Technical Specification for Rock and Roll Music API

Cole Robinson, Geoff Blaylock

Overview

Our music API will focus on the different rock and roll genre's in the music industry. The database will consist of multiple schemas such as artist, album, song, and playlist. Utilizing this data, the API will be able to make music recommendations, artist recommendations, album recommendations, and create playlists based on given input. Additionally, the API will have read/write capabilities, allowing users to update the database themselves.

User Stories

1. As a user, I want to search for a specific song, artist, or album so that I can find the desired information quickly.
2. As a musician, I want to add new songs, artists, and albums to the database so my music can be shared with others.
3. As a musician, I would like to be able to update or delete my music from the database.
4. As a user, I want to be able to have a playlist created for me based on my previous song, artist, or album interests.
5. As a user, I would like to find similar music from individual songs, artists, or albums to broaden my music knowledge.

Endpoints

1.) **/search?type={type}&query={query} (GET) -** This endpoint will be used to search the entire database
- Type: the 'type' field will be used to specify if the user would like to search for a song, album, artist, playlist. If no type is provided, the search will include all aspects matching.
- The query field will be used to take the user's text input and match the request to find items.

2.) /songs/{id}: (GET) This endpoint will return a single song by its identifier

- song_id: internal id of the song
- artists: a list of the artists' names that perform in the song
- album: the name of the album the song is on
- runtime: the time in seconds of the song
- genre: the genre that best fits the style of the song

3.) /artists/{id}: (GET) This endpoint will return a single artist by its identifier

- artist_id: internal id of the artist
- name: the name of the artist

- gender: the gender of the artist
- age: the current age of the artist
- num_songs: total number of songs that the artist has in the database
- num_albums: total number of albums that the artist has in the database

4.) /albums/{id}: (GET) This endpoint will return a single album by its identifier

- album_id: the internal id of the album
- name: the name of the album
- artist_ids: a list of all of the artist_id's that the album belongs to
- song_ids: a list of all the song_id's that are in the album
- total_runtime: the total length of the album if it were to be played through entirely

5.) /playlists/{id}: (GET) This endpoint will return a single playlist by its identifier

- Playlist_id: the internal id of the playlist
- name: the name of the playlist
- song_ids: a list of all of the song_id's that are in the playlist
- total_runtime: the total length of the playlist if it were to be played through entirely

6-10.) (POST) Endpoints - Each of these endpoints will have post functionality as well for musicians looking to add music to the database. The respective inputs for each endpoint will likely be in JSON format and will require input validation prior to posting.

"/songs"

- **Inputs:** genre (Text), total_runtime (seconds) (int), artist_id, album_id or null
- **Output:** song_id of the newly created song, or exception if failed

"/artists"

- **Inputs:** name (Text) , gender (Text), birthday (Time)
- **Output:** artist_id of the newly created song, or exception if failed

"/albums"

- **Inputs:** name(Text)
- **Output:** album_id of the newly created song, or exception if failed

"/playlists"

- **Inputs:** name(Text), total_runtime (seconds) (int),
- **Output:** playlist_id of the newly created song, or exception if failed

See the following ER Diagram for schema references: ER Diagram

11.) /playlists/create (GET) This endpoint will return a created playlist for the user based on user input.

Input:
- song_id: the id of a song to be used to create a stylized playlist
- genre: a unique genre to use to create a stylized playlist
- artist_id: the id of an artist to use to create a stylized playlist

Output:
- Json of a stylized playlist, which can then be posted to the database if desired

Edge Cases

1.) Singles: To handle cases where a user may search for a single that does not belong to an album, we should notify the user.

2.) Multiple artist tracks: In the case that multiple artists are credited for a track, our API should return a data structure that contains data for all artists credited.

3) Bands: In the case that users search for a band or group using the artist endpoint, a data structure containing relevant information on all members should be returned.

4) Input validation: When input does not correspond to any valid data, make sure to raise a detailed error documenting the issue.

5) Multi genre: In the case that a song or album falls into multiple genres, a data structure containing both genres should be returned.

6) Duplicate Entries - In the event that a song, album, or artist potentially gets posted twice, this could lead to undesirable behavior and have transitive effects on other attributes. In order to prevent this, we will make sure that our schema definitions ensure that certain fields are unique. For instance, artist_name will likely be a unique key, as well as the combination of song_id and artist_id.

7) Missing/Incomplete Data - Additionally in our schema design, we will decide which attributes are nullable and which are required. With these definitions in place along with atomic post transactions, there should be less of an issue with missing data, and the endpoint should return a value expected by the user.

8) Large Requests - Our query parameters for the search function will help to limit the amount of data requested by the user. Though we may decide to define a hard limit on the amount of data returned, if this proves to be an issue.