Spotify Project Document

Advanced Programming - Spring 2023

Course Instructor: Dr. Saeed Reza Kheradpisheh

Teaching Assistants: Mobin Nesari, Mahan Madani, Amir Hossein Karami, Hossein Zarrabi, Mohammad Mehdi Begmaz, Mehdi Ahmadi, Parmida Jabbari, Reza Mosavi, Mohammad Rouintan, Farbod Khoramvatan, Pouyan Oskouhi, Parnia Varzdar, Alireza Rahmati, Diba Elahi, Alireza Gharavi, Alireza Afroozi, Aryan Neyzebaz

Introduction

In this project, you will build a music streaming platform like Spotify, a popular and widely-used music streaming service. The application should allow users to browse, search, and play music, create and manage playlists, and add songs and playlists to their personal library.

This project has been designed for groups of 2 or 3 students. Depending on the number of group members, certain features may be mandatory or optional.

Objectives

- **Object-Oriented Programming Concepts**: Design and implement classes and objects to represent the different components of the application, such as users, playlists, and songs. Use inheritance and polymorphism to create a flexible and reusable codebase.

- **File Handling**: Implement functionality to read and write data to and from files. This could include reading and writing metadata for music files, such as the artist, album, and track information, as well as reading and writing the actual music data, such as the audio data in an MP3 file.

You must also implement the functionality to read and write image files, such as album covers or artist photos.

- **Multi-threading**: Use multi-threading to implement both the client and server sides of the app. The server should be able to handle multiple clients at the same time.

- **Socket Programming**: Implement networking features using socket programming. This could include establishing a connection between the client application and a server to exchange data, such as user information or music data.

- **Database Management**: Design and implement a database schema to store user data and music information. Use queries to retrieve and manipulate data.

- **GUI**: Design and implement a user-friendly interface for the application using Java's GUI libraries. It is recommended to use JavaFX for this project.

Main Tasks

(The following tasks are mandatory for all groups)

1. Design The Application's Client-Server API

The Client and Server sides of the app should be able to communicate with each other in a predetermined manner. In order to achieve this, a series of Request-Response interactions between the Client and the Server can be implemented.

1.1. Request

A Request must have these features:

- A Request is sent from the Client to the Server over the network.
- You should create different types of Requests for different actions.
- Any interaction between the Client and the server should be handled using Requests and Responses. Creating an account, viewing an artist’s page, downloading music, etc., should all have unique Requests designed for them.
- It is up to you to design the Request's format. For example, A Request can be a JSON string, which is easy to send on a socket.

1.2. Response

A Response must have these features:

- A Response is sent from the Server to the Client over the network.
- You should create different Response types corresponding to the received Request.
- Attach appropriate data to the Response based on the Request. A Response might contain the data a user has requested, or it might just be a boolean indicating the result of a previously sent Request (such as confirming a user's login attempt by returning a True boolean).
- It is up to you to design the Response's format. Similar to a Request, a Response can also be a JSON string.

Note that each Request received from the Client must be answered with a Response from the Server.

2. Design The Application's Architecture

The Client and Server are the two main components of your app. They must be connected through the use of a socket connection.
Plan out your project's files accordingly. You can research Software Architectural Patterns and use one as a baseline for your project.

- **Note that one server may be active at a time, but multiple clients should be able to connect to that server simultaneously.**

2.1. Client

The Client component is responsible for providing an interactable menu and generating requests. The Client must provide the following functionalities:

- **GUI:** Provide a graphical user interface for users to interact with the app
- **Socket Connection:** Establish a connection with the Server over the local network using socket programming.
- **Request Generator:** Generate Requests based on the user's input.
- **Response Handler:** After sending a Request, wait to receive a Response from the Server. Then provide the user with appropriate info based on the response type.

2.2. Server

The Server component is responsible for handling Client requests and managing the database. The Server must provide the following functionalities:

- **Database Connection:** Before a Server is ready to accept clients, it must connect to the database to access the stored information.
- **Socket Listener:** Listen for incoming Client connections and redirect requests to the appropriate handlers. Once a request has been fully handled, enter listening mode again.
- **Request Handlers:** Process Client Requests and interact with the database to fetch the requested data. You may need to create multiple handlers for various Requests.

- **Database Manager:** Interact with the database system to run a query on the database according to the received Request.
- **Response Generator:** The final step in handling a Request is to send an appropriate Response to the Client. Attach the needed data to the Response based on the Request.
- **Logging:** Try to log every major action the Server performs (e.g. accepting a Client, sending a file, etc.) to simplify the debugging process.

3. Implement The Necessary Features

The following features MUST be implemented:

- **Accounts:** Allow users to create new accounts, log in, and log out (securely).
- **Profile Page:** Each user must have a personal profile page.
- **Artist Page:** Users should be able to view an artist's profile that displays all of their songs and albums.
- **Music Library:** Enable users to browse the available music library.
    - A collection of songs should be included in the music library
    - Each song must have an artist and must belong to an album
    - Each album must have an album cover that is displayed in the UI
- **Search:** Implement a search function based on the song title, album title, genre, or artist name. Search results should include public playlists as well.
- **Like:** Users should have the ability to Like songs and playlists. A selection of liked songs and playlists should appear on a user's personal profile page.
- **Playlists:** Allow users to create and view playlists. A playlist is a list of songs meant to be played in a specific order. Allow users to repeat the songs on a playlist or shuffle it. A playlist can be Public or Private.
    - Note that each user has a default "Liked Songs" playlist. Any song that is Liked by the user is automatically added to this playlist.
- **Audio Files**: Download music files and use the app's audio player to play music. Display each song's attributes (such as title and genre) in addition to the songs
- **Home Page/Timeline**: A default welcome page should be displayed to a user after logging in.

4. Create a Database to Store The App's Data Persistently

The Server's database plays a central role in storing essential data. You are allowed to use a SQL-based database or a NoSQL database (such as MongoDB). Remember to add the necessary JDBC (Java Database Connectivity) dependency to your project.
The database must contain the following data:

- **Genre**: This entity will store information about different music genres such as rock, pop, classical, hip-hop, jazz, etc. You should include attributes such as genre ID, name, description, and possibly a list of related genres.
- **Music**: This entity will store information about individual songs or tracks. You could include attributes such as track ID, title, artist, album, genre, duration, release date, and popularity.
- **Album**: This entity will store information about music albums. You could include attributes such as album ID, title, artist, genre, release date, and popularity. Additionally, you may want to include a list of tracks associated with each album.
- **Playlist**: This entity will store information about user-created playlists. You could include attributes such as playlist ID, title, description, creator, number of likes, and a list of tracks included in the playlist.
- **User**: This entity will store information about users who create and interact with the platform. You could include attributes such as user ID, username, email address, password, profile picture, and a list of playlists created or liked.
- **Artist**: This entity will store information about musical artists. You could include attributes such as artist ID, name, genre, biography, social media links, and a list of albums or tracks associated with each artist.

It's important to ensure that these entities are properly normalized and related to one another within the database. For example, a song should be associated with an album and an artist, while an album should be associated with a genre and potentially many songs. Similarly, a playlist should be associated with a user and potentially many songs.

Feel free to add other attributes and entities to your database based on the project’s architecture and implementation. There is no correct way to design a database schema, but try to avoid data redundancy as much as possible.

Additional Features

(These tasks are mandatory for groups with 3 members, but are considered bonus tasks for smaller groups)

- **Dynamic Timeline**: A timeline consists of a selection of liked artists, albums, and playlists. A dynamic timeline updates itself based on a user's activity.
- **Friend System:** Allow users to add friends and view each other's profile page. On a user's profile page, a list of their friends should be visible to other users.
- **Like albums and artists**: Users should be able to like albums and follow artists. Display a user's liked albums/artists on their profile.
- **Display Song Lyrics**: Store the lyrics for each song available. You should set an option in your application to display the lyrics.
- **Songs with multiple artists**: A song may have multiple artists. You should handle these cases both in the database and application.

Notes

- Make sure to use a Package Manager for your project (either Maven or Gradle)
- Before starting your work on the GUI, you should first implement a menu in the command line to test your code and then build the graphical interface on top of the terminal-based interface.
- Spotify is mostly known as a streaming app, meaning that it downloads music files in small packets and plays them in real-time. Your app doesn't need to support this feature, you can fully download a file and then start to play it.
- Add some sample data before presenting your code. Artists, genres, albums, and songs must be added manually either through a DBMS or by code.
- Working together as a team is essential for the successful completion of this project. By dividing tasks and responsibilities, you can maximize efficiency and ensure timely progress. Note that GitHub has many features developed to enhance teamwork, such as creating Issues.
- Communicate with your project mentor and report your progress to them. You must contact your mentor after implementing each phase of the project.

Bonus Features

Below you can find a collection of Additional features you can implement in your project. While this is our recommended list, you are free to add more optional and challenging features at your discretion. In other words, **your creative effort won't go unnoticed.**

1. **Data Encryption:** Use an encryption method to encrypt the data transferred between the server and the client.
2. **Shared Playlist:** Allow multiple users to manage the same playlist. The owner must authorize each user before gaining access.
3. **Playlists Created by Spotify:** Develop your app so that it can suggest playlists to users. For example, it can offer users a playlist of their most repeated songs.
4. **Music Videos:** Create a section in the app where users can view music videos for some songs.
5. **Access Limited Version:** Develop your app so that a user without an account can have limited access to the app. For example, any user could be able to listen to 5 songs before creating an account.
6. **Smart Shuffle:** Implement a smart shuffle system so that the app can shuffle music based on a criteria, such as playing songs with different genres.
7. **Podcasts:** Create a section where artists or producers can post their podcasts in the app. Store a description of the podcast and display all its episodes.
8. **Avoid Certain Songs/Genres/Artists:** Add this feature to your program so that the user can avoid a specific type of music (such as a specific genre), which means that type of music will never be shown to them.
9. **Add to Queue:** Develop a queue system for the app. While playing a song, users should be able to add a number of songs to play immediately after the current one ends. Display the song queue in UI. Implementing a drag & drop system for changing the order of the songs in the queue is optional.
10. **Share Music:** Implement a sharing system where users can share their favorite music with their friends.
11. **Chat With Friends:** Implement a chat system so that users can communicate with their friends and share their favorite content.
12. **Leave Comments for Songs/Albums:** Implement a commenting system where users can comment on the desired music or rate it.

13. **Android:** Develop your application for Android systems so that you can run it on an Android device.

Project Phases

1. Phase One:

In the first phase of the project, You must design the application's overall structure. Complete the following tasks before moving on:

- Determine your application's design pattern and structure.
- Determine which Database System to use and design the schema for each table.
- Establish a connection between the server and the client using your API.

2. Phase Two:

In this phase, you must implement all of the basic features of the app for both the client and the server. These features include account creation, a profile page, a search feature, playlist creation, artist pages, and the ability to like songs and playlists.

3. Phase Three:

In this phase, you must wrap up the main section of the project. Implement the ability to store files on the server, download and play music, display album covers, and adjust audio features such as changing the volume.

4. Phase Four:

The final phase of the project is when you're allowed to implement the bonus or additional features that you've decided to add to the project. Your app must offer every mandatory feature before you can move into this phase.

Useful Tools & Packages

You can find a selection of helpful libraries, apps, and tools here. You are not required to use any of them, but they may prove useful in your project.

- **UUID:** The Universally Unique Identifier class can be used to generate IDs for objects that need to be uniquely identified. Note that using a UUID for every class might not always be the best approach.
- **Scene Builder:** A visual layout tool that allows developers to design a JavaFX UI by dragging and dropping components onto a scene, and generating FXML markup code that can be used in Java applications.
- **JLayer:** JLayer (JavaLayer) is a Java library for decoding and playing MP3 audio files. It supports features like seeking, volume control, and playback control.
- **mp3agic:** mp3agic is a Java library for reading and editing MP3 files. It allows you to read MP3 metadata (tags) such as title, artist, album, and cover art. It also provides functionality for adding, modifying, or removing tags from MP3 files.
- **Gson:** Gson is a Java library for converting Java objects to JSON format and vice versa. It allows developers to serialize and deserialize Java objects to JSON strings and parse JSON strings back into Java objects effortlessly.

Evaluation

- Your code should compile and run without any errors.
- Your code should be well-organized, readable, properly commented, and should follow clean code principles.
- Your Database should be well-structured and organized.
- You should use Git for version control and include meaningful commit messages. Utilize as many branches as necessary.
- You will be required to present your code to a team of judges at a later date for the final evaluation. All members of the team must have sufficient mastery over the project.

Submission

- Push the project's code to its repository on GitHub
- Merge your development branches with the main branch of the project

The deadline for submitting your project is exactly one week after your final exam, which is **Wednesday, June 21 (31st of Khordad)**. Good luck and happy coding!