Initial Design Document

for

Music Streaming System

Version 2.0 approved

Prepared by Group 5

2324II INT2208E 23, VNU-UET

April 9, 2024

Revision History

Date	Version	Description	Author
April 16th 2024	0.1	First draft with section titles	Nguyễn Đức Khánh
April 23th 2024	1.0	Add subsection details: 1. Introduction 2. System Architecture 3. Sequence Diagrams 4. Class Diagrams 5. User Interface Prototype 6. Database Diagrams	 Nguyễn Đức Khánh Ngô Lê Hoàng Trần Thế Mạnh Nguyễn Đức Khánh Nguyễn Đức Khánh Nguyễn Đức Khánh Ngô Lê Hoàng
April 28th 2024	1.1	Complete all section in detail 7. Data Structures and Algorithms 8. APIs and Dependencies	 Ngô Lê Hoàng Ngô Lê Hoàng
May 7th 2024	2.0	Fix sequence diagram and class diagram	Ngô Lê Hoàng Nguyễn Đức Khánh

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to present the initial design for a comprehensive Music Streaming System. This system is envisioned as a robust, user-friendly platform that will enable music enthusiasts to stream their favorite tracks, discover new music, and curate personalized playlists. The document aims to provide a detailed roadmap for the development team, outlining the system's architecture, components, and their interactions. It will serve as a blueprint during the development phase and a reference point for future system enhancements and maintenance.

1.2. Scope

The Music Streaming System is a web-based application that will allow users to listen to music from a vast library of tracks spanning various genres, moods, and eras. Users will be able to search for songs, albums, or artists, create and manage their own playlists, and explore music based on their listening history and preferences. The system will also include features for user authentication, music recommendation, and social interaction among users.

2. System Architecture

2.1. Component diagram (overview)

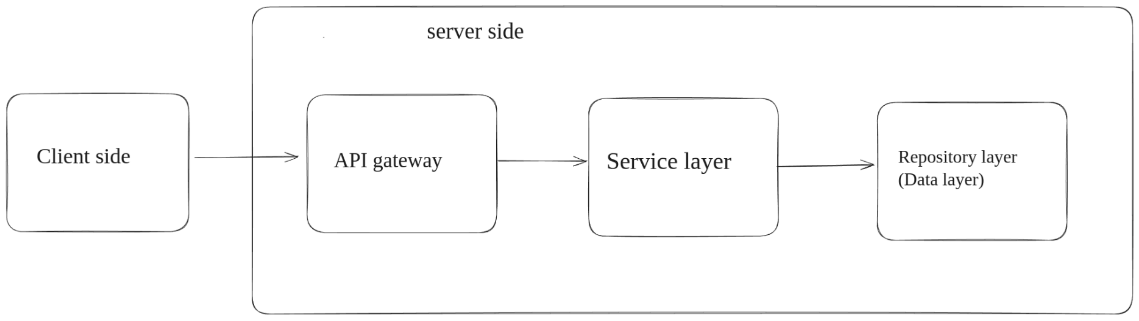


Figure 1: Overview component diagram

A client-server architecture made up of clients, servers, and resources, with requests managed through REST API

The client sends requests with methods like GET/POST/PUT/DELETE, etc., to the server-side, and the server-side responds with JSON format. On the client side, you can reference the Swagger API document and directly interact with it for testing.

Client-side manages UI rendering, user interactions, and executes scripts like JavaScript for dynamic behaviors. Data is fetched from the server side.

The API gateway is responsible for routing requests to the correct service for processing, then aggregating and organizing the results (including exceptions and errors) returned from the service, and sending the response back to the client side. Additionally, the API gateway manages rate limiting and authentication.

Service layer handles the business logic for the application:
- Data validation, such as checking if the requested song ID is valid or if the size of the uploaded track file exceeds 50MB
- Process handling, for example, when adding a new song: first, checking if the album and artist of the track exist, then verifying the size of the uploaded music file, followed by storing the information in the database and saving the mp3 file to static storage.
- Interaction with other system components or external APIs, such as sending requests to the data layer to store track information.

Repository layer (Data layer) is responsible for directly interacting with the database to manage data for the application. Unlike the service layer, which only calls abstract APIs, the repository layer implements details such as creating sessions, transactions, etc.

2.2. Component diagram (Detail)

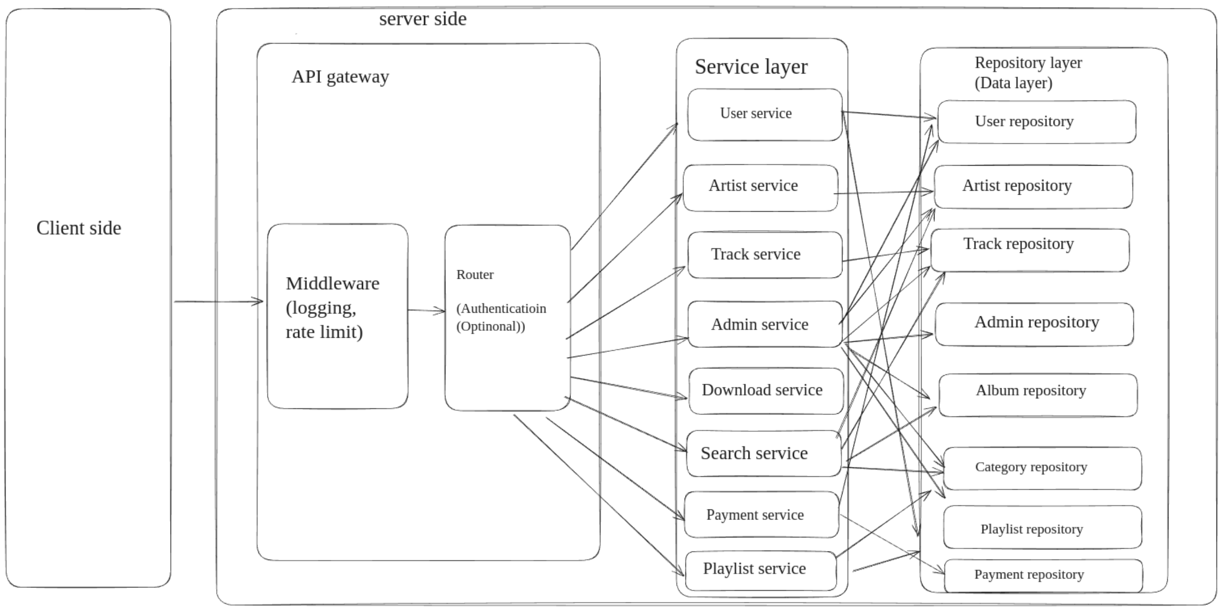


Figure 2: Detail component diagram

Functions like logging and rate limiting are embedded within middleware, meaning that these middleware can be applied before, during, or after processing each request sent.

Authentication will be applied to critical APIs to ensure the integrity and security of the application's data. Authentication in the application is implemented using JWT tokens with a refresh mechanism.

The router is responsible for directing requests to the corresponding service and aggregating responses to send to the client side.

Each service can utilize multiple repositories from the repository layer to handle requests. This approach increases the logical coherence of the application and enhances the separation of concerns. (For example, the user service can perform functions related to creating an account, deleting an account, or editing account information by calling APIs from the user repository. Additionally, the user service can also call APIs from the playlist repository to perform functions like creating a playlist for that user's account)

3. Sequence Diagrams

3.1. Create an account

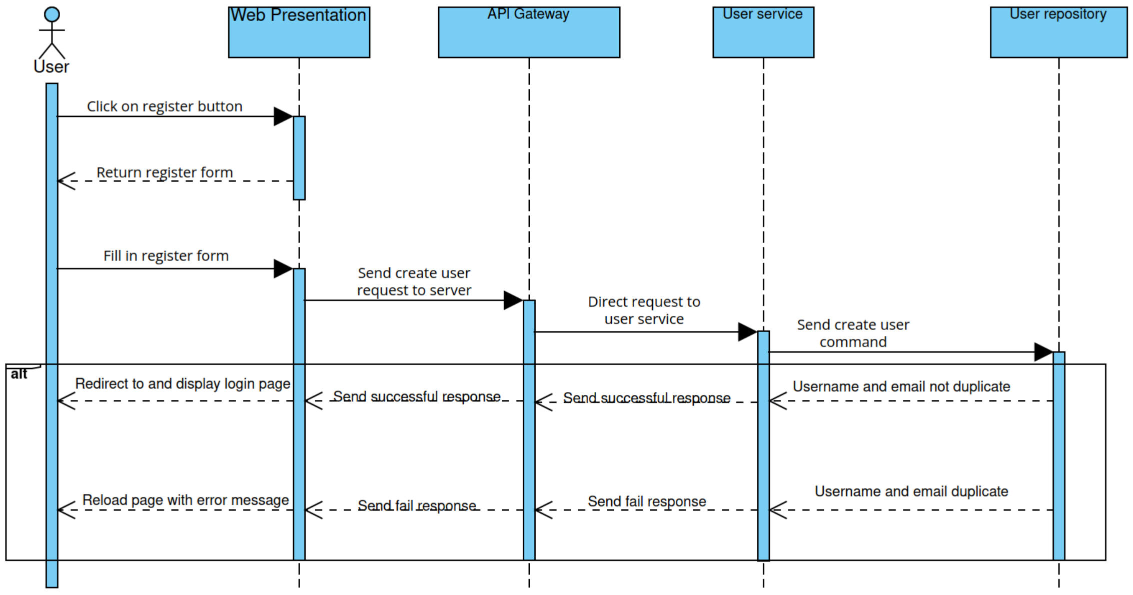


Figure 3: Account registration sequence diagram

3.2. Log in

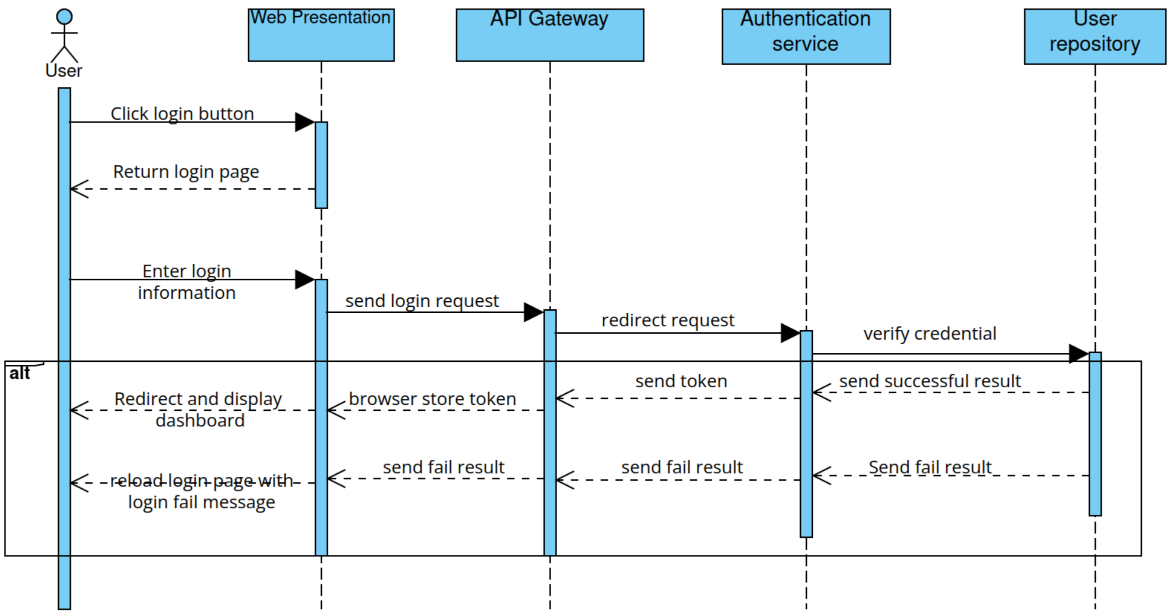


Figure 4: Login sequence diagram

3.3. Play the track

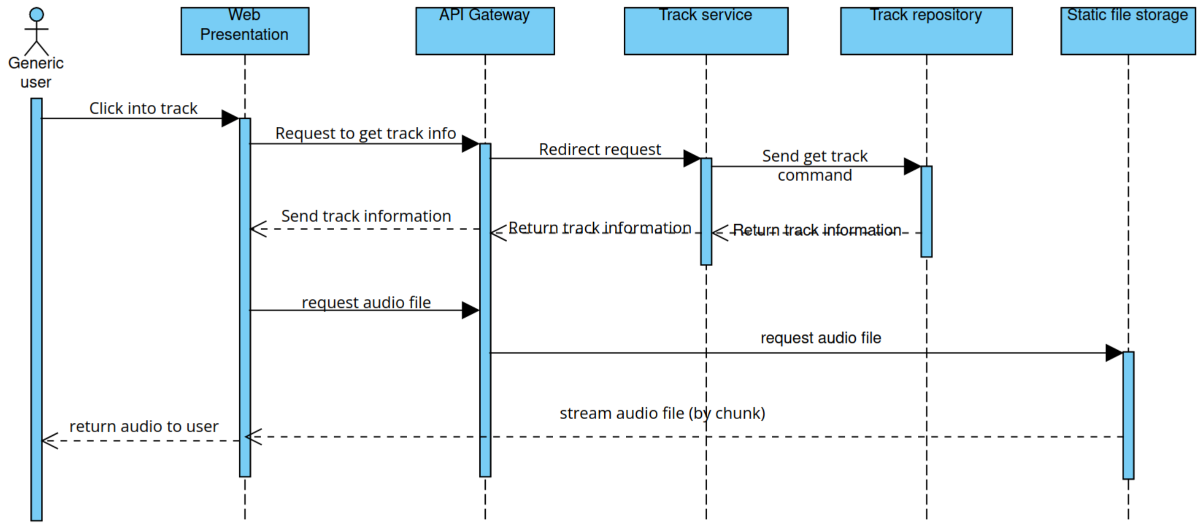


Figure 5: Playing Track sequence diagram

3.4. Search the tracks, artists and genres

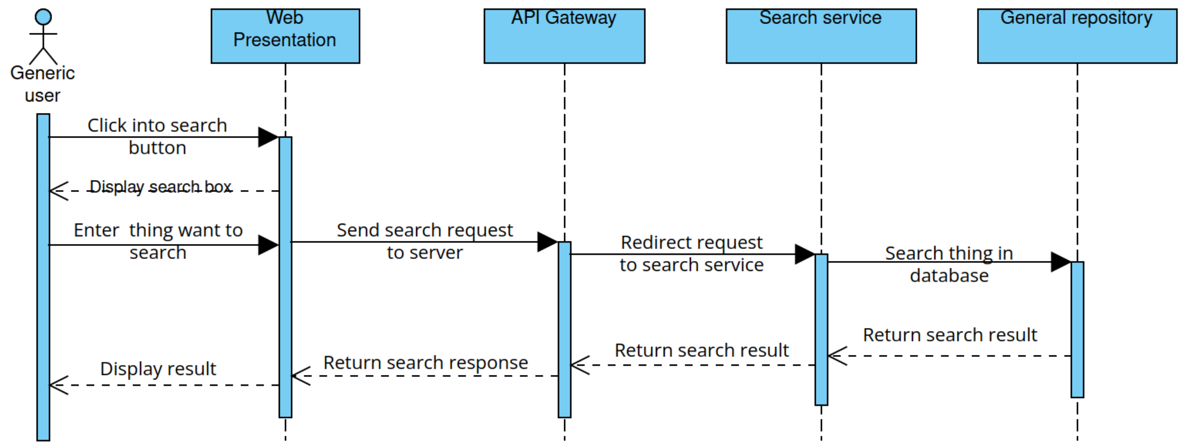


Figure 6: Search Engine sequence diagram

3.5. Download the track

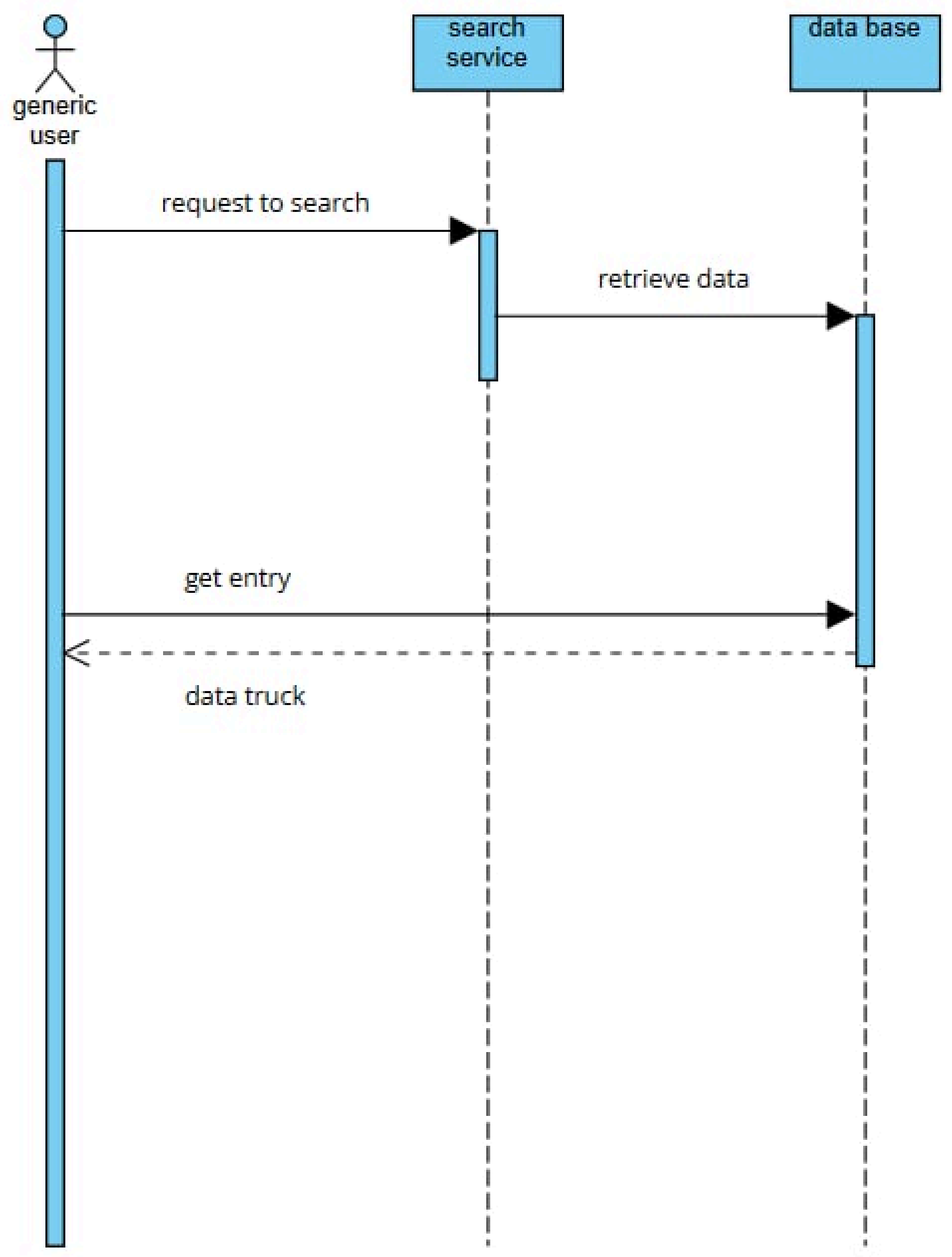


Figure 7: Track download sequence diagram

3.6. Create playlists

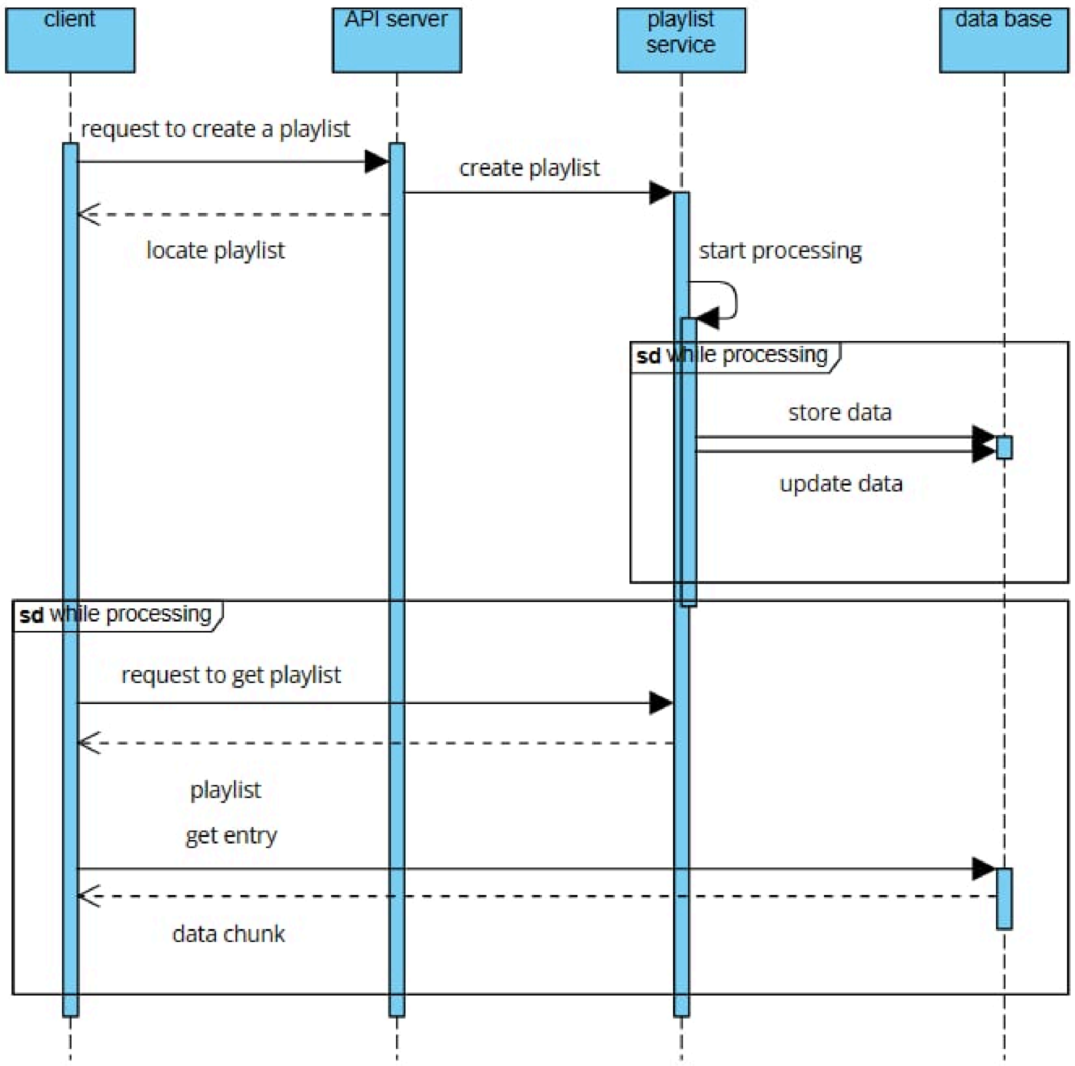


Figure 8: Playlist creation sequence diagram

3.7. Manage playlists

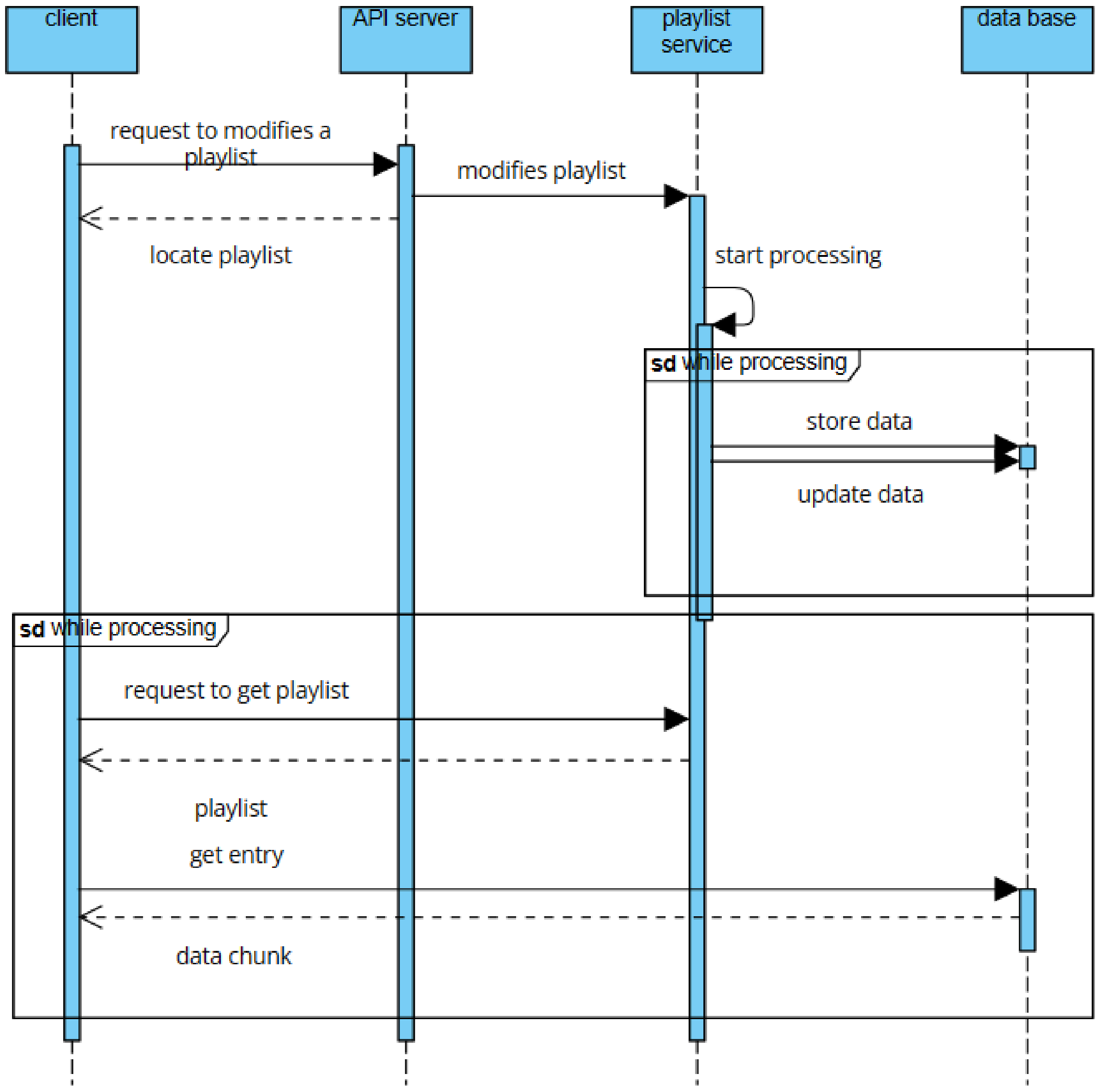


Figure 9: Playlist management sequence diagram

3.8. Upload the track

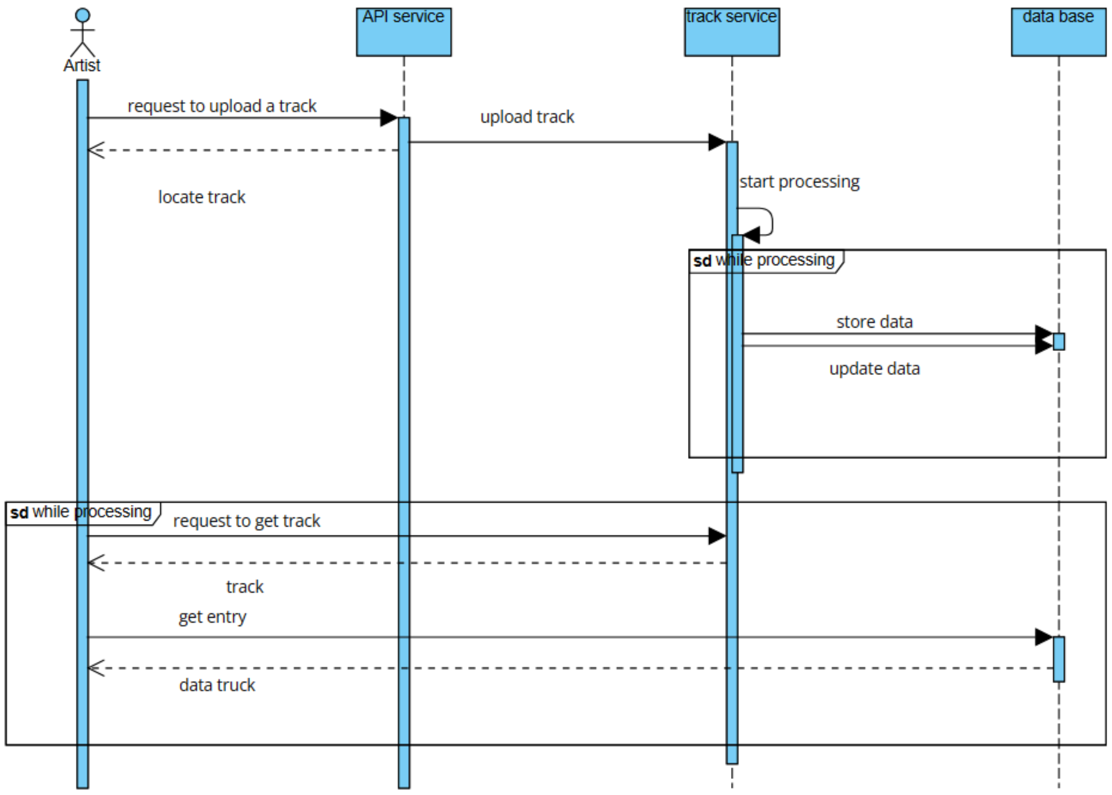


Figure 10: Track Upload sequence diagram

3.9. Manage track

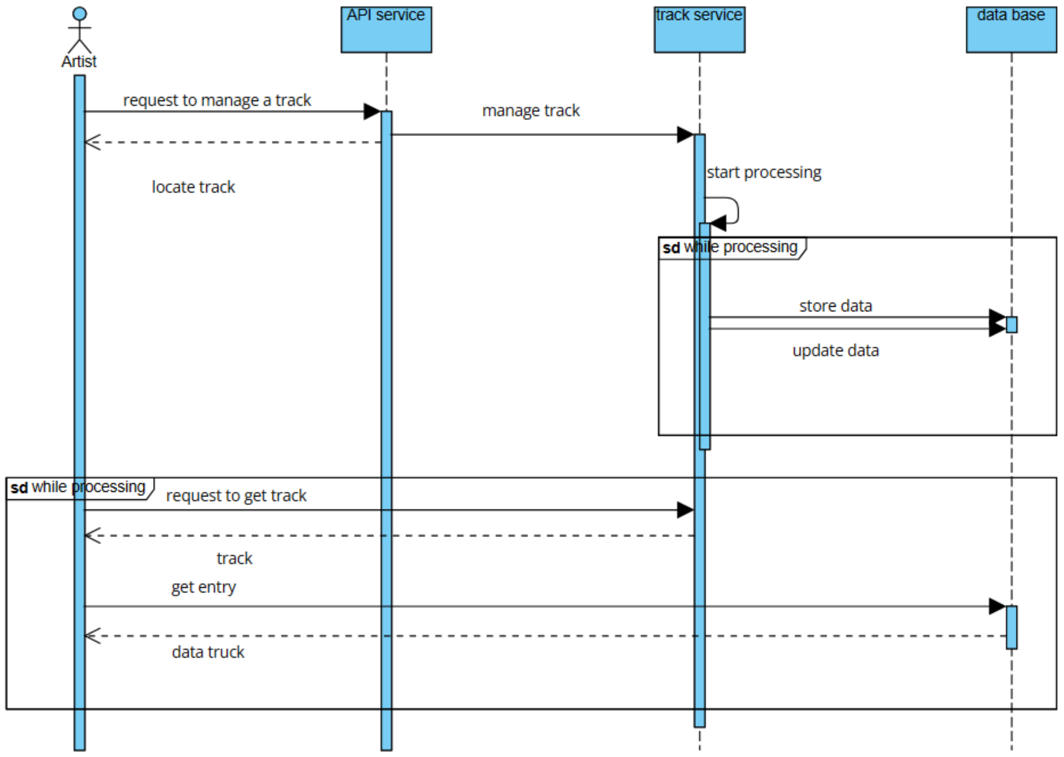


Figure 11: Track management sequence diagram

3.10. Create album

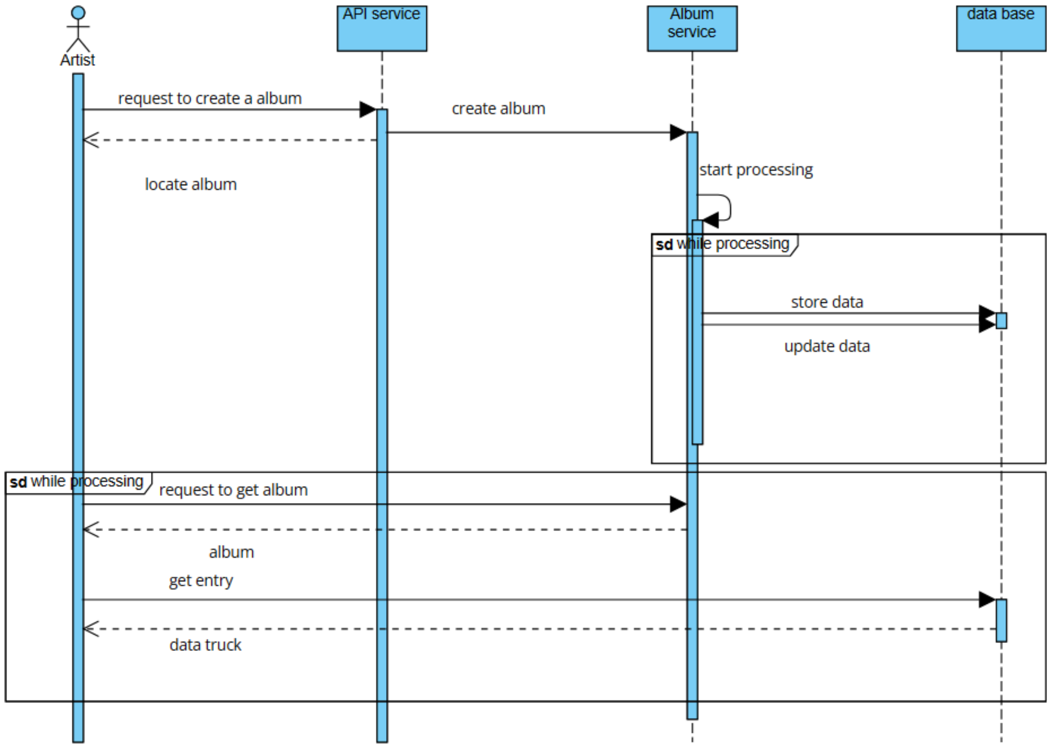


Figure 12: Album creation sequence diagram

3.11. Manage album

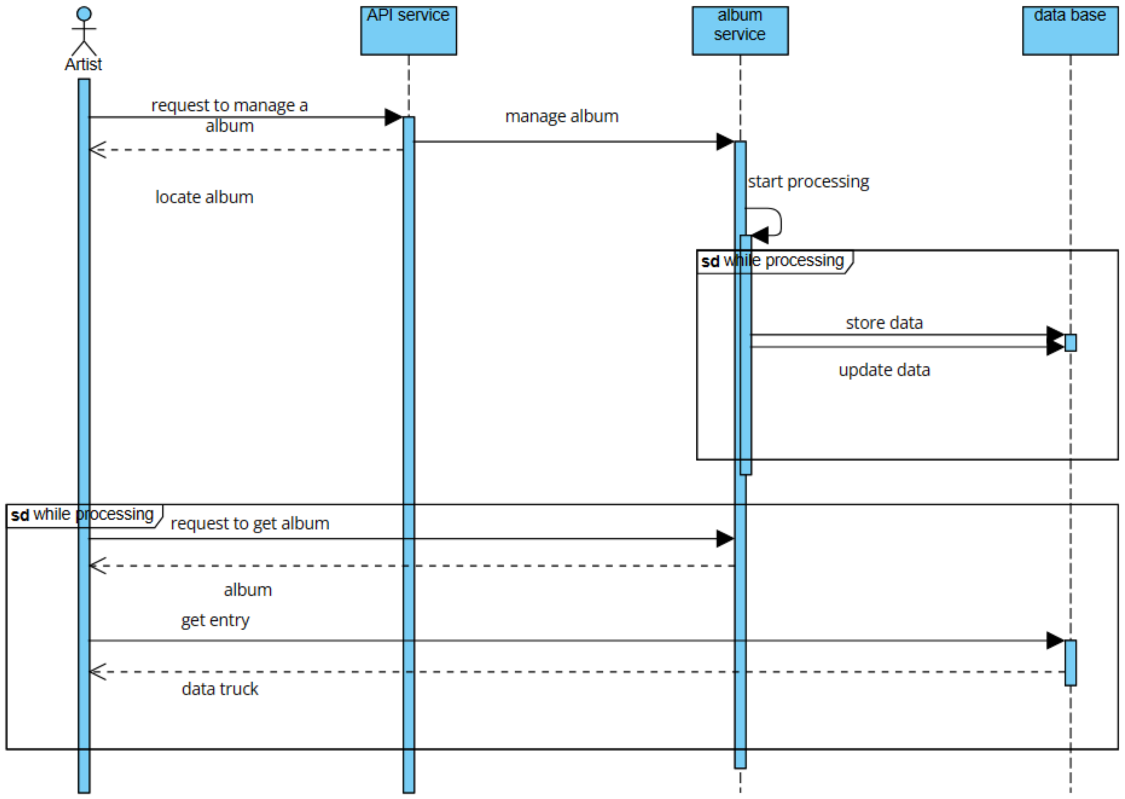


Figure 13: Album management sequence diagram

3.12 Upgrade account

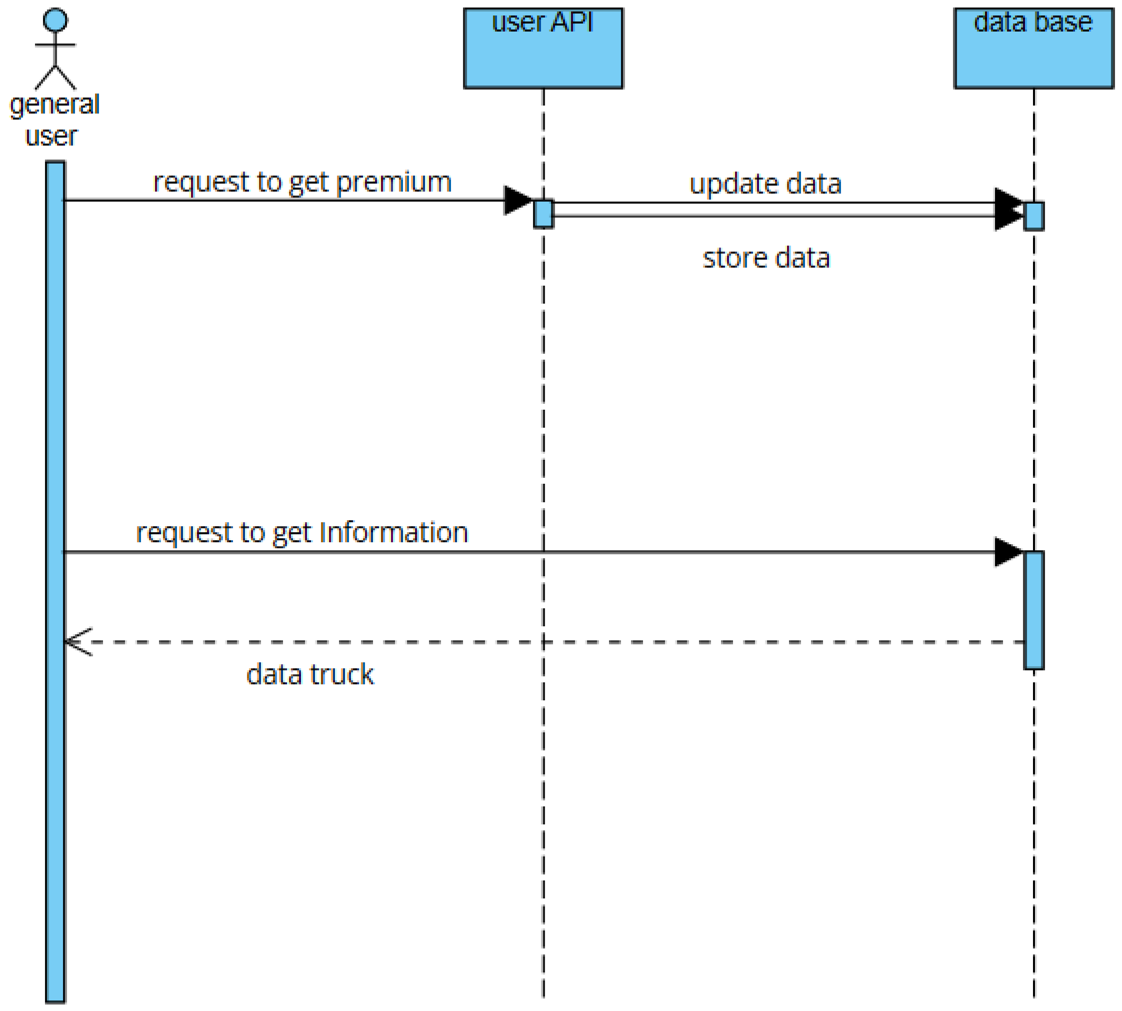


Figure 14: Account upgrade sequence diagram

4. Class Diagrams

4.1. Sign up

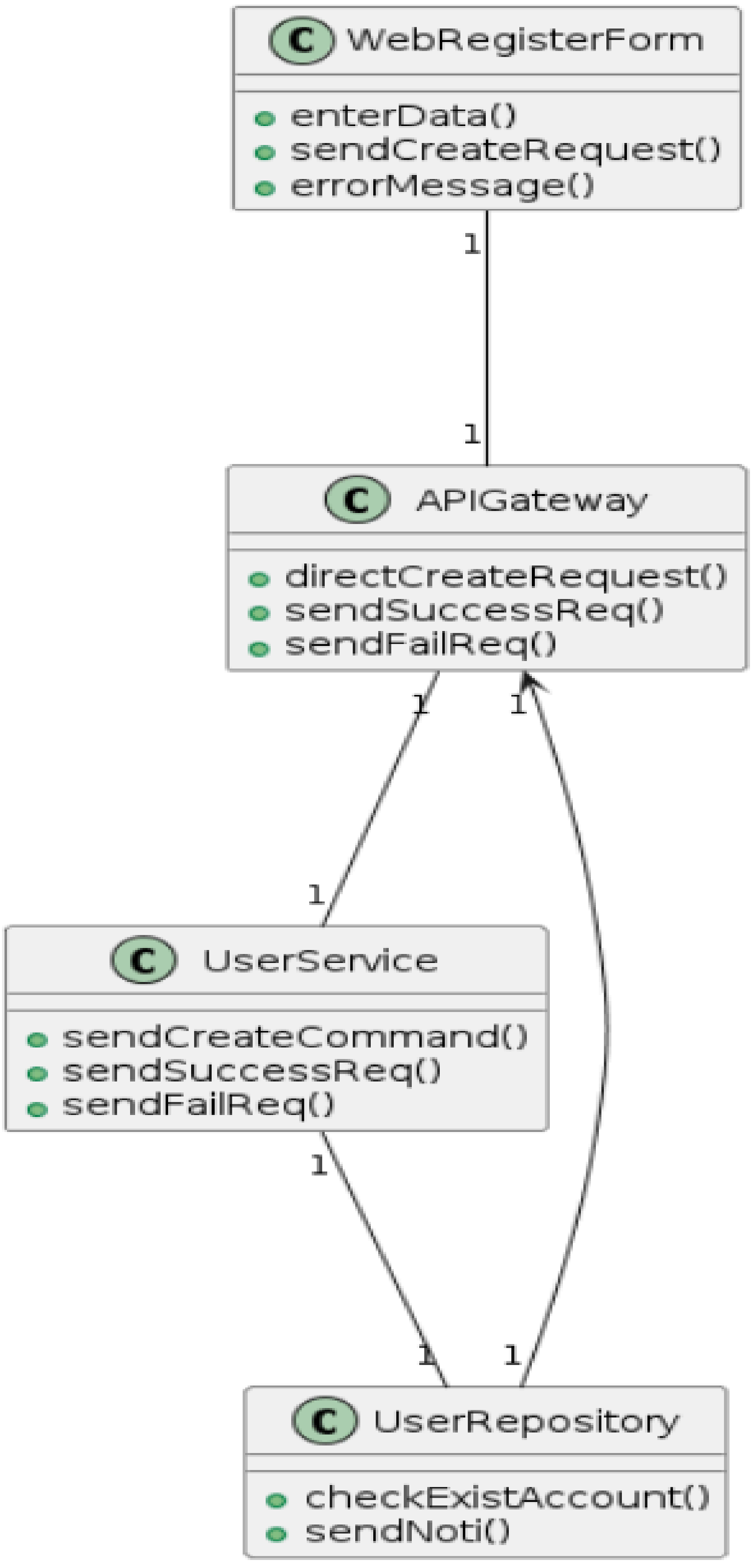


Figure 15: Account registration class diagram

4.2. Log in

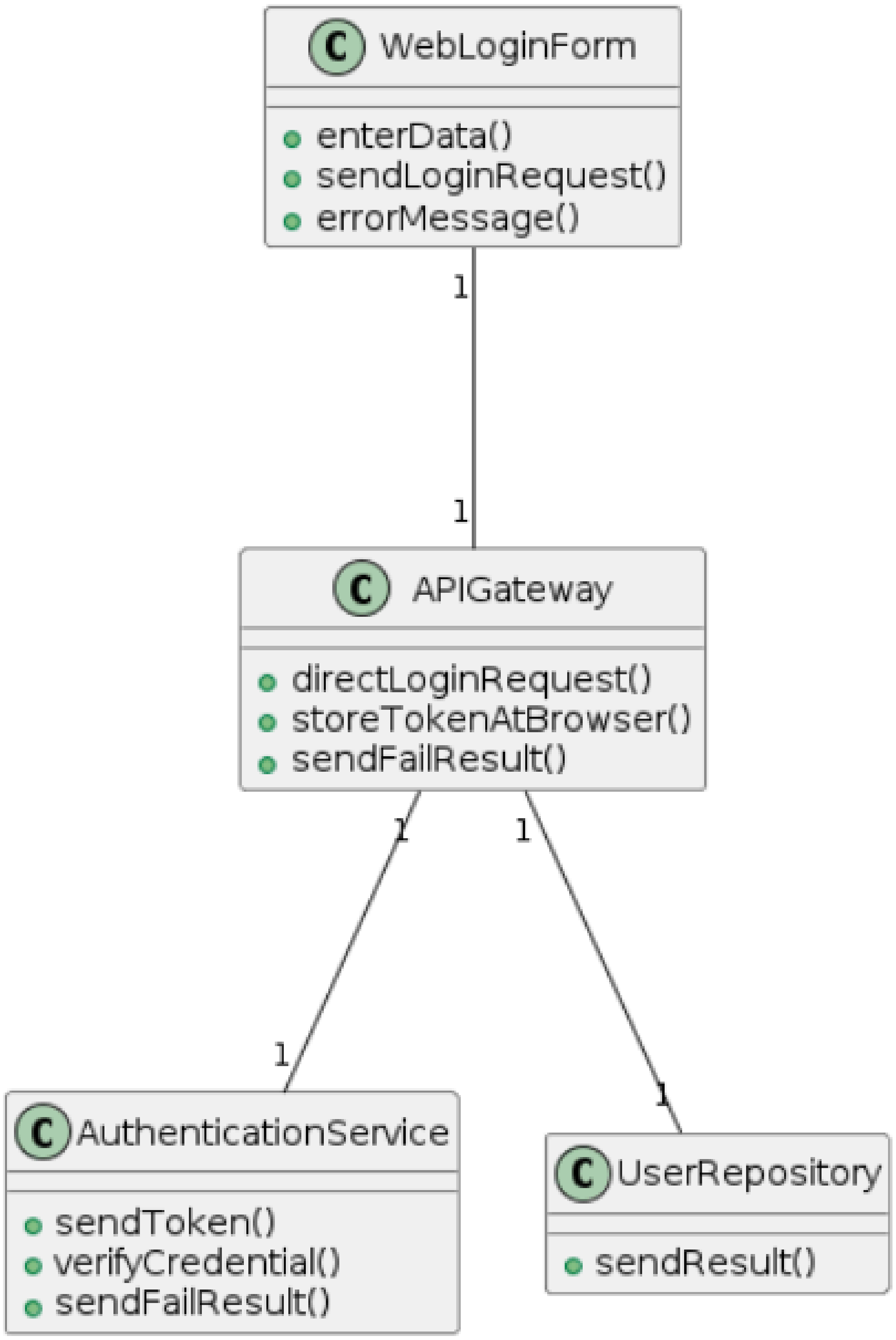


Figure 16: Login class diagram

4.3. Play the tracks

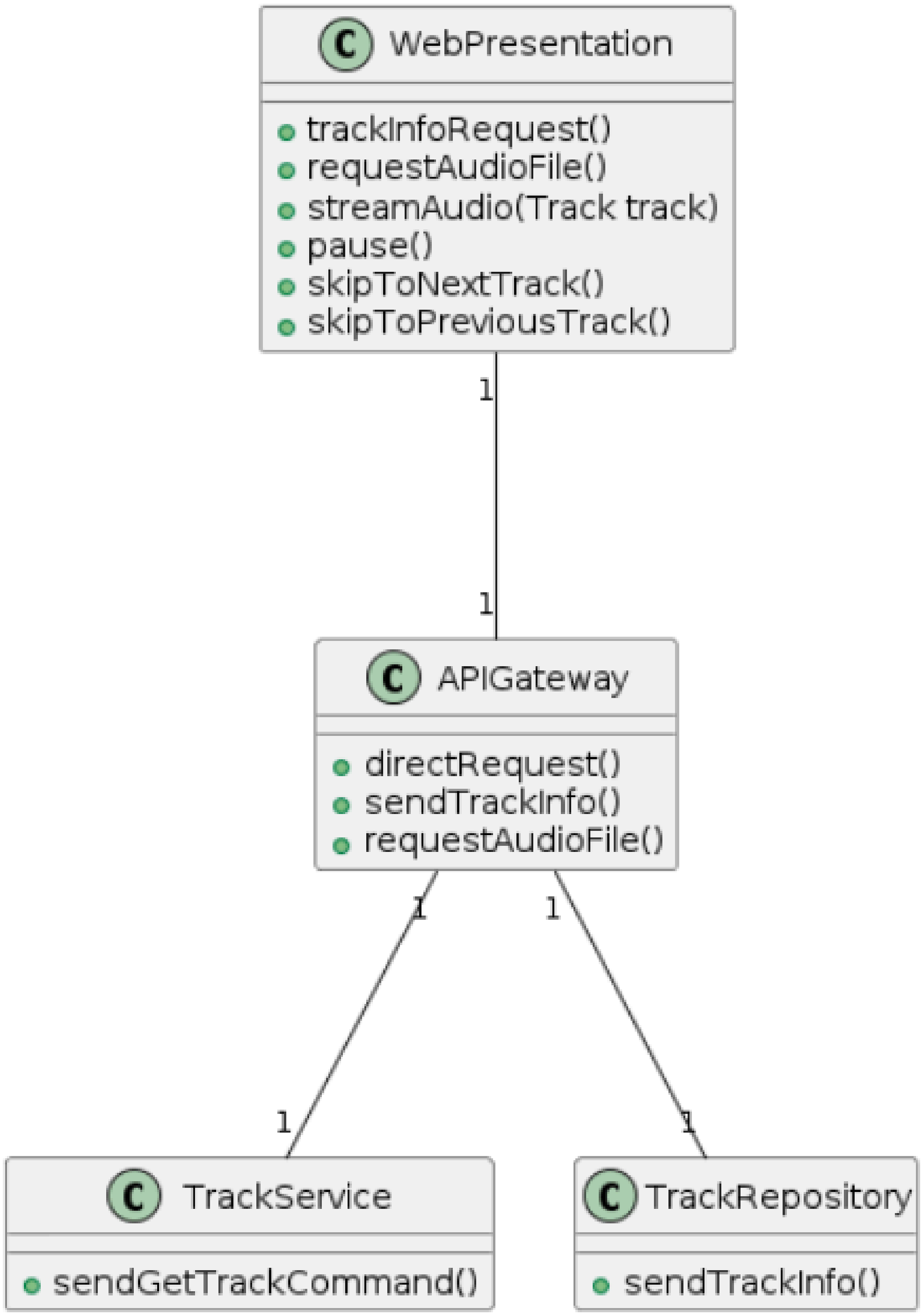


Figure17 : Playing Track class diagram

4.4. Search tracks, artists and genres

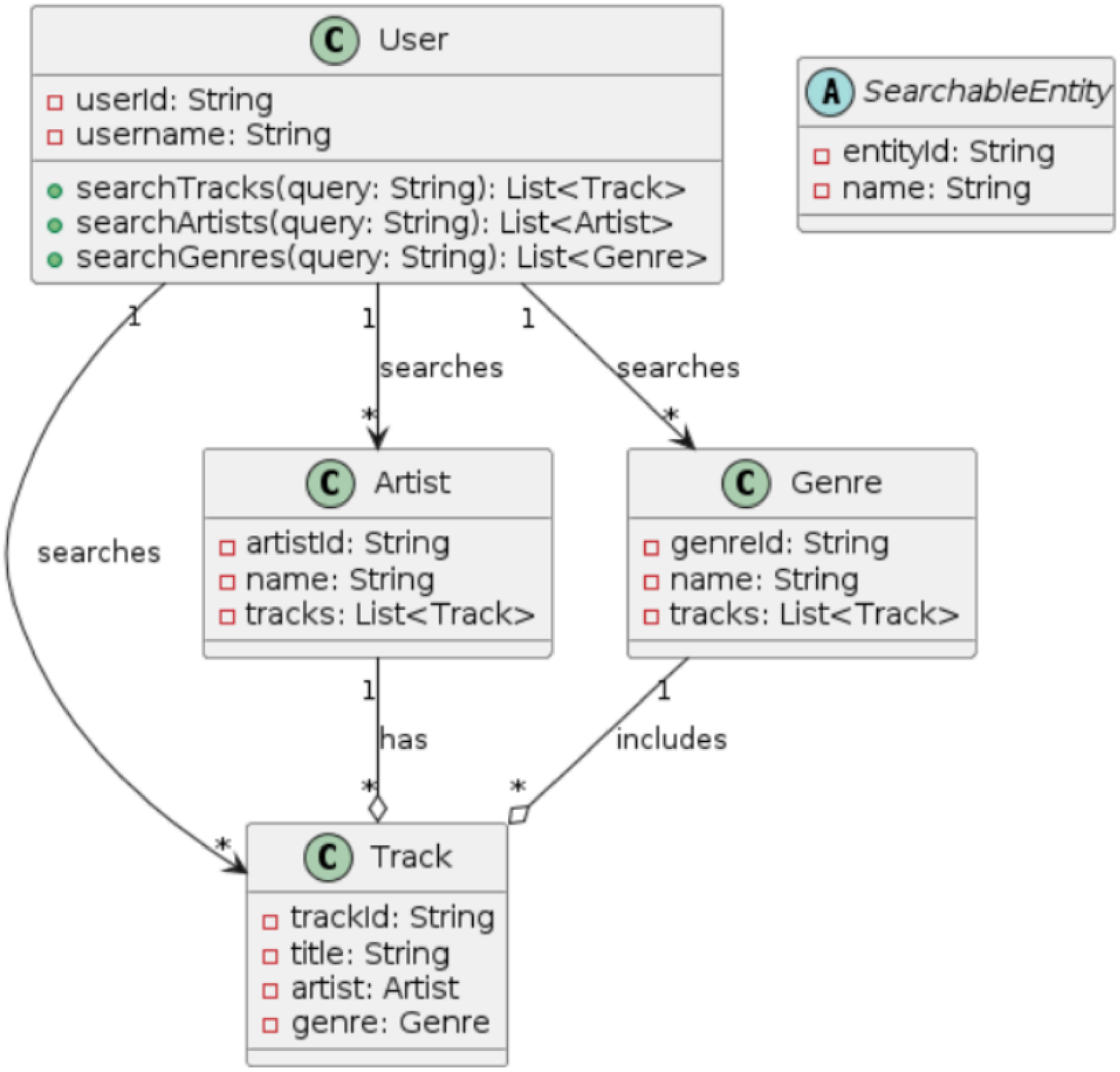


Figure 18: Search Engine class diagram

4.5. Download tracks

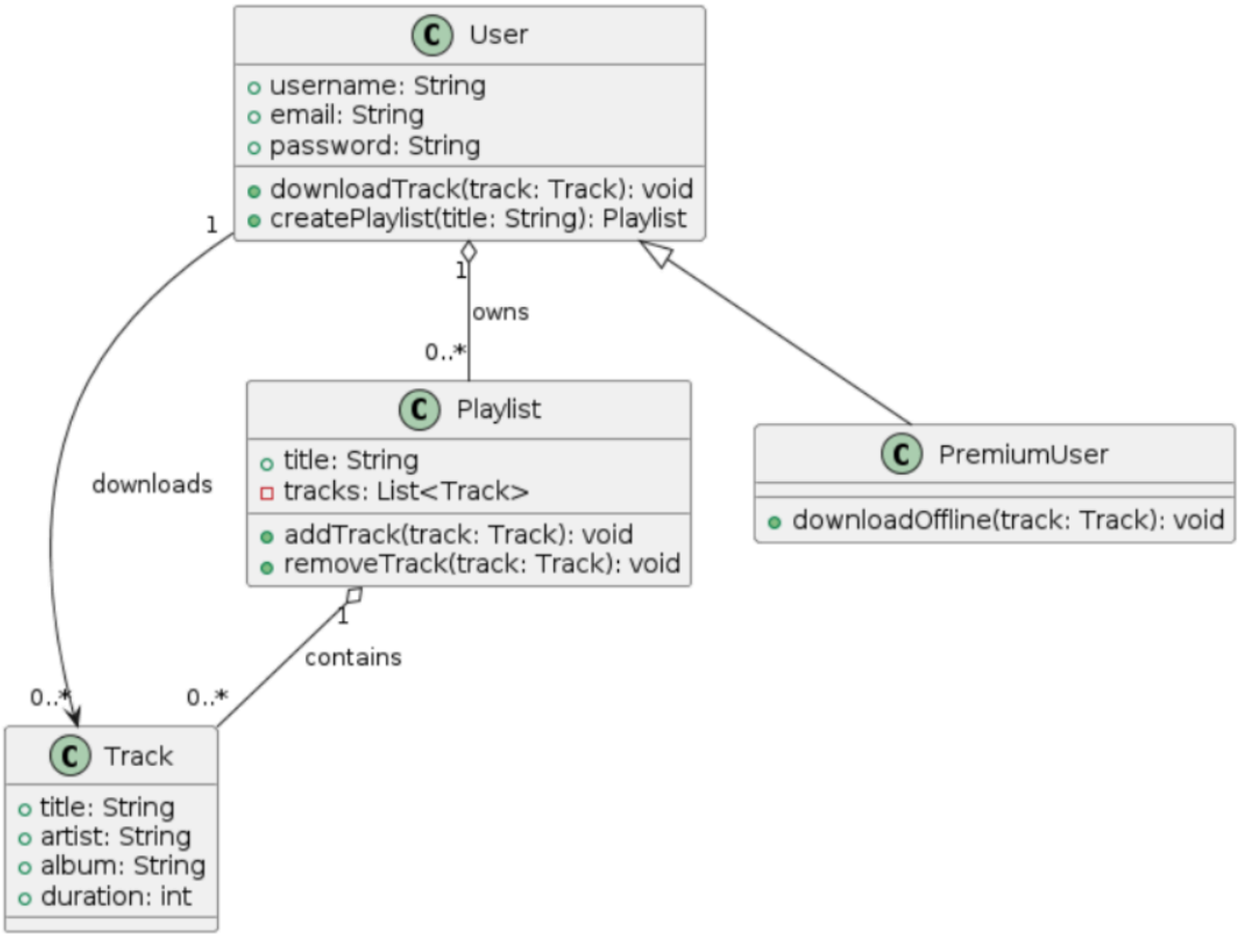


Figure 19: Track download class diagram

4.6. Create playlists

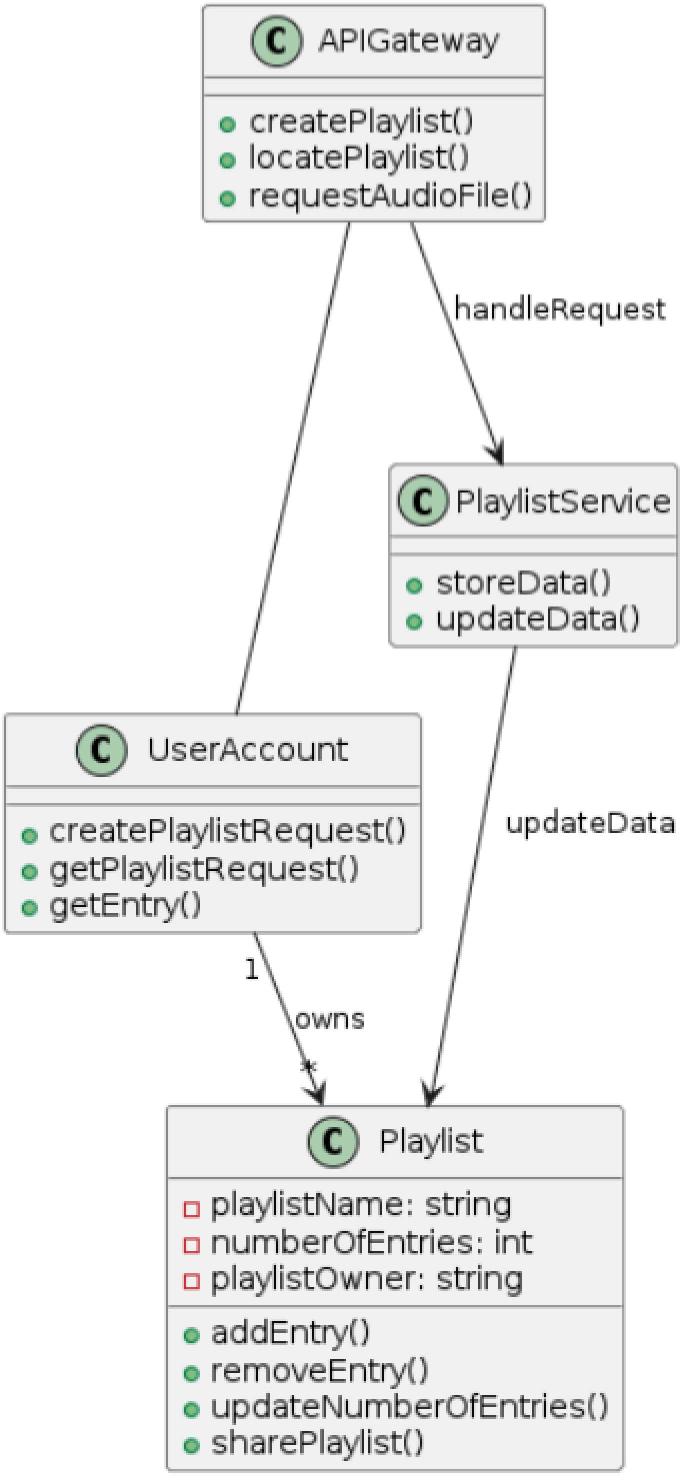


Figure 20: Playlist Creation class diagram

4.7. Upload tracks

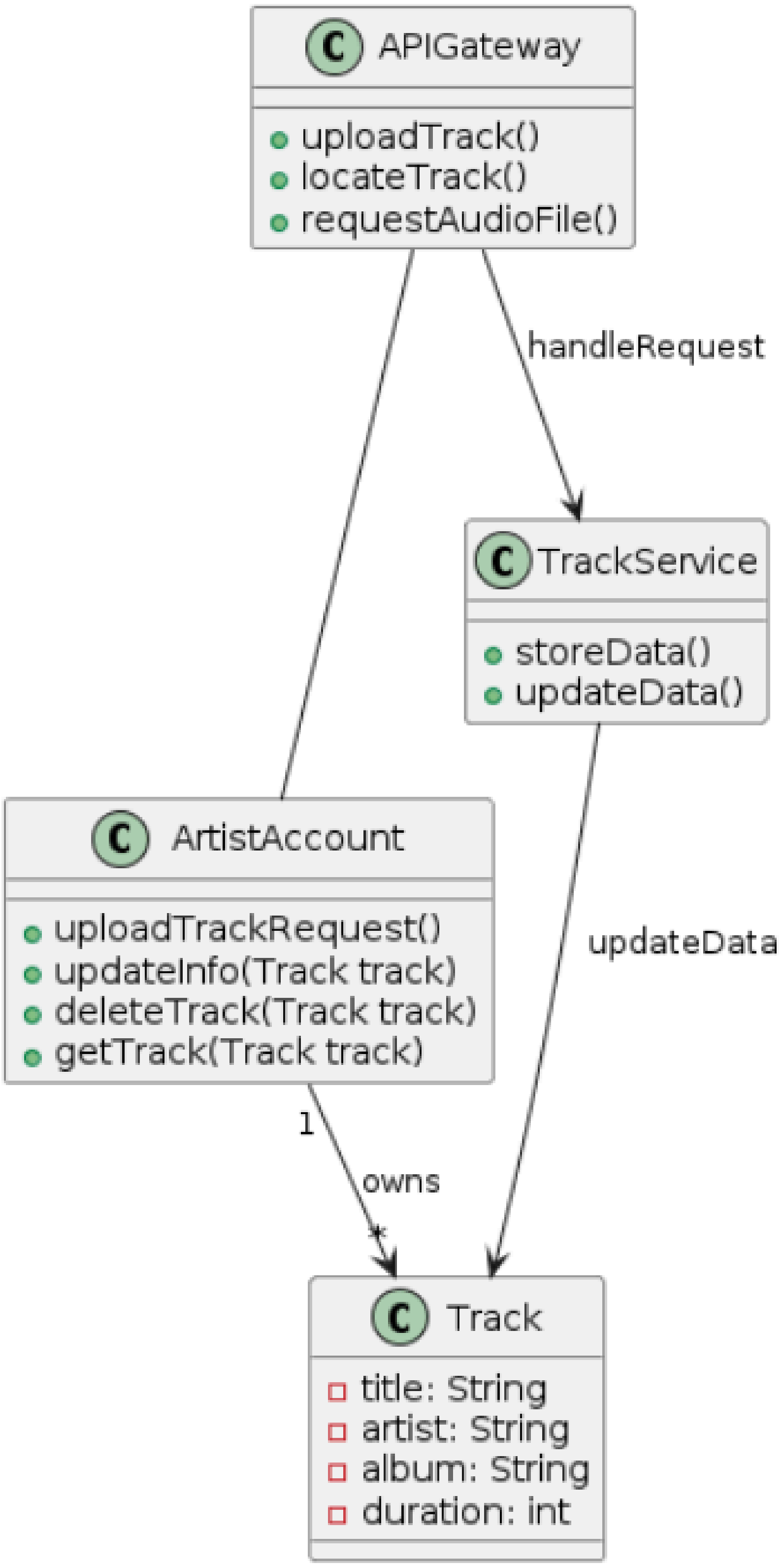


Figure 21: Track Upload class diagram

4.8. Create album

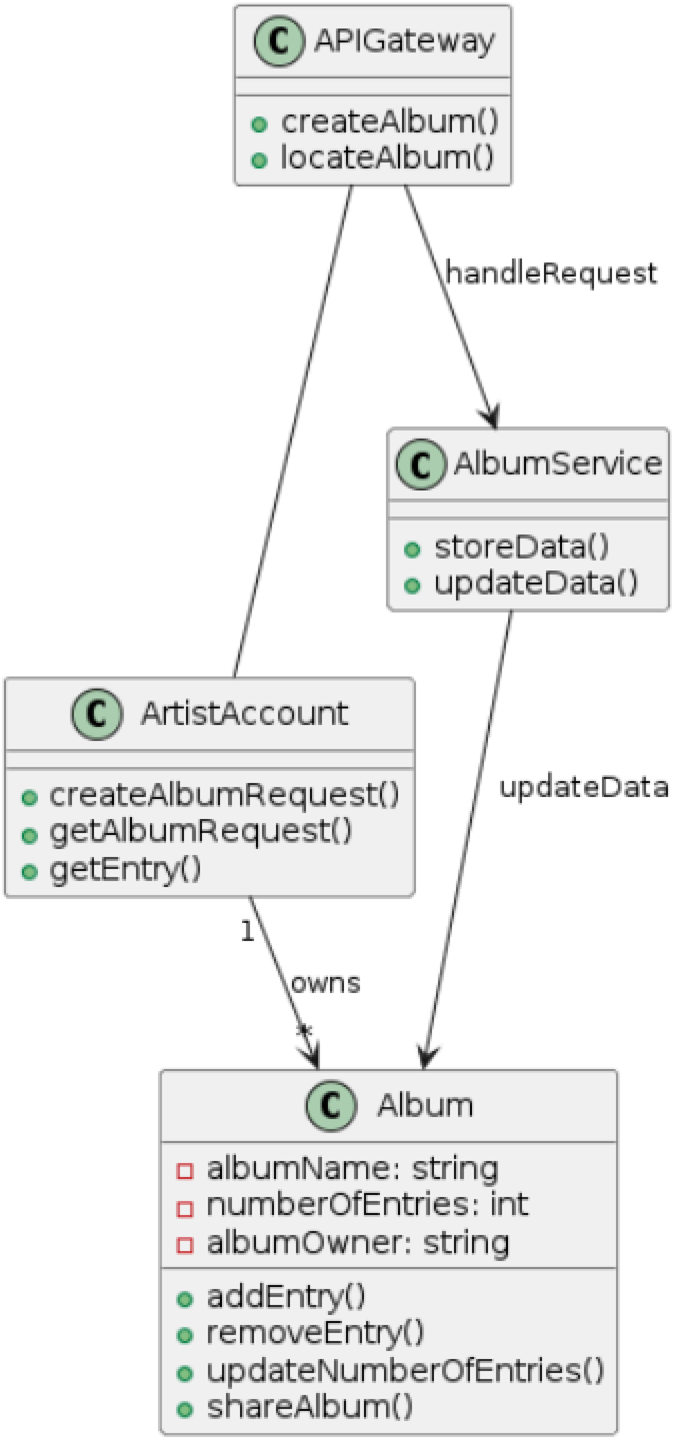


Figure 22: Album creation class diagram

4.9. Upgrade account

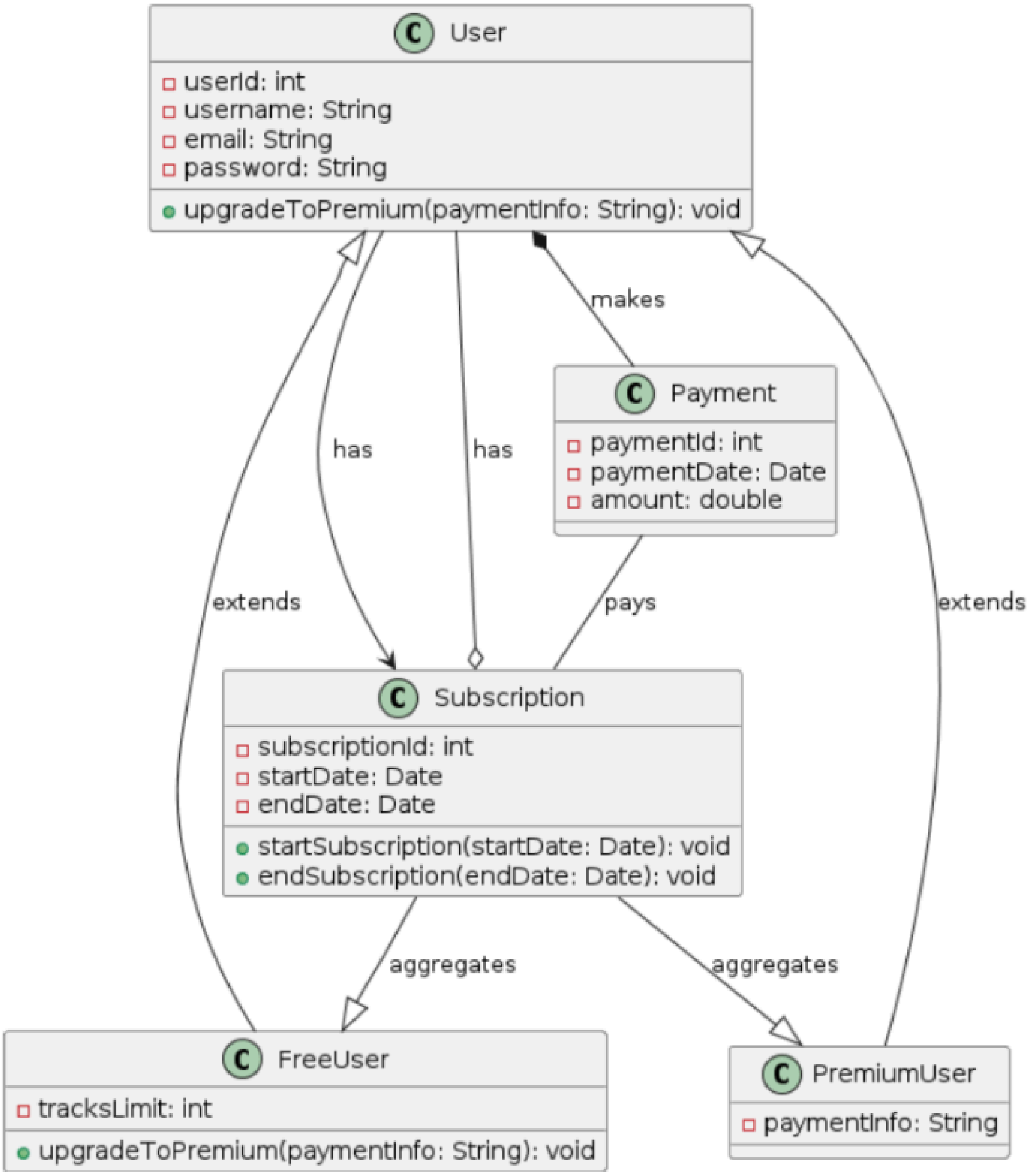
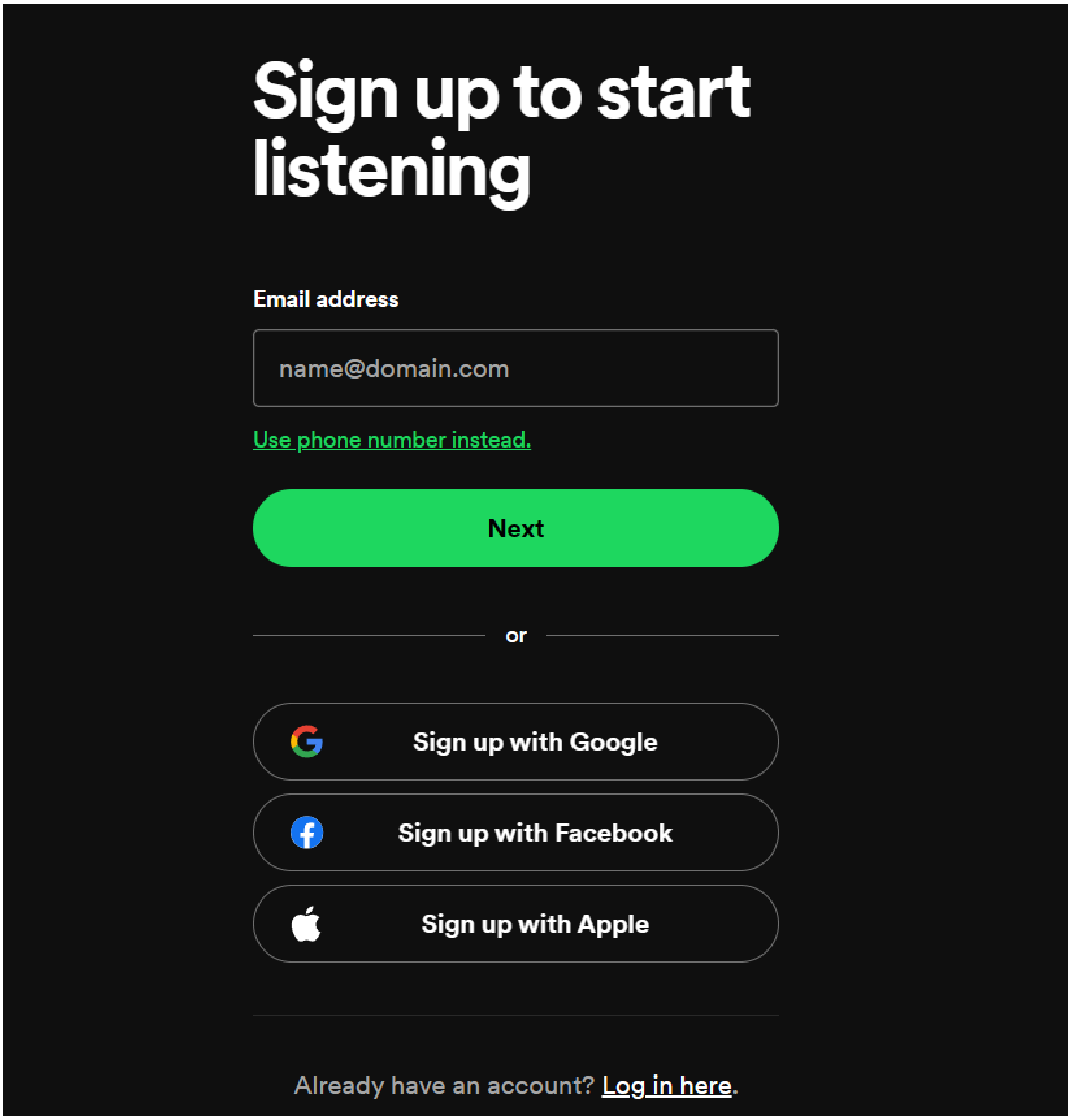


Figure 23: Account upgrade class diagram

5. User Interface Prototype

5.1. Create an account



Sign up to start listening

Email address

name@domain.com

Already have an account? Log in here.

Figure 24: Demo registration form

5.2. Login

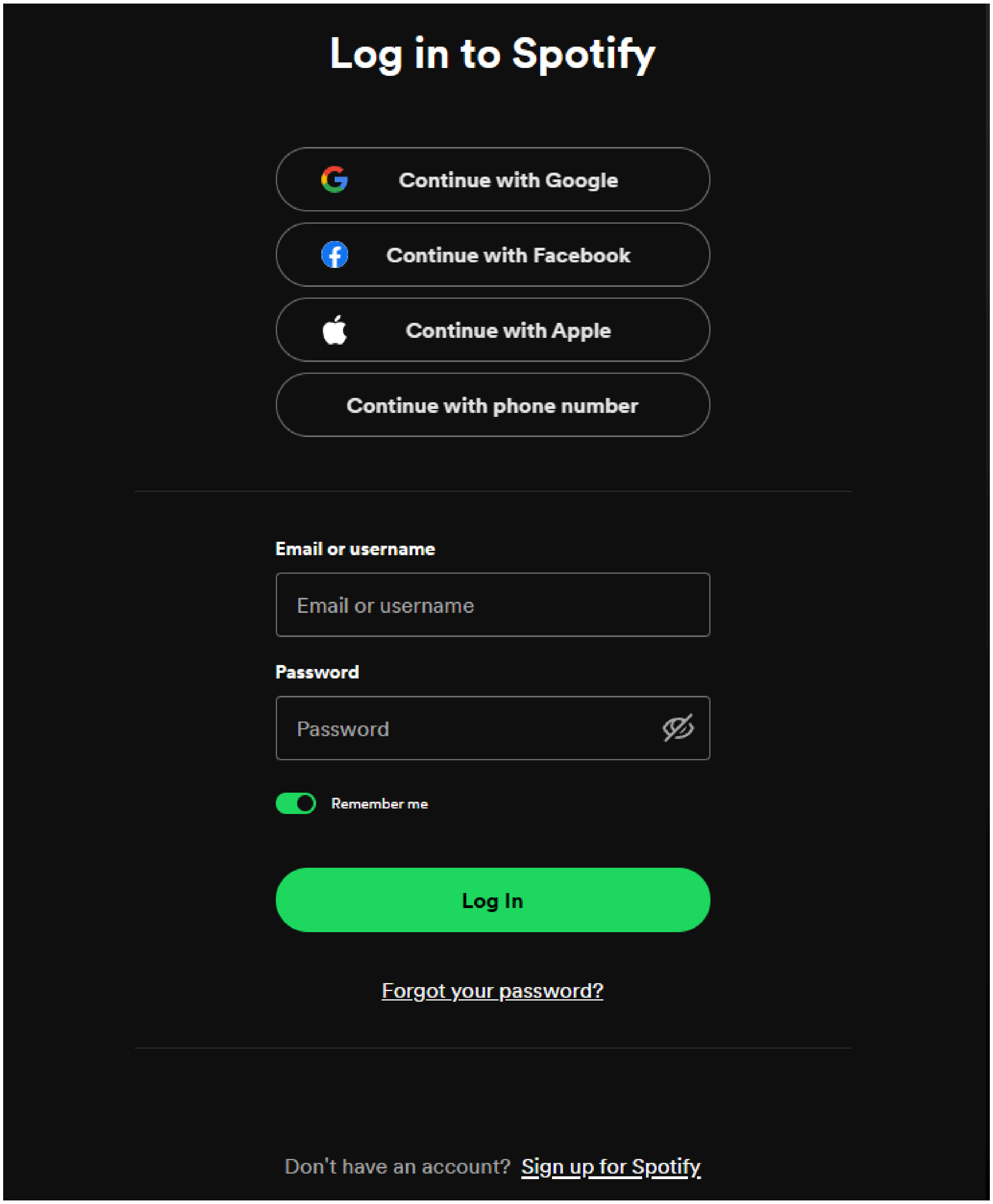


Figure 25: Demo login view

5.3. Search

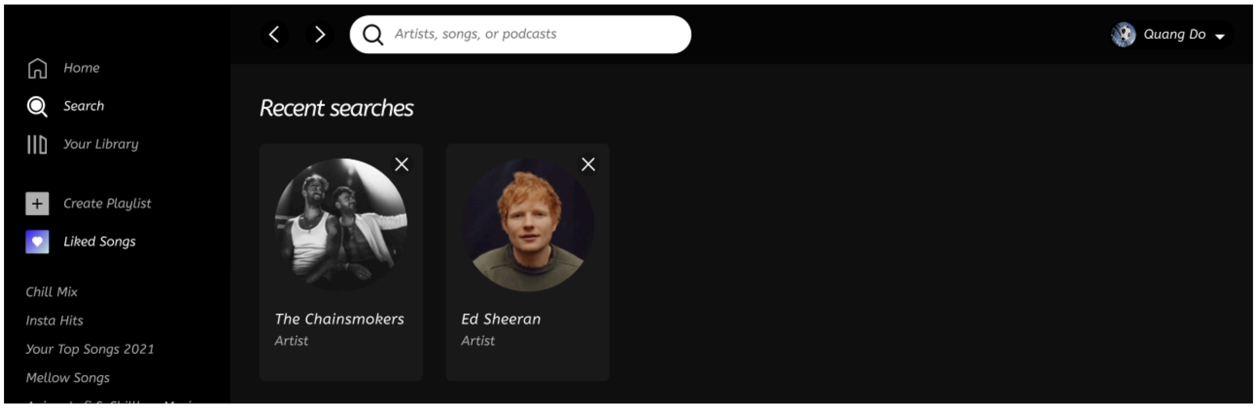


Figure 26: Demo search area

5.4. Play track

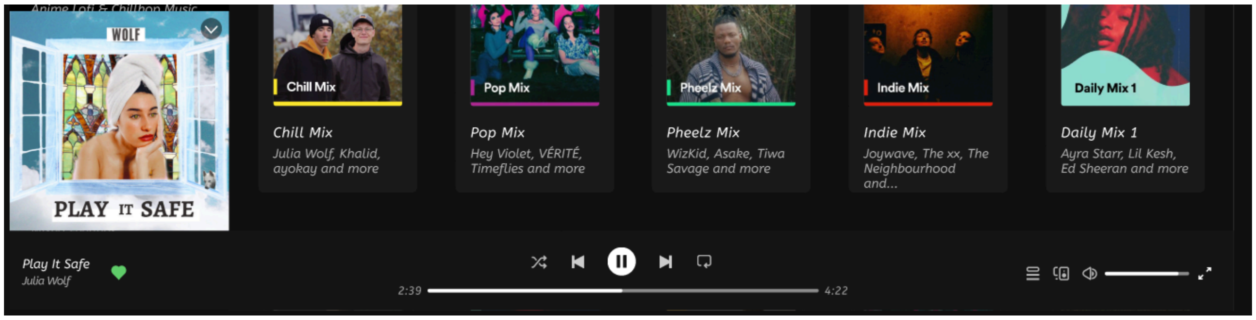

PLAY IT SAFE
Chill Mix
Julia Wolf, Khalid, ayokay and more
Pop Mix
Hey Violet, VÉRITÉ, Timeflies and more
Pheelz Mix
WizKid, Asake, Tiwa Savage and more
Indie Mix
Joywave, The xx, The Neighbourhood and...
Daily Mix 1
Ayra Starr, Lil Kesh, Ed Sheeran and more
Play It Safe
Julia Wolf
2:39
4:22

Figure 27: Demo track player

5.5. Create playlist

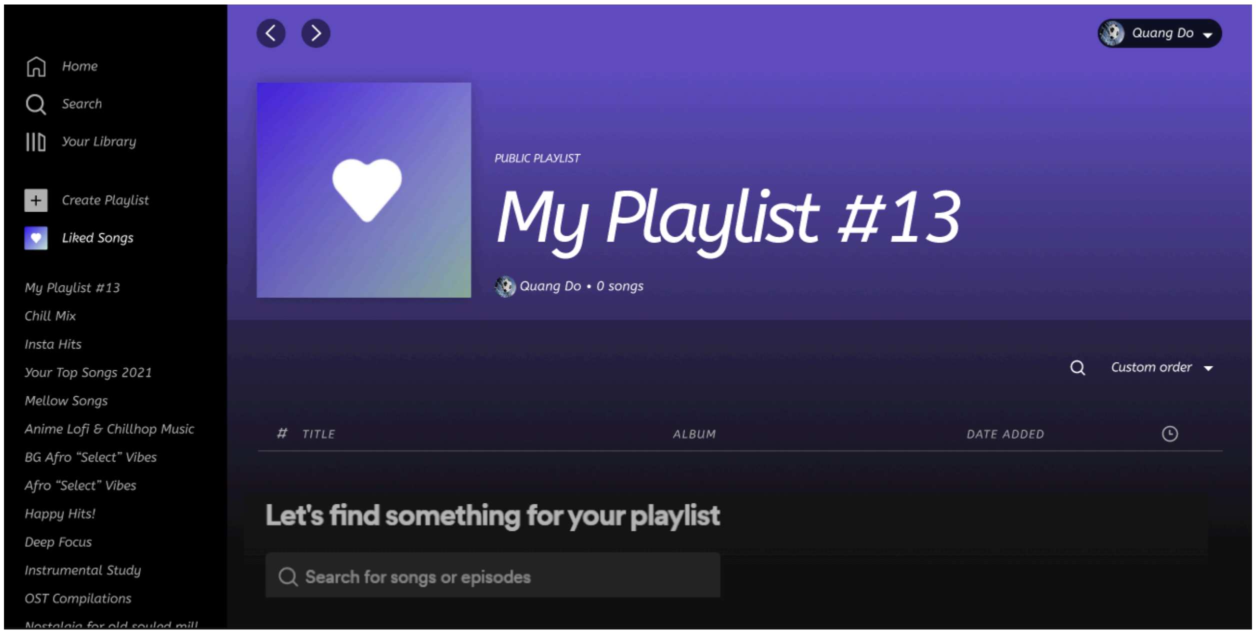


Figure 28: Demo create playlist

5.6. Manage playlist

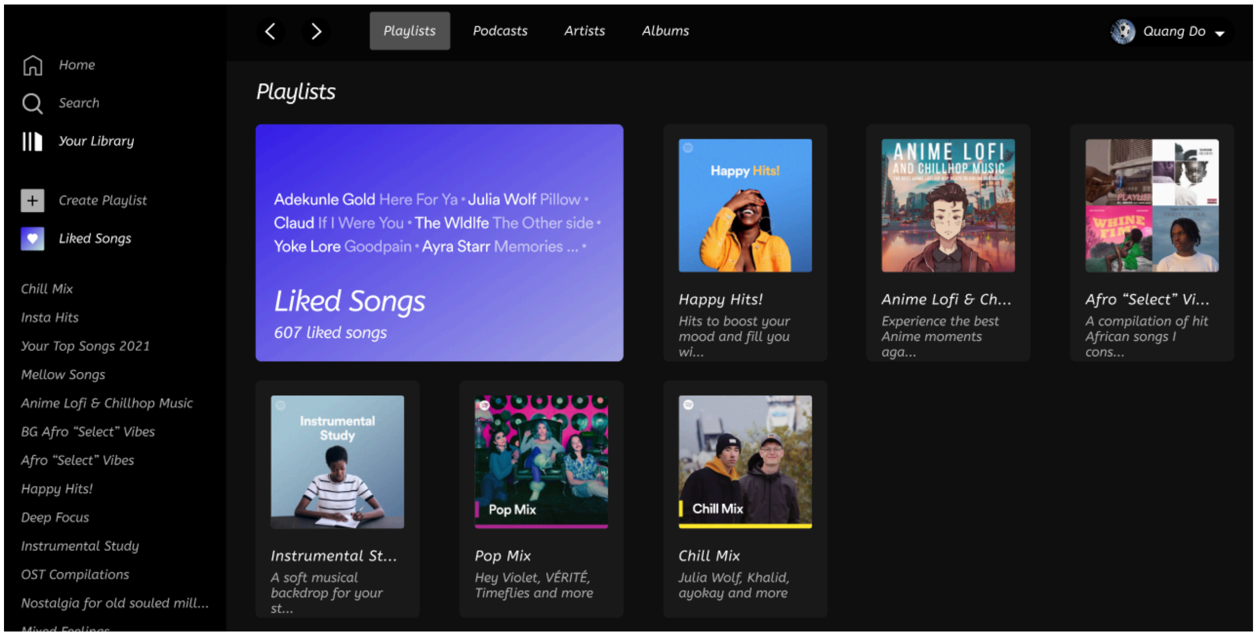

Playlists
Podcasts
Artists
Albums
Quang Do
Home
Search
Your Library
Create Playlist
Liked Songs
Chill Mix
Insta Hits
Your Top Songs 2021
Mellow Songs
Anime Lofi & Chillhop Music
BG Afro "Select" Vibes
Afro "Select" Vibes
Happy Hits!
Deep Focus
Instrumental Study
OST Compilations
Nostalgia for old souled mill...
Playlists
Adekunle Gold Here For Ya • Julia Wolf Pillow • Claud If I Were You • The Wldlfe The Other side • Yoke Lore Goodpain • Ayra Starr Memories ... •
Liked Songs
607 liked songs
Happy Hits!
Hits to boost your mood and fill you wi...
Anime Lofi & Ch...
Experience the best Anime moments aga...
Afro "Select" Vi...
A compilation of hit African songs I cons...
Instrumental St...
A soft musical backdrop for your st...
Pop Mix
Hey Violet, VÉRITÉ, Timeflies and more
Chill Mix
Julia Wolf, Khalid, ayokay and more

Figure 29: Demo playlist management

5.7. Download track

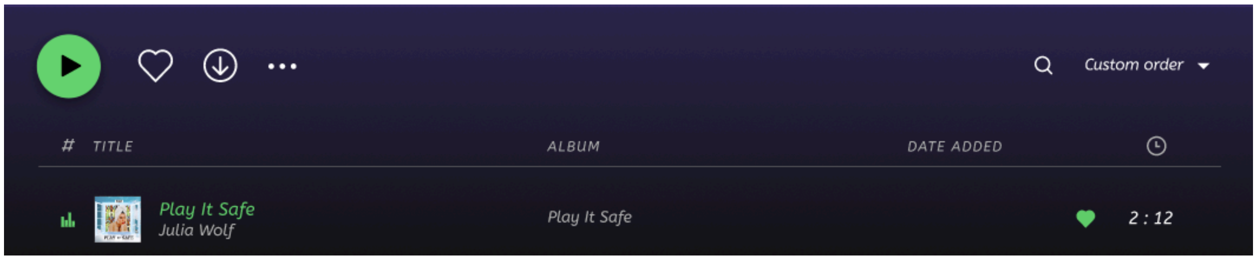
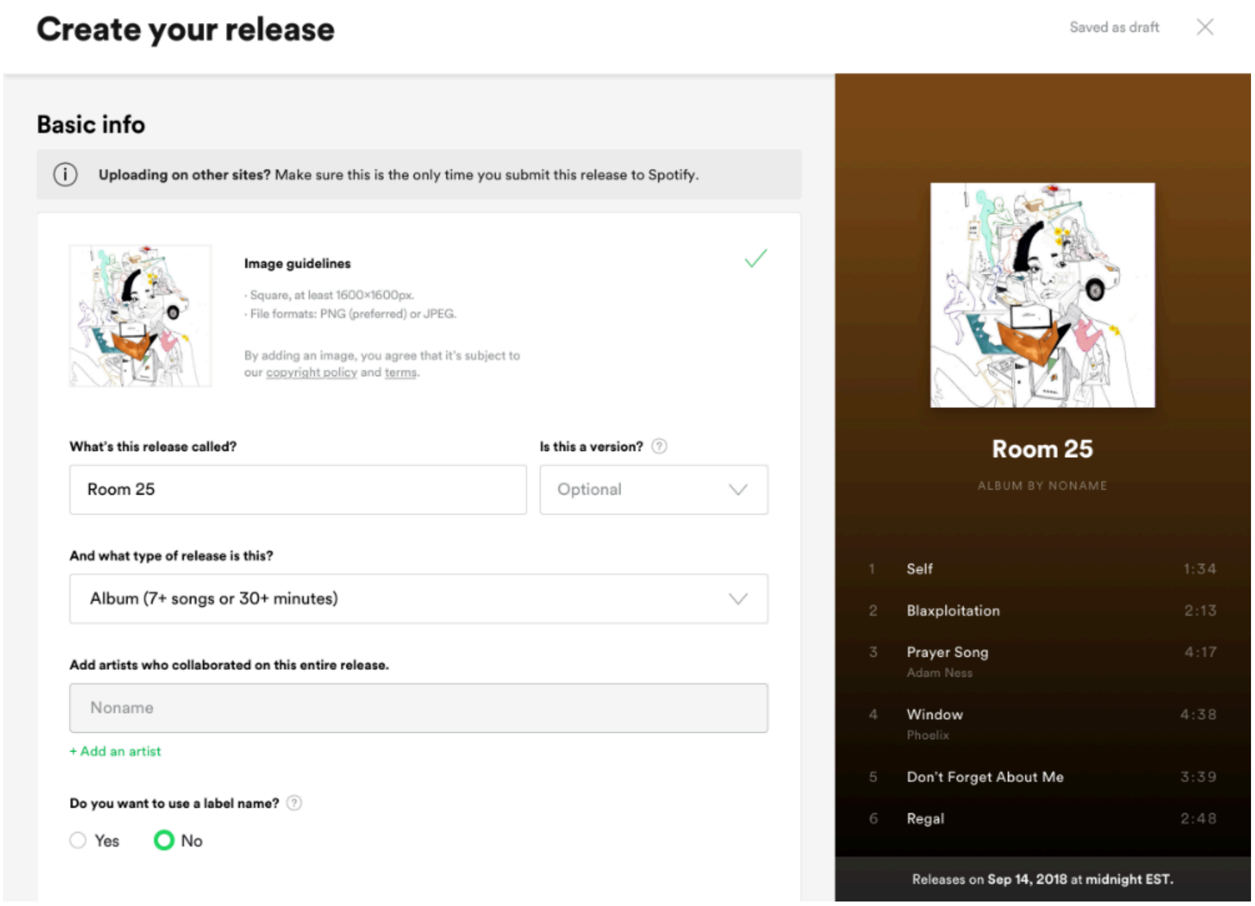


Figure 30: Demo download track

5.8. Upload and manage track



Create your release

Saved as draft

Basic info

Uploading on other sites? Make sure this is the only time you submit this release to Spotify.

Image guidelines

· Square, at least 1600×1600px.
· File formats: PNG (preferred) or JPEG.

By adding an image, you agree that it's subject to our copyright policy and terms.

What's this release called? Room 25

Is this a version? Optional

And what type of release is this? Album (7+ songs or 30+ minutes)

Add artists who collaborated on this entire release. Noname

+ Add an artist

Do you want to use a label name? Yes [✓ No]

Room 25

ALBUM BY NONAME

1	Self	1:34
2	Blaxploitation	2:13
3	Prayer Song Adam Ness	4:17
4	Window Phoelix	4:38
5	Don't Forget About Me	3:39
6	Regal	2:48

Releases on Sep 14, 2018 at midnight EST.

Figure 31: Demo artist upload and manage track

6. Database Diagrams

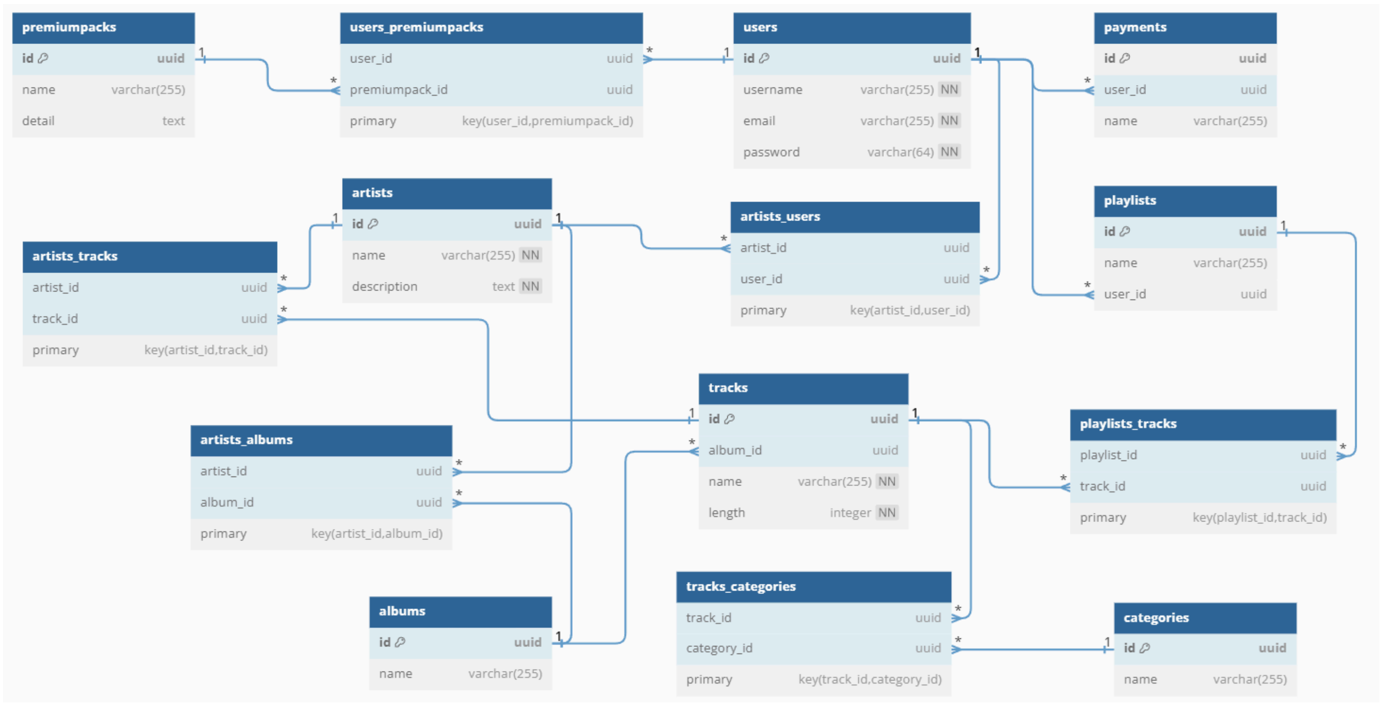


Figure 32: Database Diagram

7. Data Structures and Algorithms

To accelerate query speed for the search function and other retrieval features, we implement indexing on fields with potential for querying. For instance, indexing is applied to fields like the names of tables such as artists, tracks, albums, and categories. Indexing is employed to leverage the fast read capabilities of the B-tree structure in PostgreSQL. In addition to enhancing query speed, indexing also aids in maintaining data consistency and integrity. Furthermore, it allows for more complex queries, such as multi-column searches and range queries, to be executed efficiently.

GIN (Generalized Inverted Index) and full-text search vectors are also utilized to enhance query accuracy and speed. GIN indexes are particularly effective for data types that have multiple component elements such as arrays and full-text search vectors. They provide fast search capabilities over complex data types, making them an excellent choice for improving the performance of full-text searches.

LRU cache is applied for implementing the history search bar. The LRU (Least Recently Used) cache is an efficient way to store the most recently used search terms. It helps to quickly retrieve the user's search history, providing a seamless and responsive user experience. This approach significantly reduces the need for database queries, thereby improving the overall performance of the search feature.

8. APIs and Dependencies

Our software relies on downloading songs from the website MP3 Juices. MP3 Juices is a free MP3 search engine with a built-in downloader. We leverage the features of Playwright to simulate a web browser and send download queries for songs to MP3 Juices to save them into a static file. (Playwright is a framework for Web Testing and Automation. It allows testing Chromium, Firefox, and WebKit with a single API).

The Spotify for Developers API is utilized to retrieve information about tracks, albums, and artists. This ensures accurate information for the software.