculate language similarity component (language_similarity):
8: - Count the number of common languages between s1 and s2.
9: - Calculate the total number of unique languages between s1 and s2.
10: - Calculate $language\_similarity = \frac{CommonLanguages}{TotalUniqueLanguages}$.
11: Calculate user liking similarity component (liking_similarity):
12: - Retrieve the list of users who liked s1 (from the "liked_songs" table).
13: - Retrieve the list of users who liked s2 (from the "liked_songs" table).
14: - Calculate the number of common users who liked both s1 and s2.
15: - Calculate the total number of unique users who liked s1 and s2.
16: - Calculate $liking\_similarity = \frac{CommonLikedUsers}{TotalUniqueLikedUsers}$.
17: Calculate combined similarity score (combined_similarity) for the song pair (s1, s2):
18: - Combine the three similarity components using specified weights (e.g., 0.4 for genre, 0.2 for language, and 0.4 for liking).
19: - $combined\_similarity = (0.4 \cdot genre\_similarity) + (0.2 \cdot language\_similarity) + (0.4 \cdot liking\_similarity)$.
20: Create a song object for s1 (if not already created) and update its combined similarity score by adding *combined_similarity*.
21: **end for**
22: Sort the list of song objects by *combined_similarity* in descending order to get a ranked list of songs.
23: **return** The ranked list of songs.