
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 Đỗ Minh Quang Nguyễn Đức Khánh Đỗ Minh Quang 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

Table of Contents

1. Introduction	1
1.1. Purpose	1
1.2. Scope	1
2. System Architecture	2
3. Sequence Diagrams	2
3.1. Create an account	2
3.2. Log in	3
3.3. Play the track	4
3.4. Search the tracks, artists and genres	5
3.5. Download the track	6
3.6. Create playlists	7
3.7. Manage playlists	8
3.8. Upload the track	9
3.9. Manage track	9
3.10. Create album	9
3.11. Manage album	9
4. Class Diagrams	9
4.1. Sign up	9
4.2. Login	10
4.3. Feature 3	10
4.4. Feature 4	10
4.5.	10
5. User Interface Prototype	10
6. Database Diagrams	10
6.1. EER Diagram	10
7. Data Structures and Algorithms	11
8. APIs and Dependencies	11

1. Introduction

1.1. Purpose

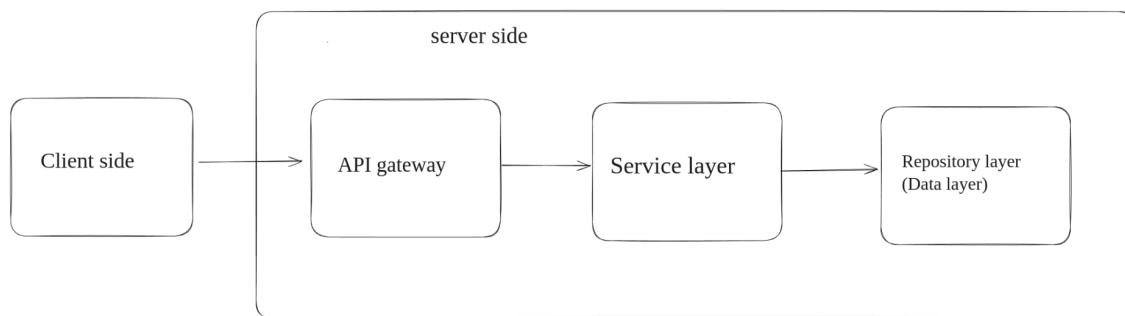
The purpose of this document is to present the initial design for a comprehensive Music Streaming System. This system is envisioned as a robust, user-friendly platform that will enable music enthusiasts to stream their favorite tracks, discover new music, and curate personalized playlists. The document 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)



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:

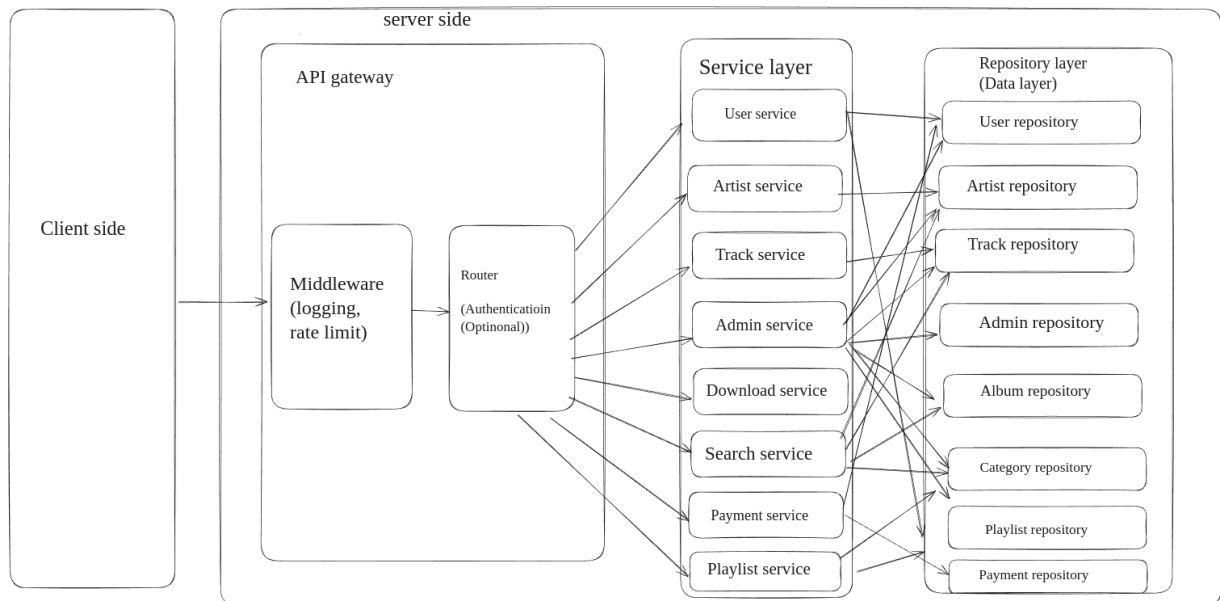
- Data validation, such as checking if the requested song ID is valid or if the size of the uploaded track file exceeds 50MB

- 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)



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

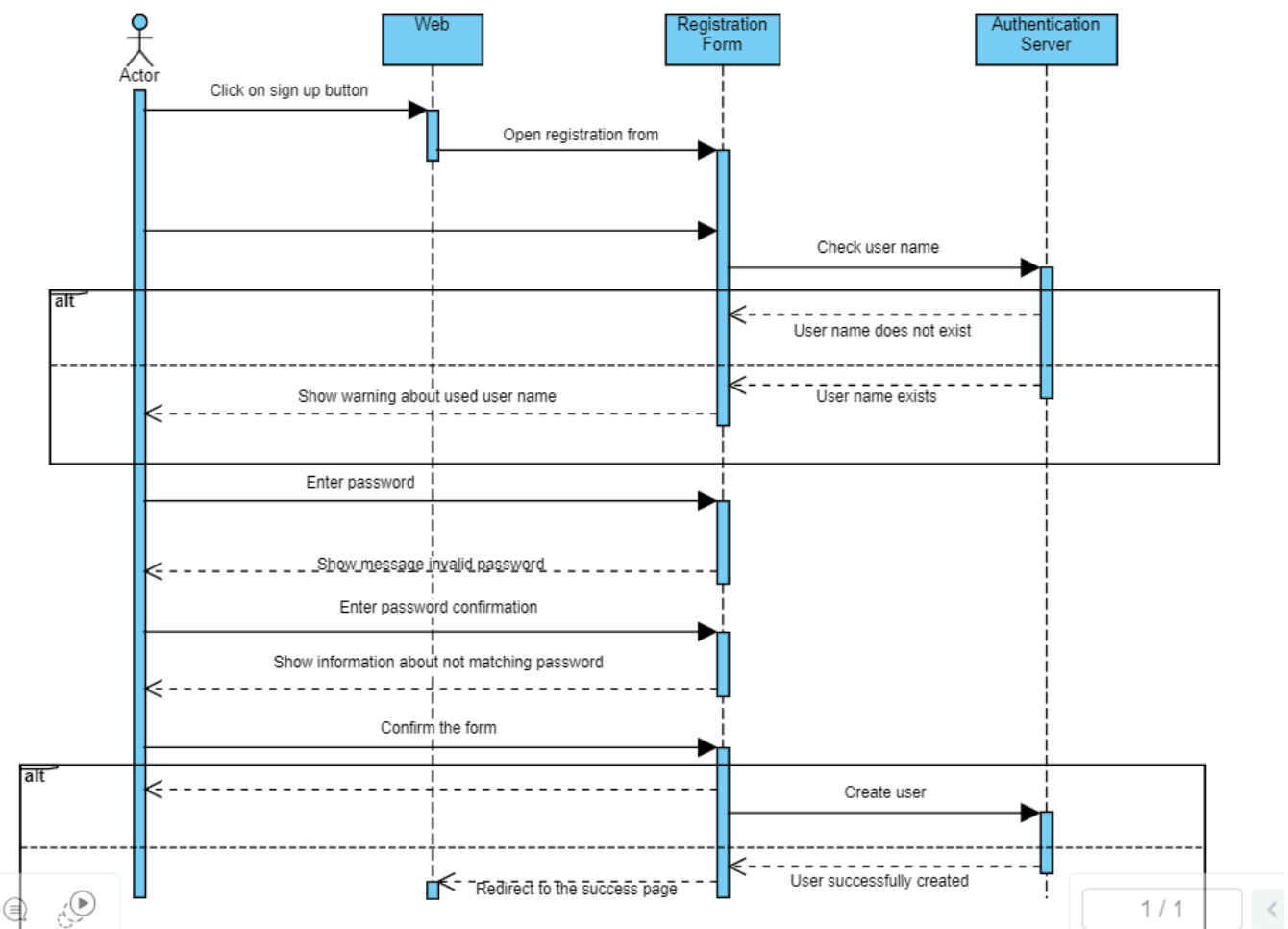


Figure : Account registration sequence diagram

3.2. Log in

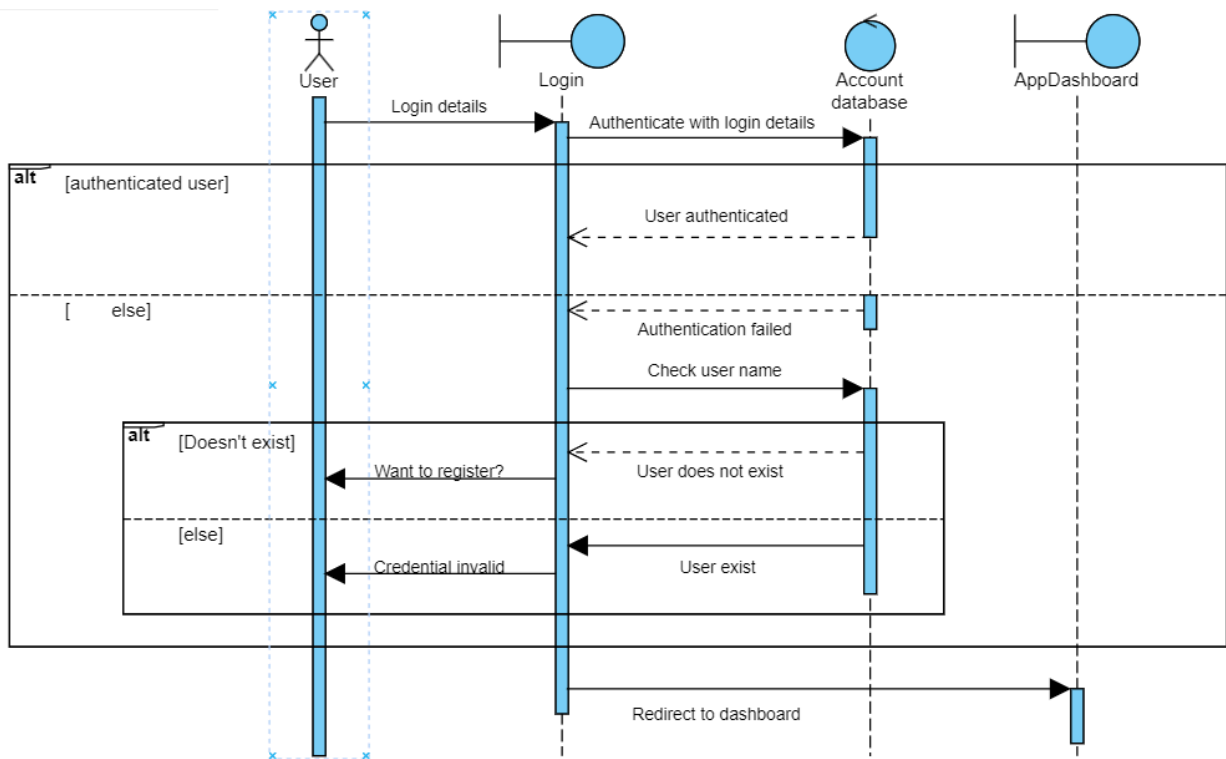


Figure : Login sequence diagram

3.3. Play the track

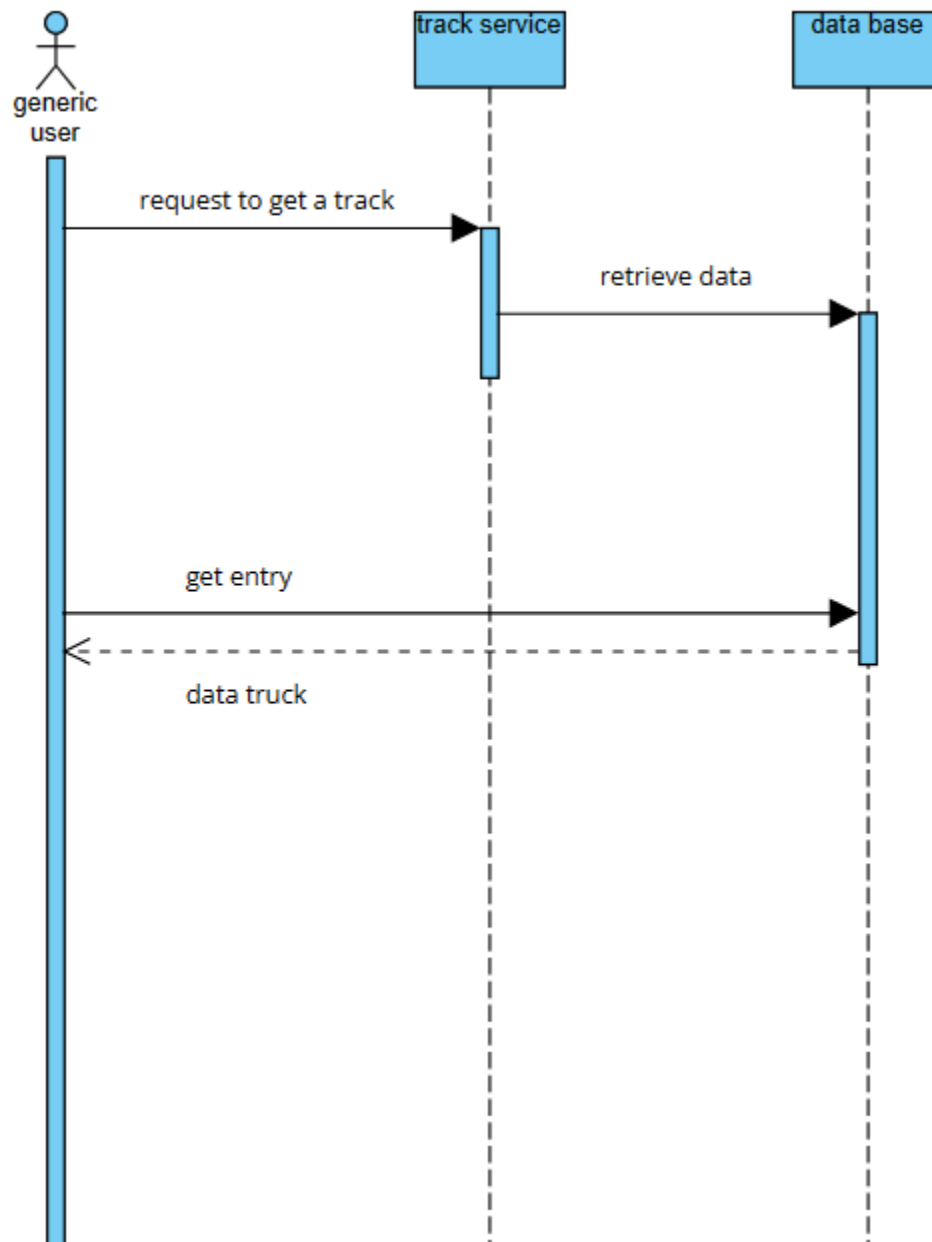


Figure : Playing Track sequence diagram

3.4. Search the tracks, artists and genres

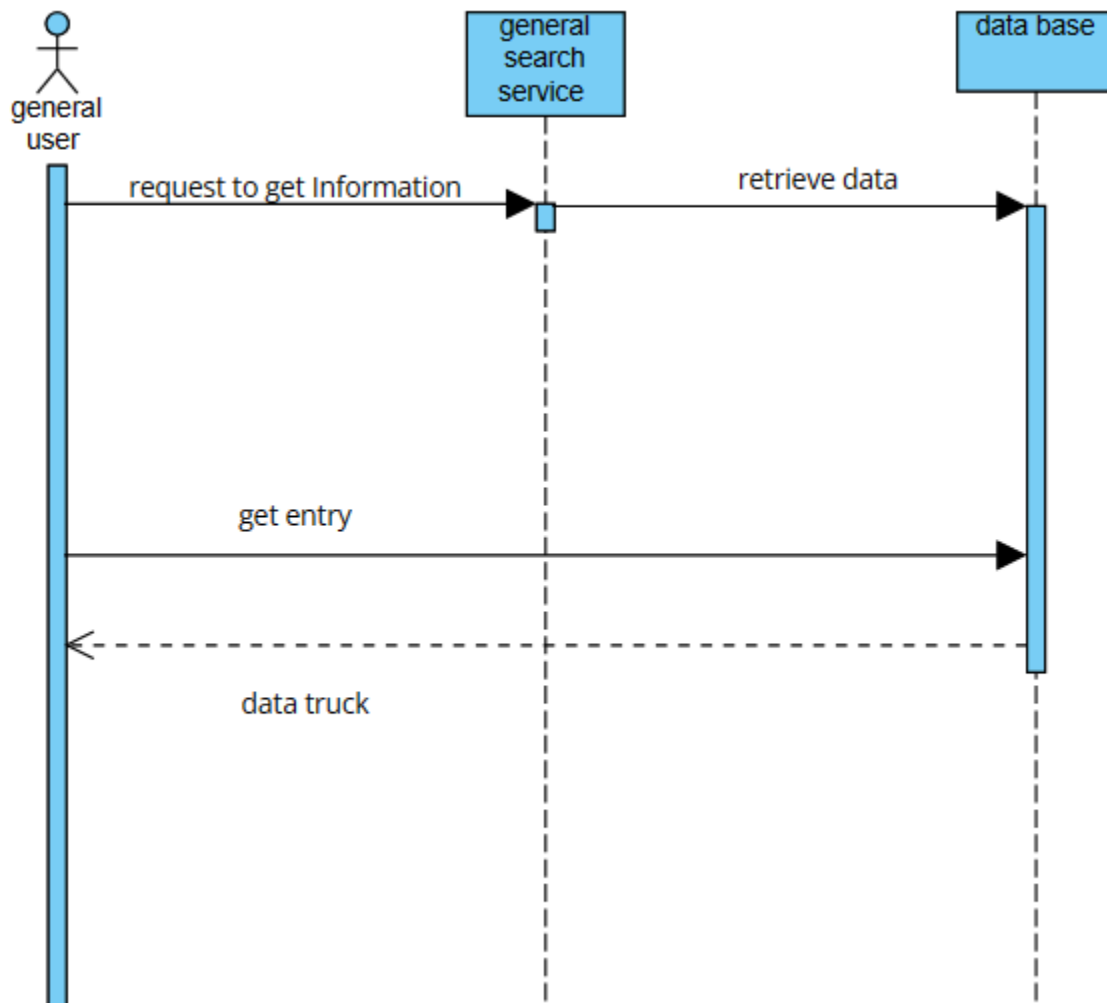


Figure : Search Engine sequence diagram

3.5. Download the track

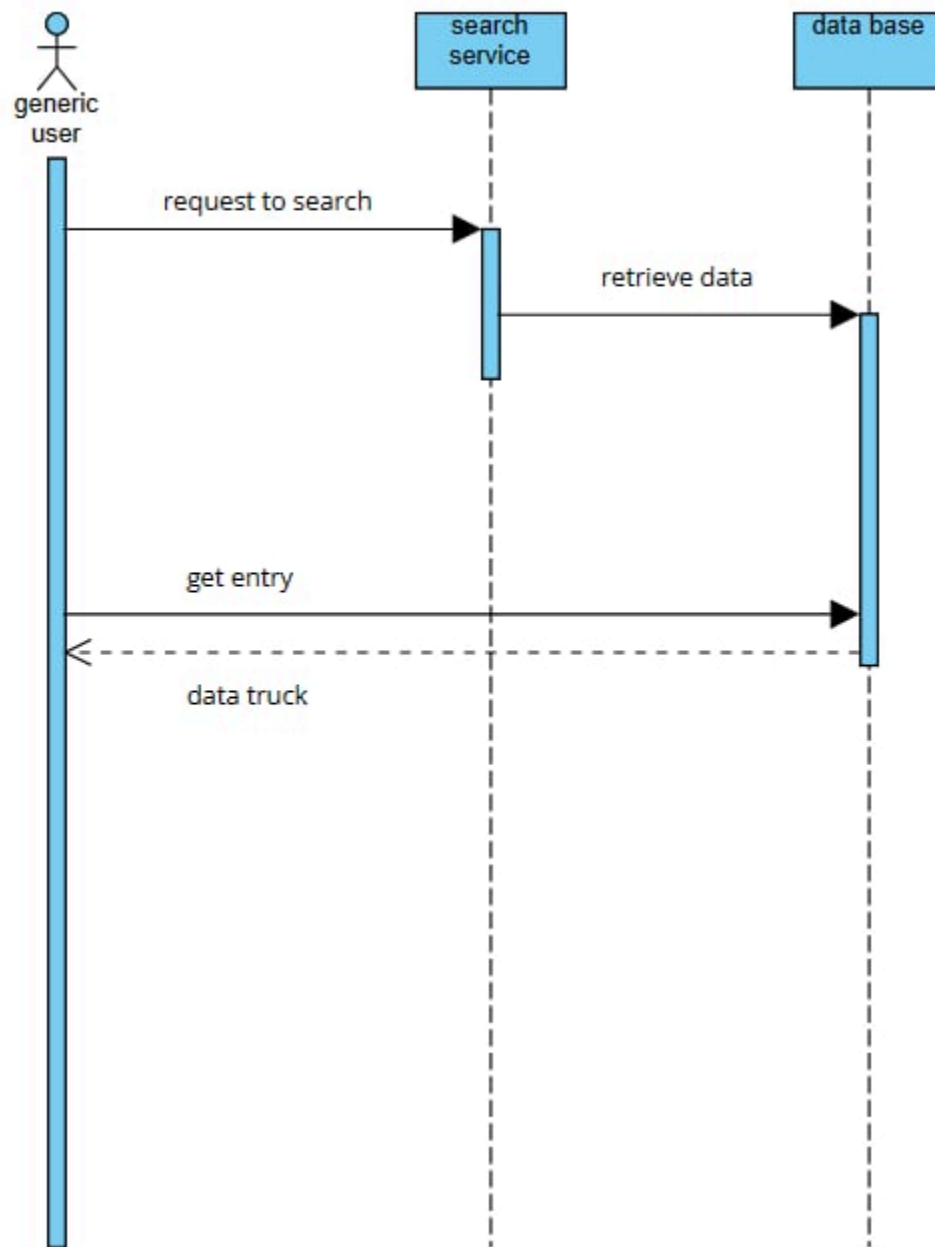


Figure : Track download sequence diagram

3.6. Create playlists

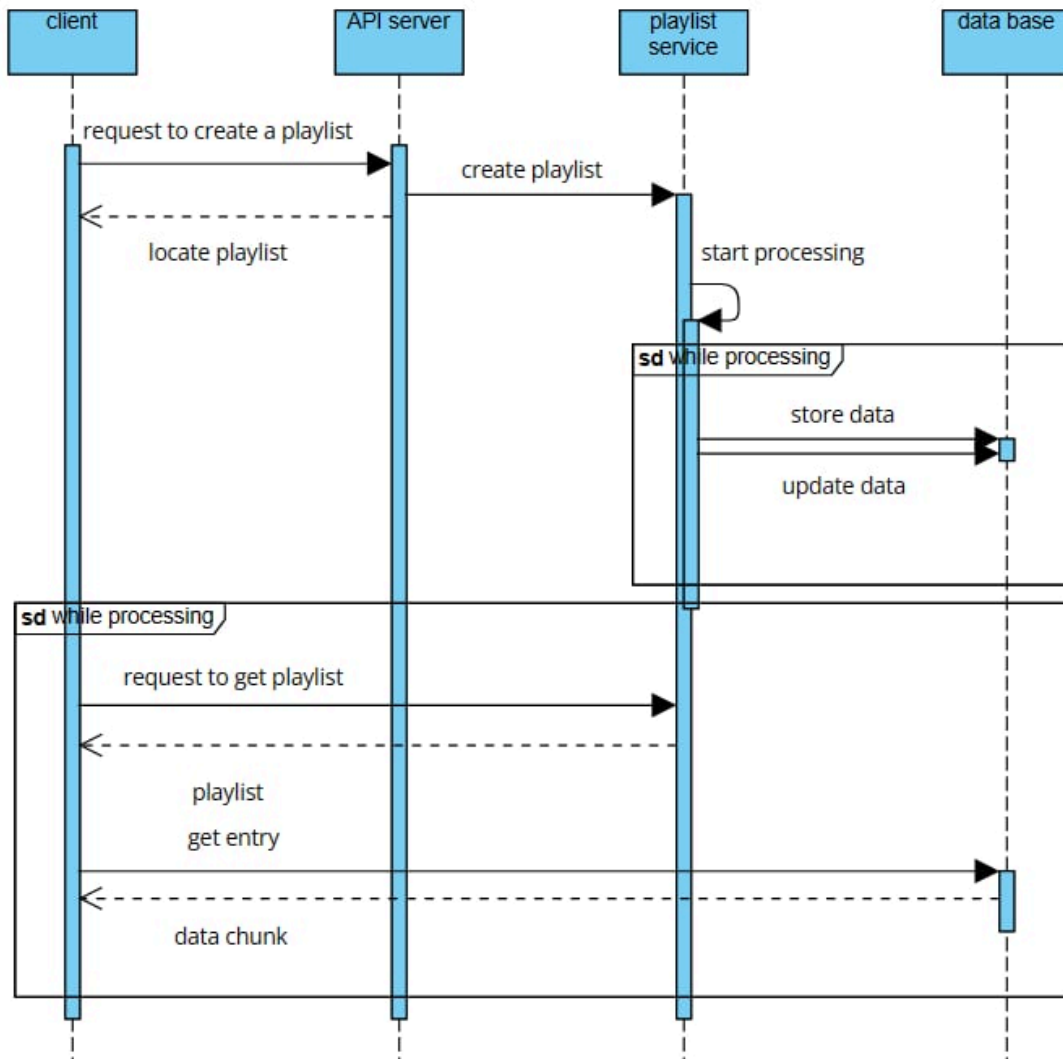


Figure : Playlist creation sequence diagram

3.7. Manage playlists

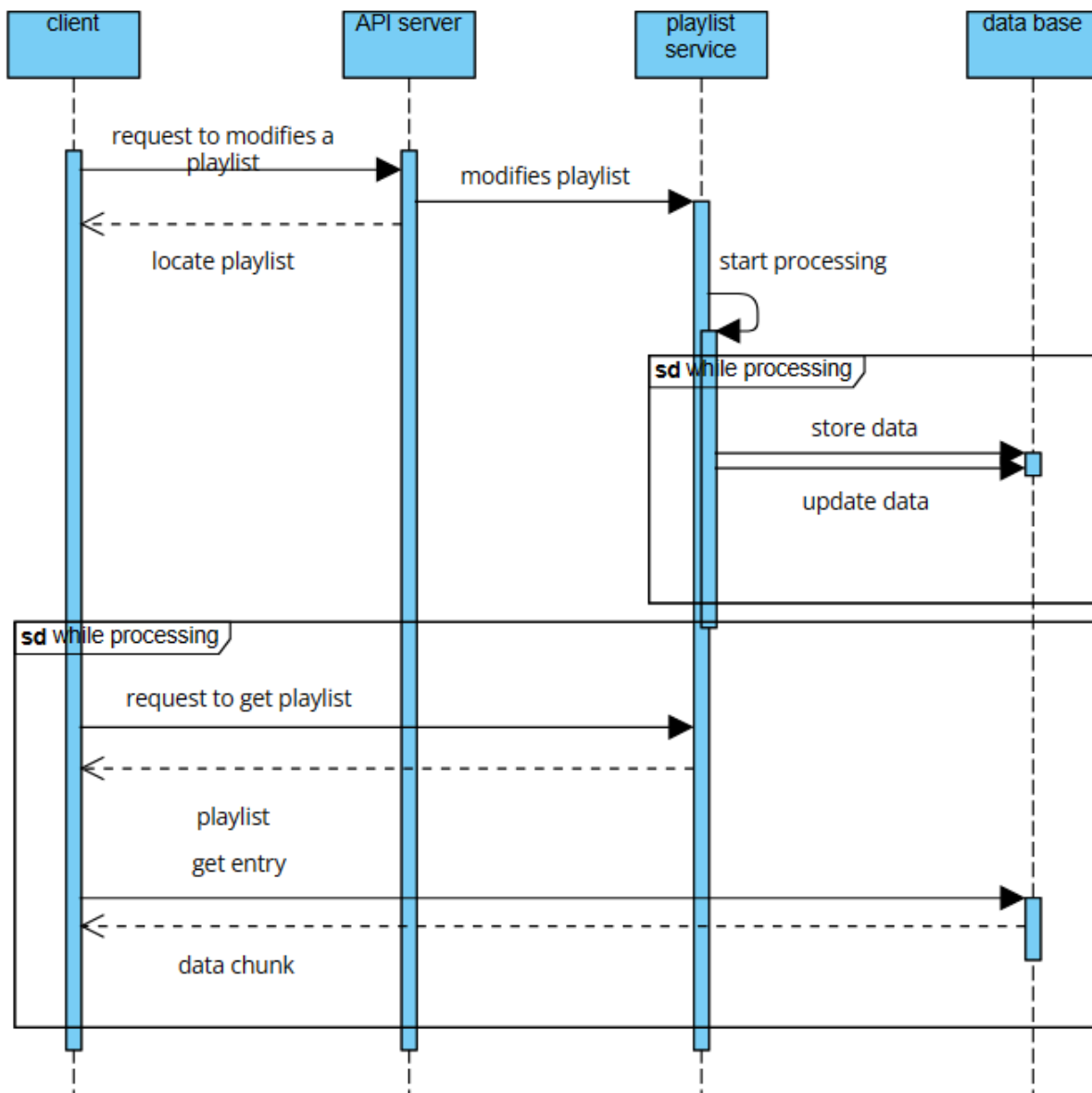


Figure : Playlist management sequence diagram

3.8. Upload the track

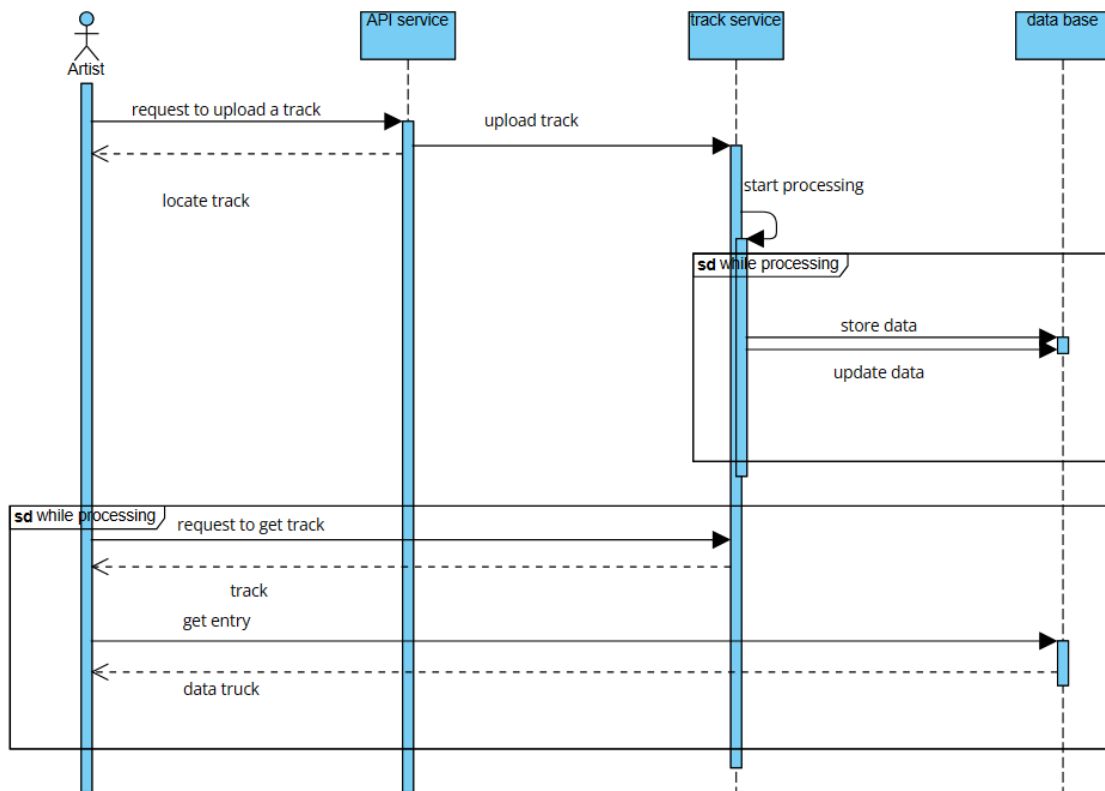


Figure : Track Upload sequence diagram

3.9. Manage track

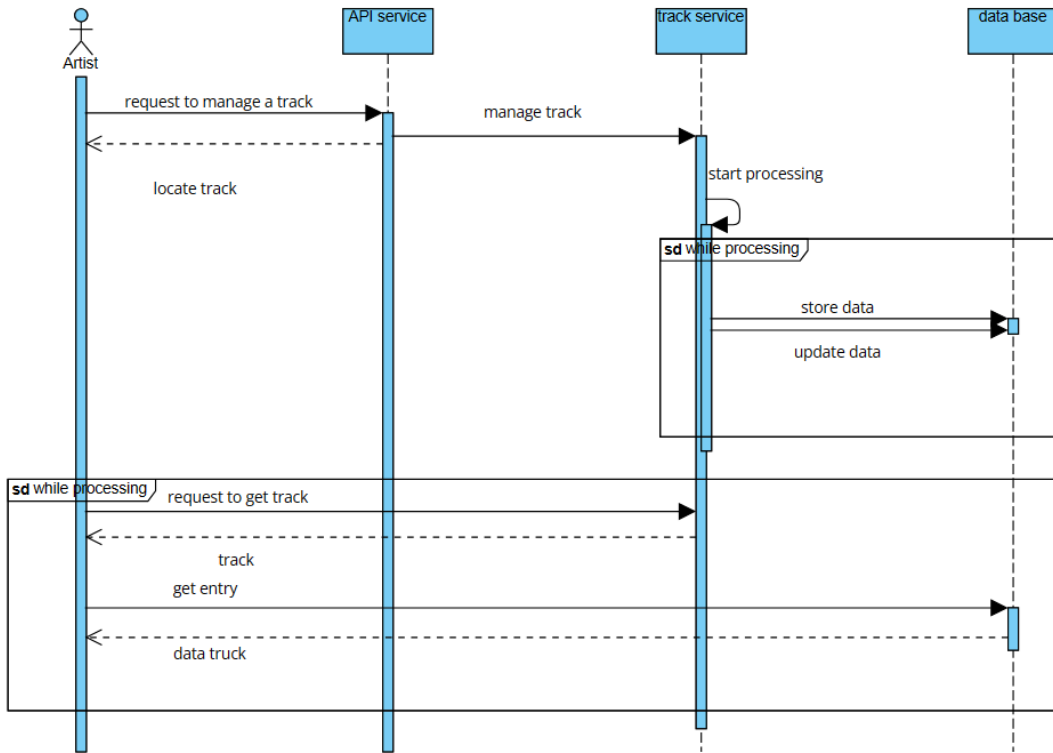


Figure : Track management sequence diagram

3.10. Create album

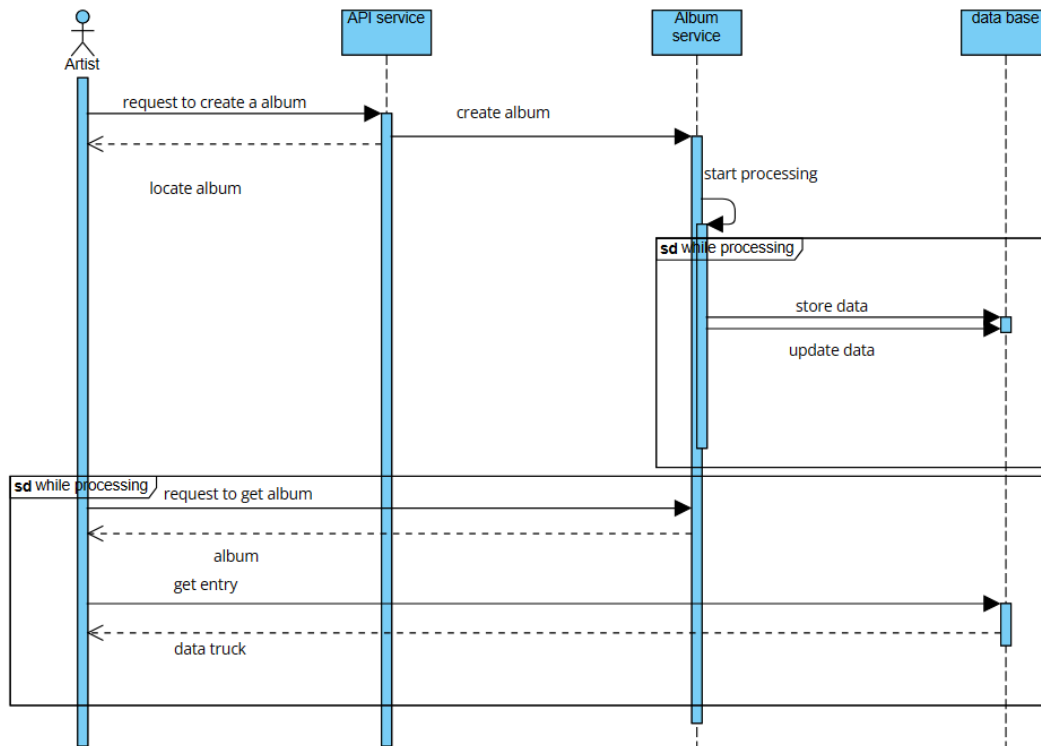


Figure : Album creation sequence diagram

3.11. Manage album

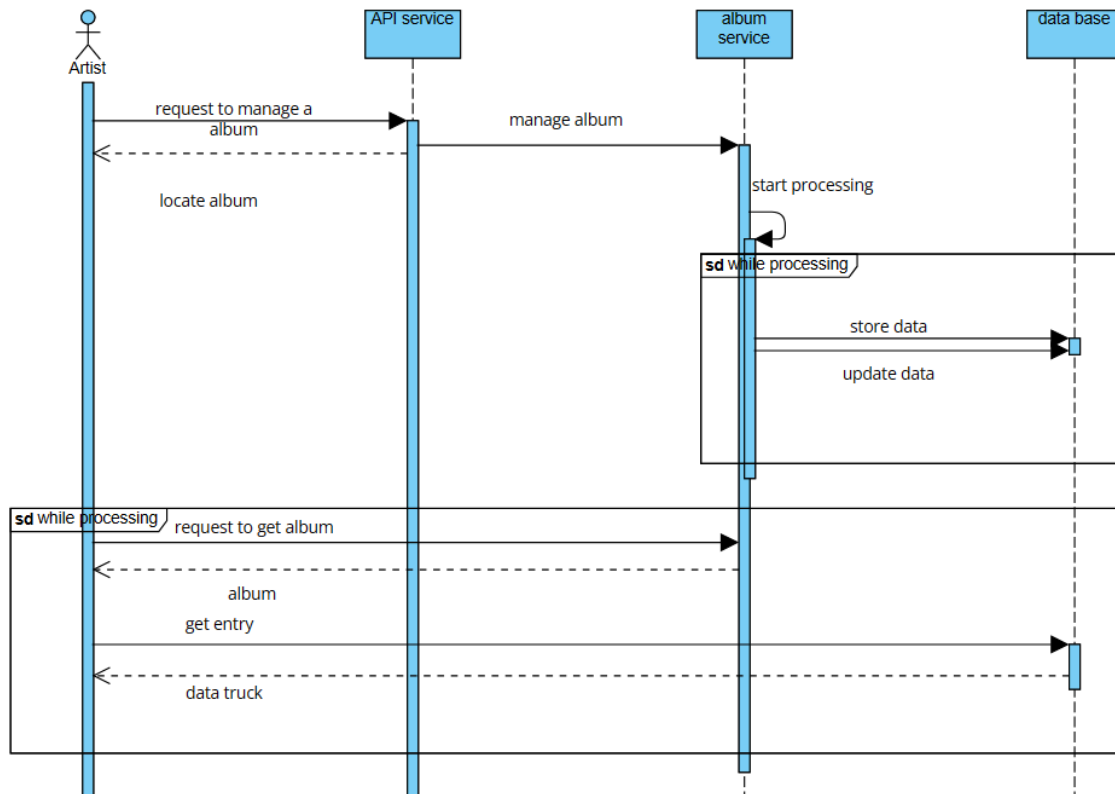
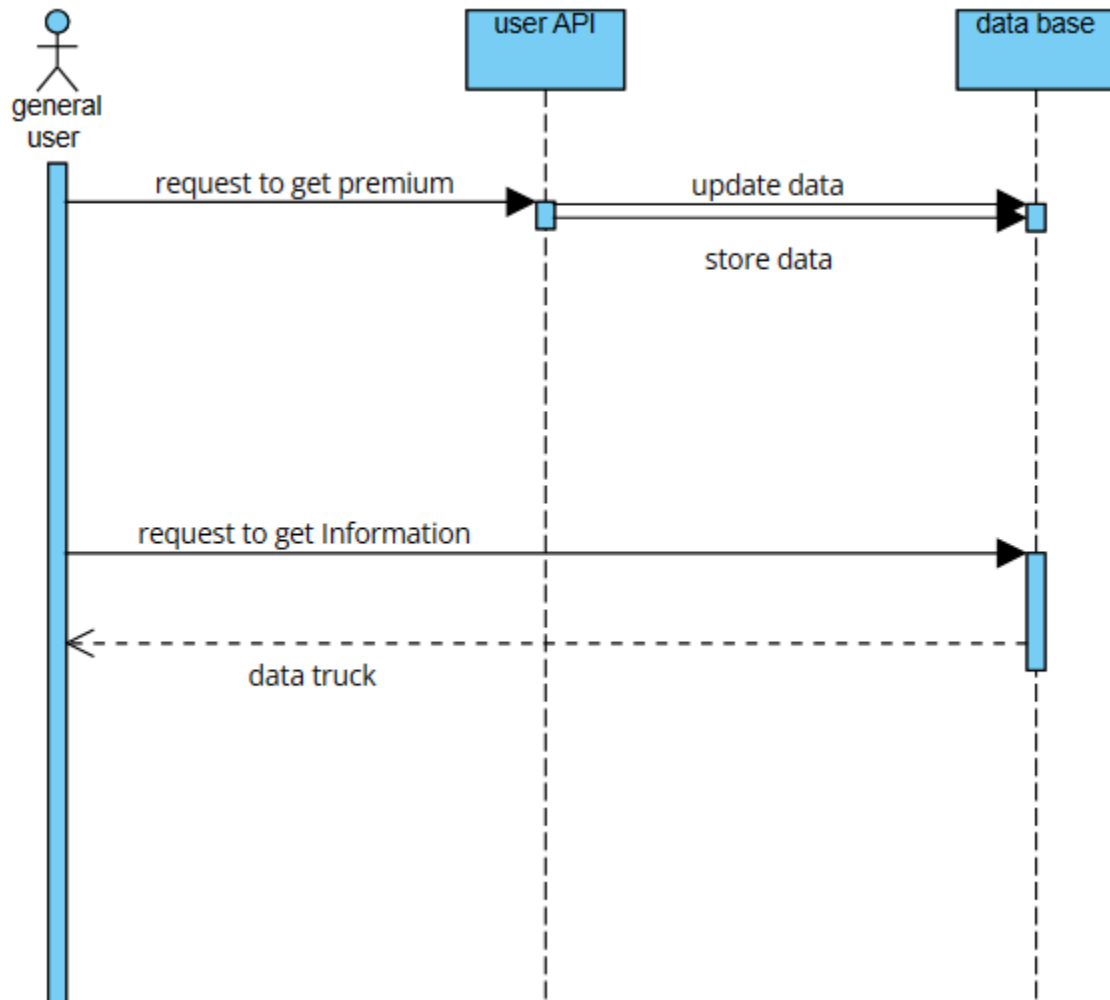


Figure : Album management sequence diagram

3.12. Upgrade account

*Figure : Account upgrade sequence diagram*

4. Class Diagrams

4.1. Sign up

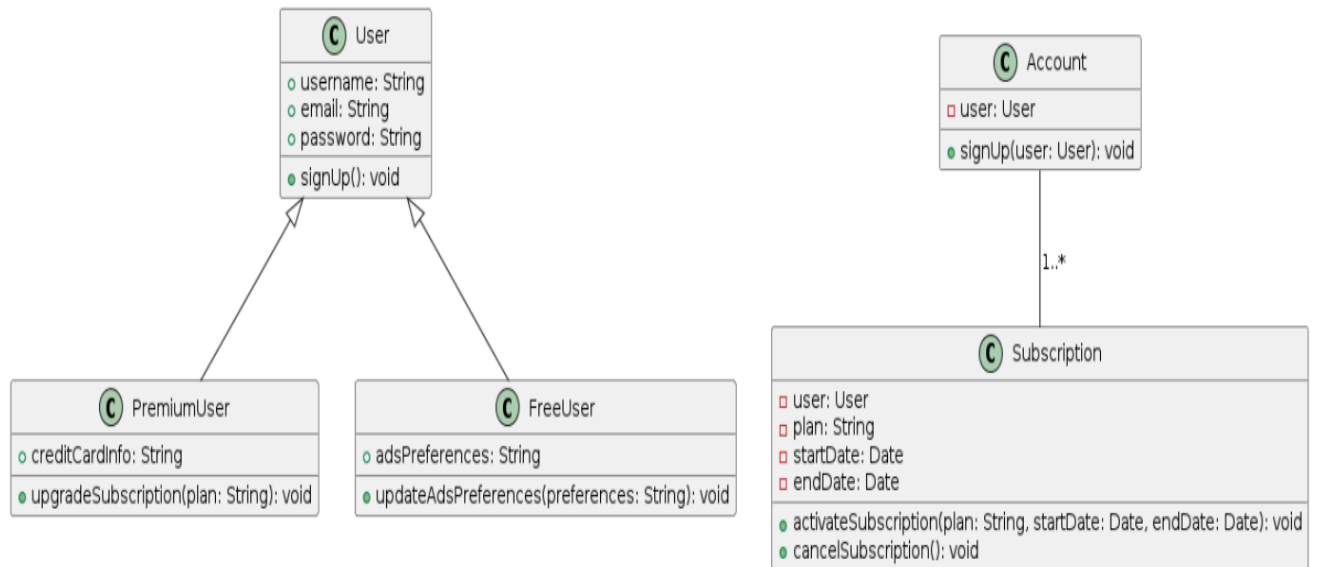


Figure : Account registration class diagram

4.2. Log in

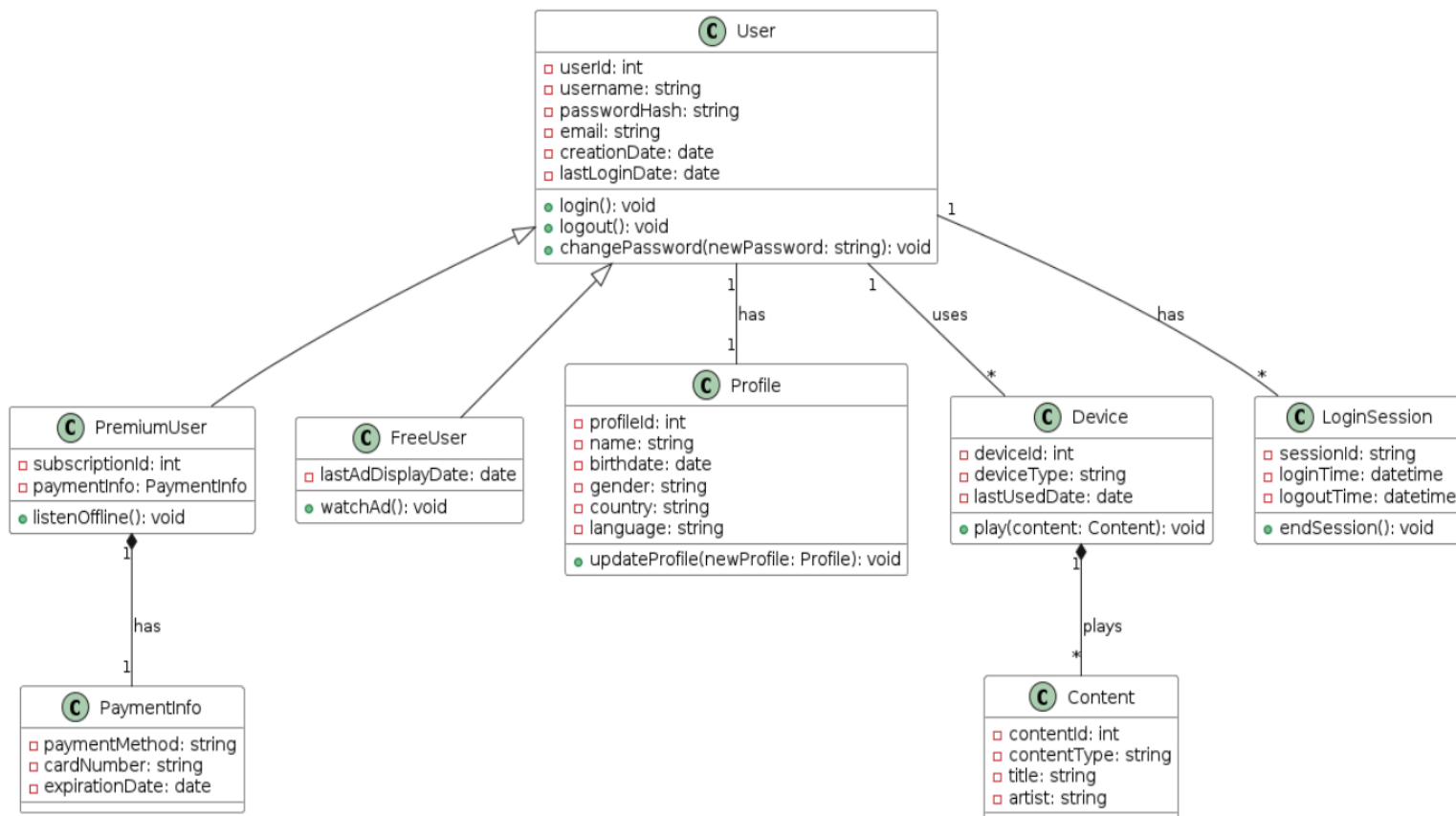


Figure : Login class diagram

4.3. Play the tracks

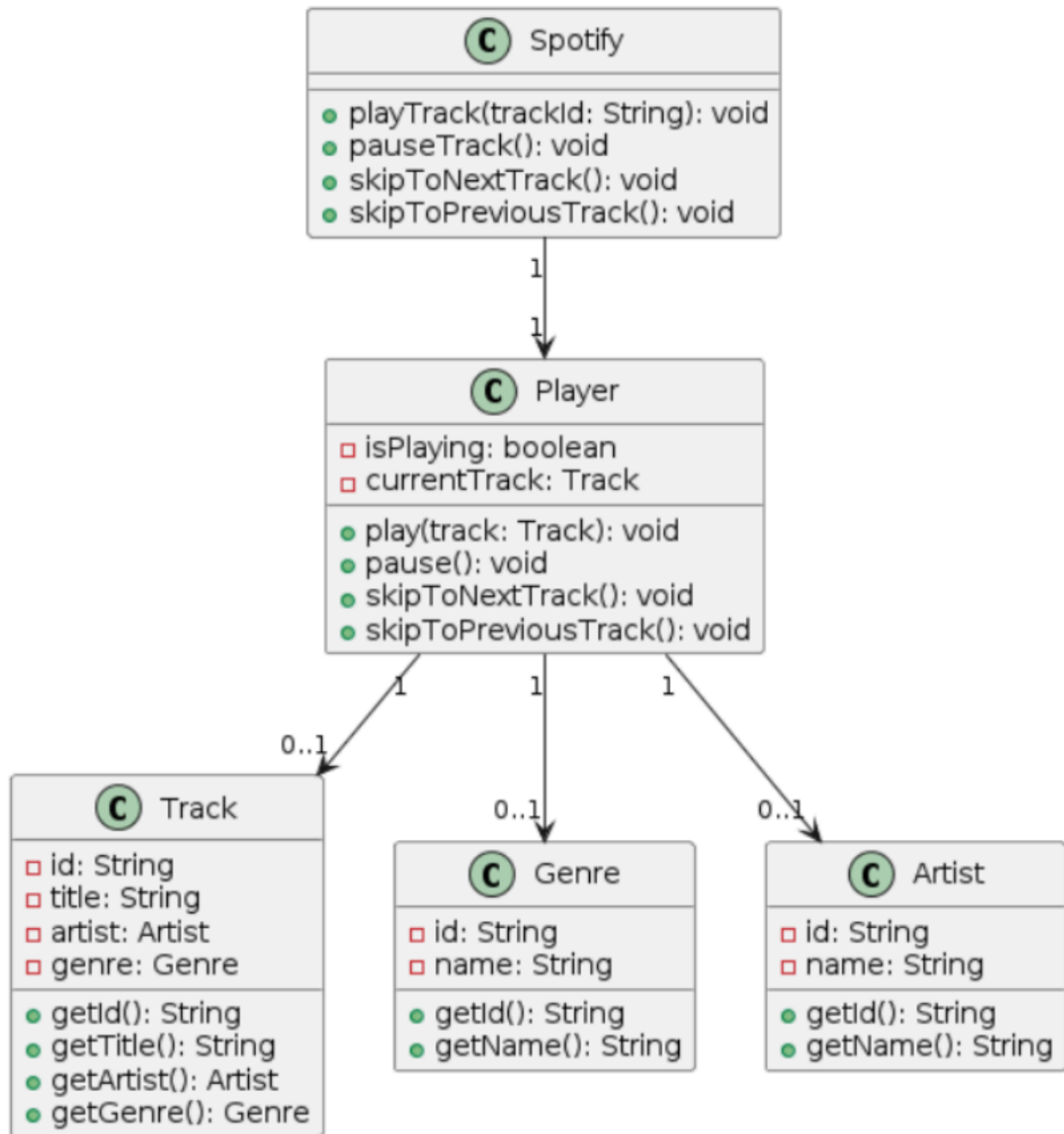


Figure : Playing Track class diagram

4.4. Search tracks, artists and genres

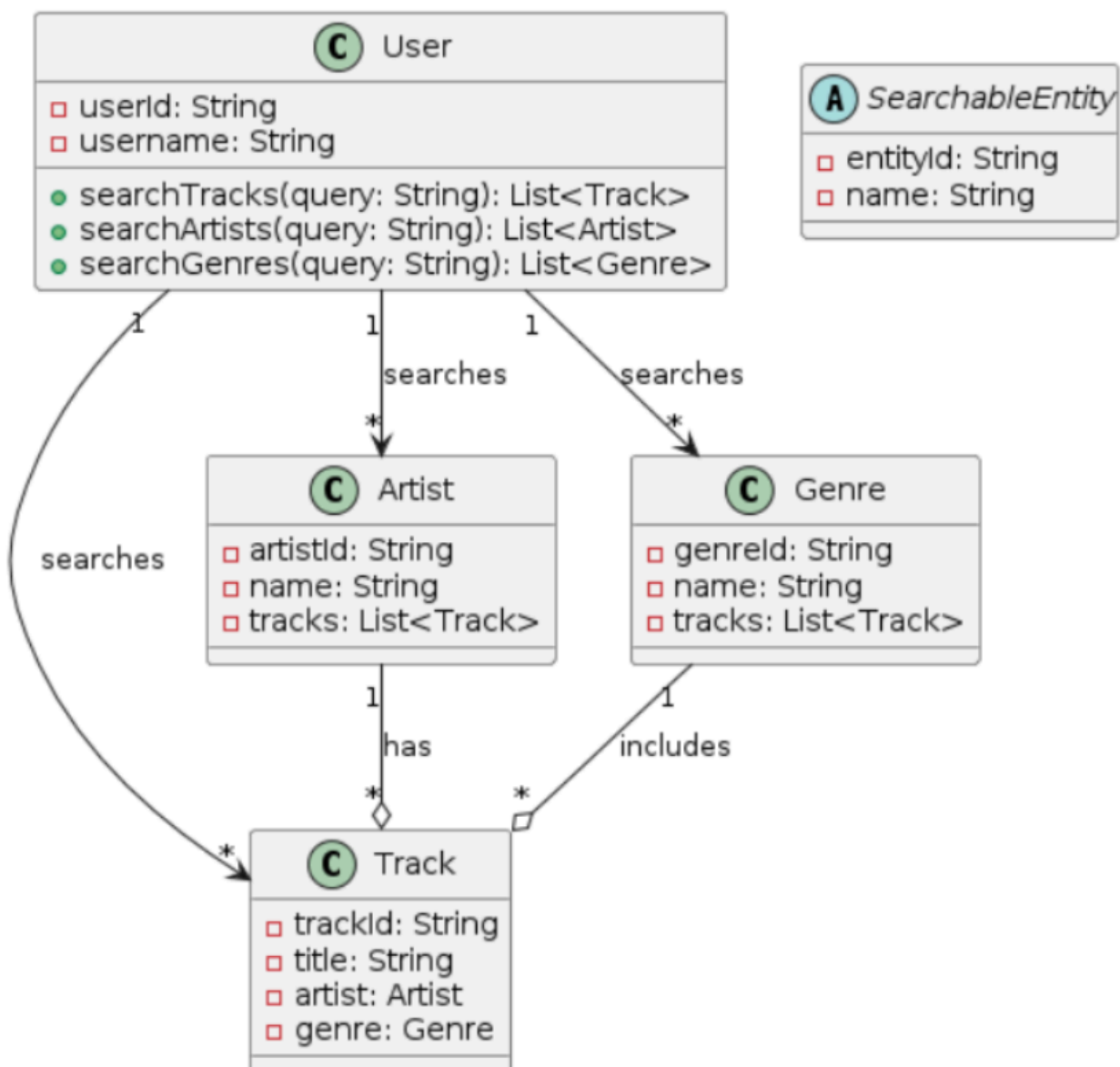


Figure : Search Engine class diagram

4.5. Download tracks

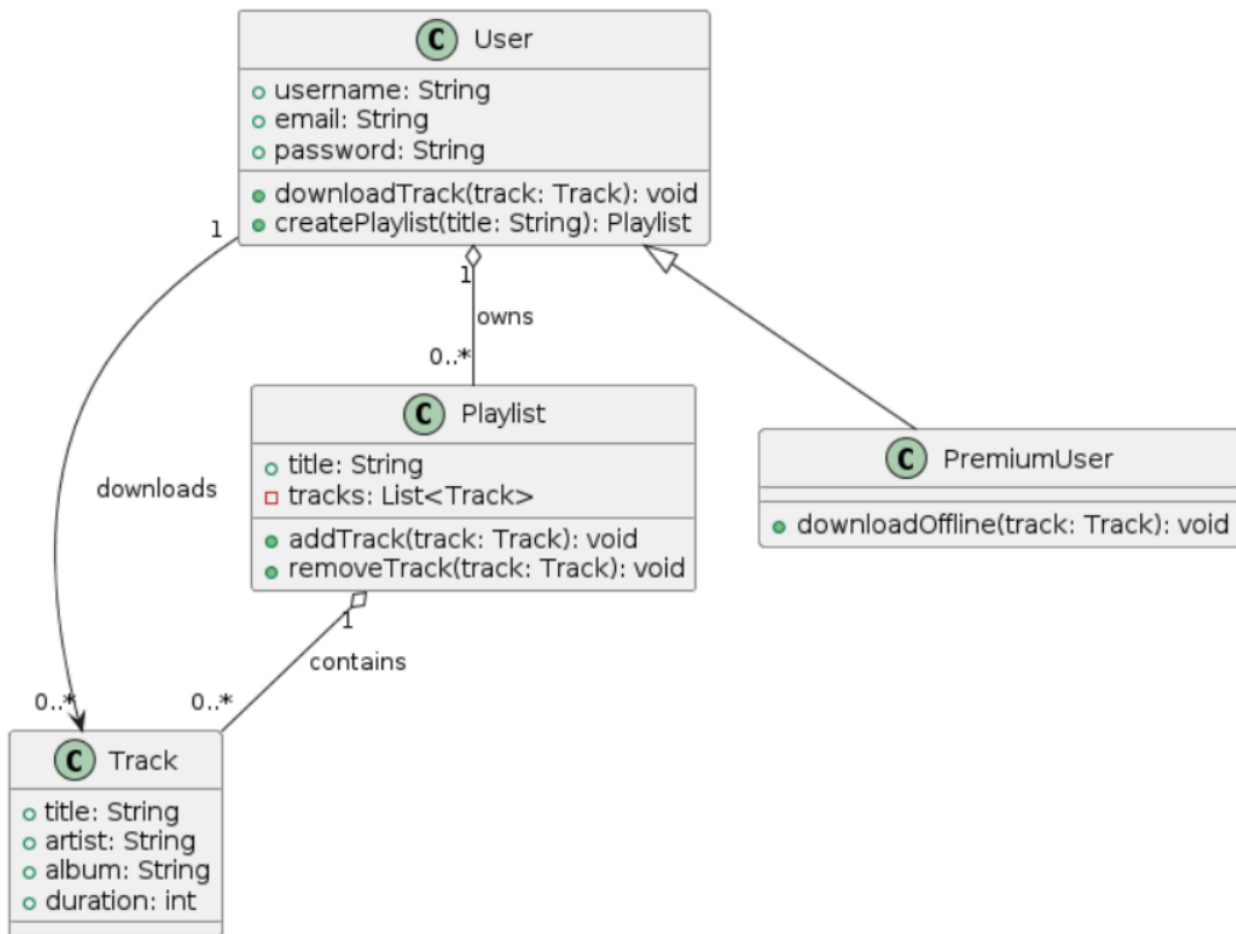


Figure : Track download class diagram

4.6. Create and manage playlists

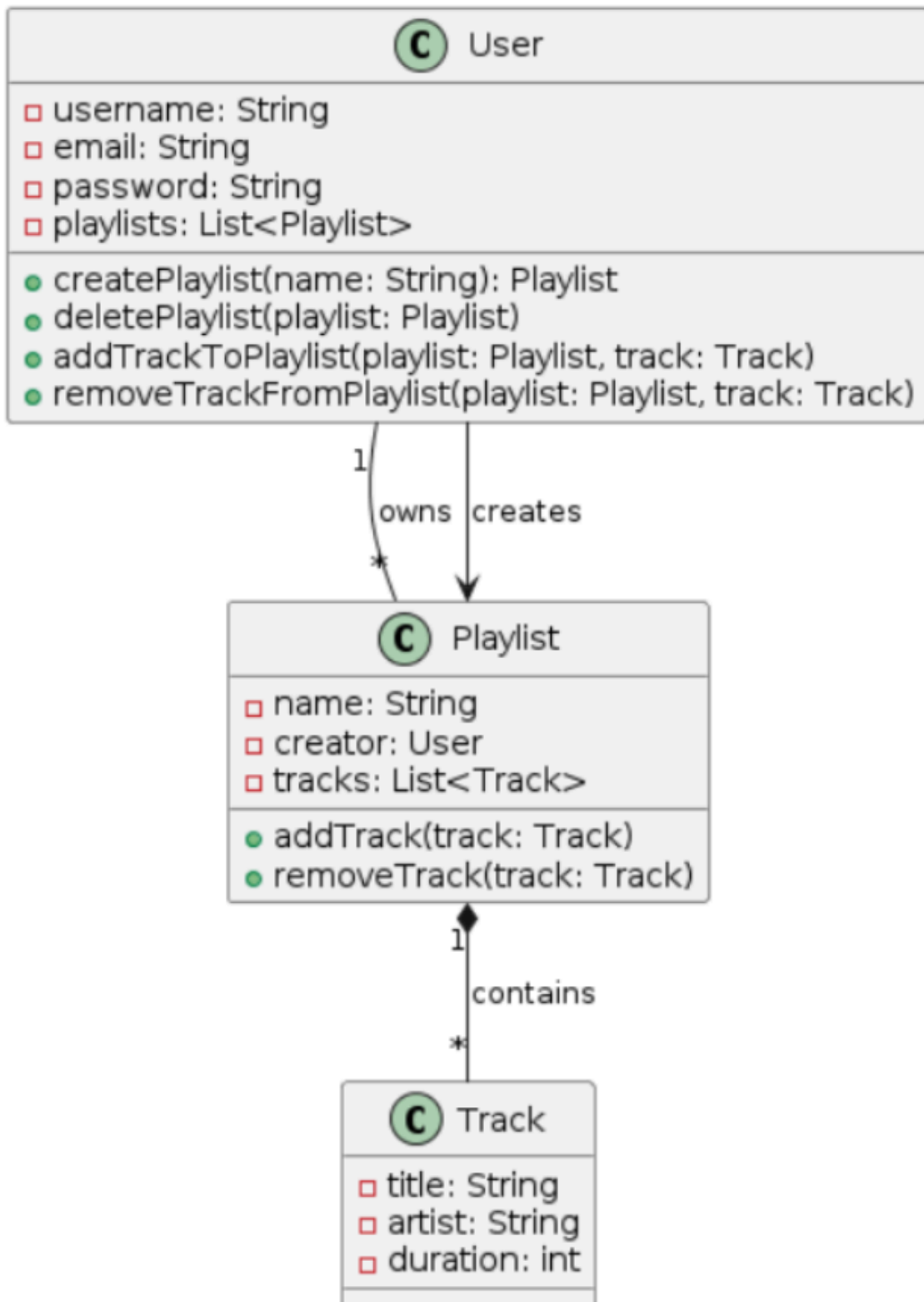


Figure : Playlist management class diagram

4.7. Upload and manage tracks

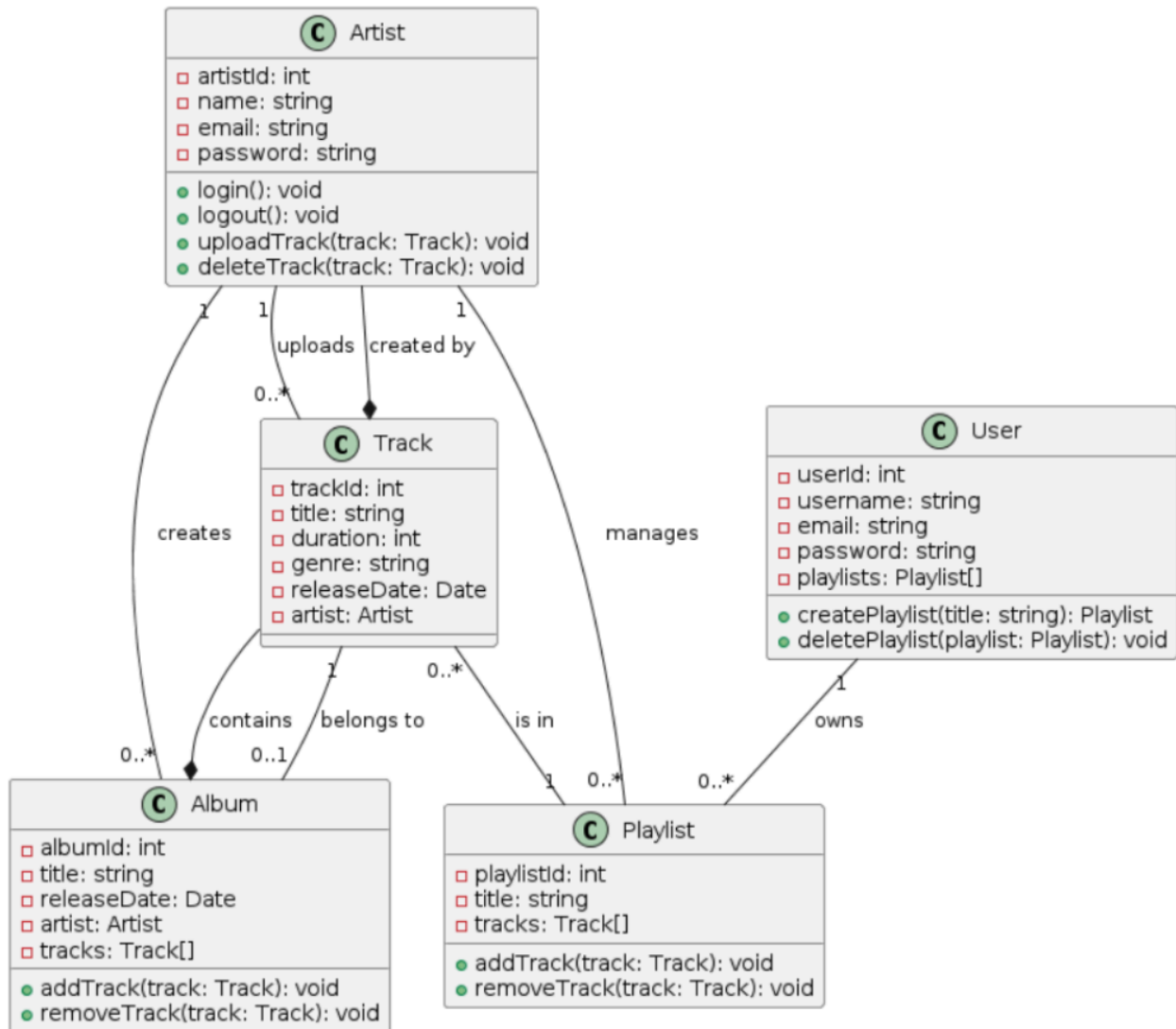


Figure : Track management class diagram

4.8. Upgrade account

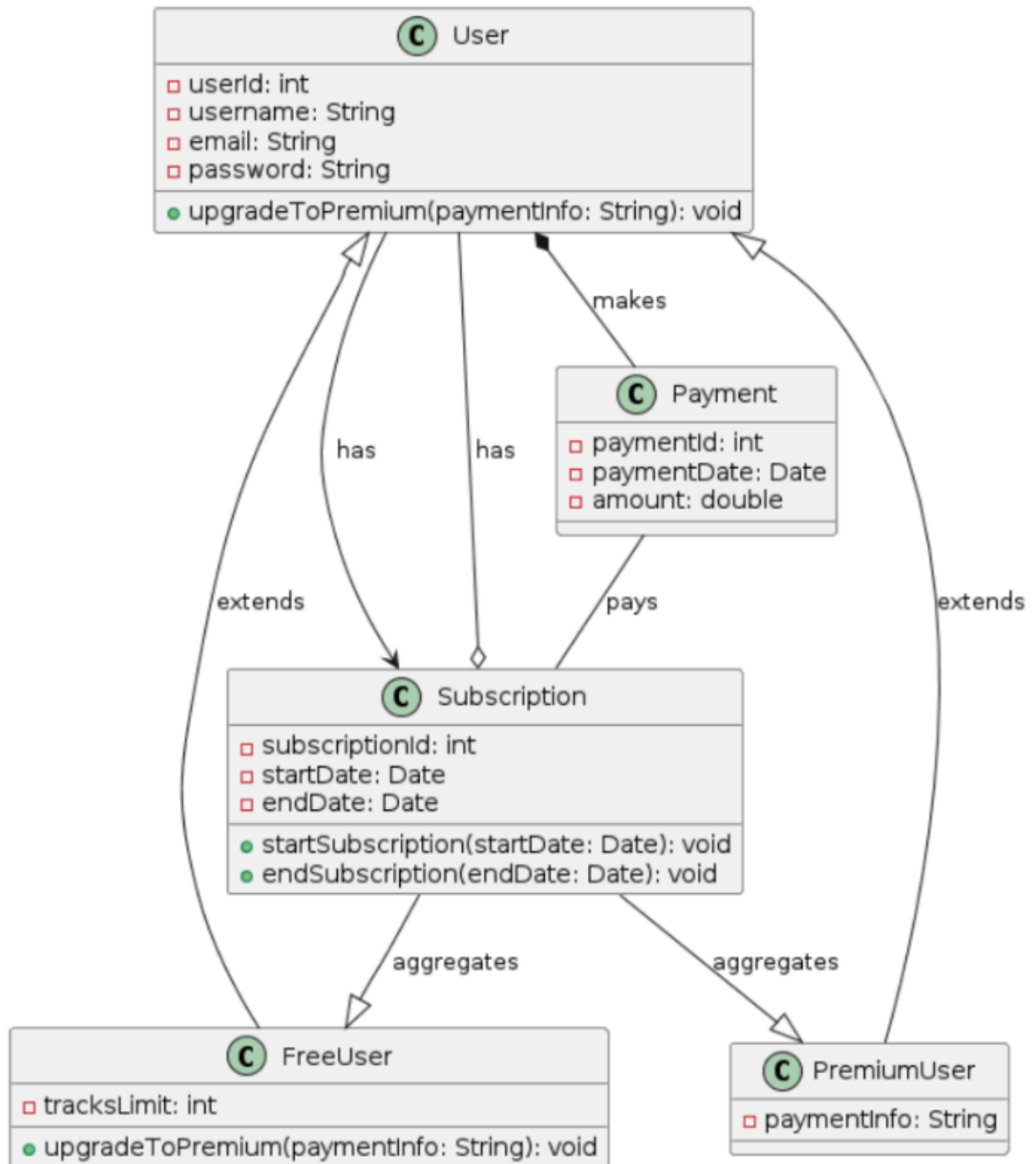


Figure : Account upgrade class diagram

5. User Interface Prototype

5.1. Create an account

Sign up to start listening


Email address


name@domain.com


[Use phone number instead.](#)

Next

or

 Sign up with Google

 Sign up with Facebook


 Sign up with Apple


Already have an account? [Log in here.](#)


Figure : Demo registration form

5.2. Login

Log in to Spotify

 Continue with Google


 Continue with Facebook

 Continue with Apple

Continue with phone number

Email or username

Password



☒ Remember me

Log In

[Forgot your password?](#)

Don't have an account? [Sign up for Spotify](#)

Figure : Demo login view

5.3. Search

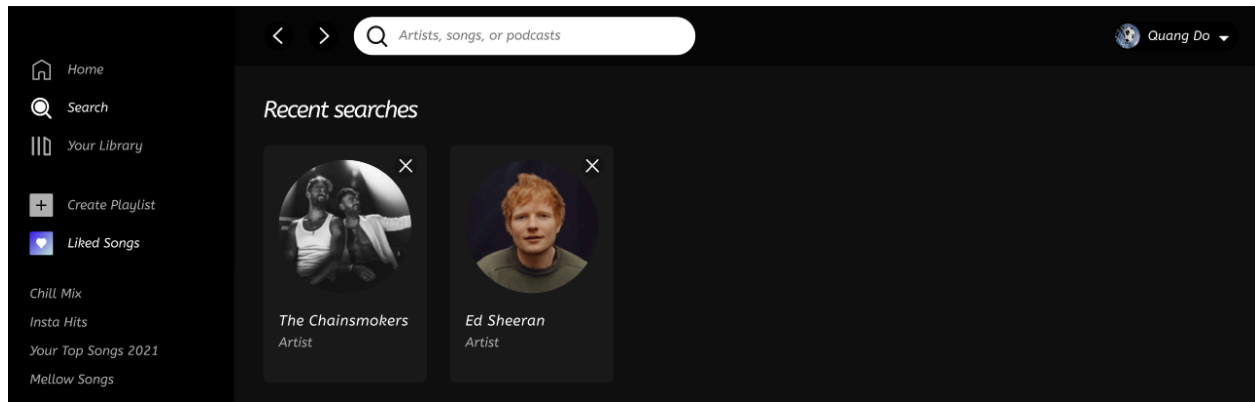


Figure : Demo search area

5.4. Play track

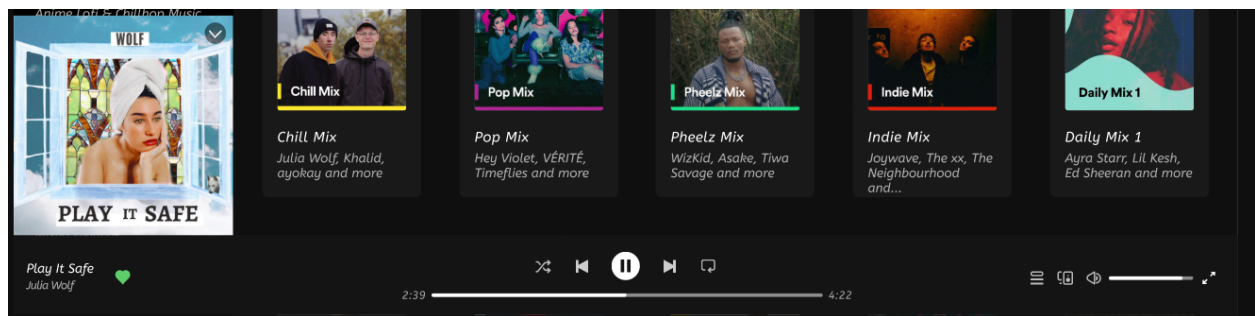


Figure : Demo track player

5.5. Create playlist

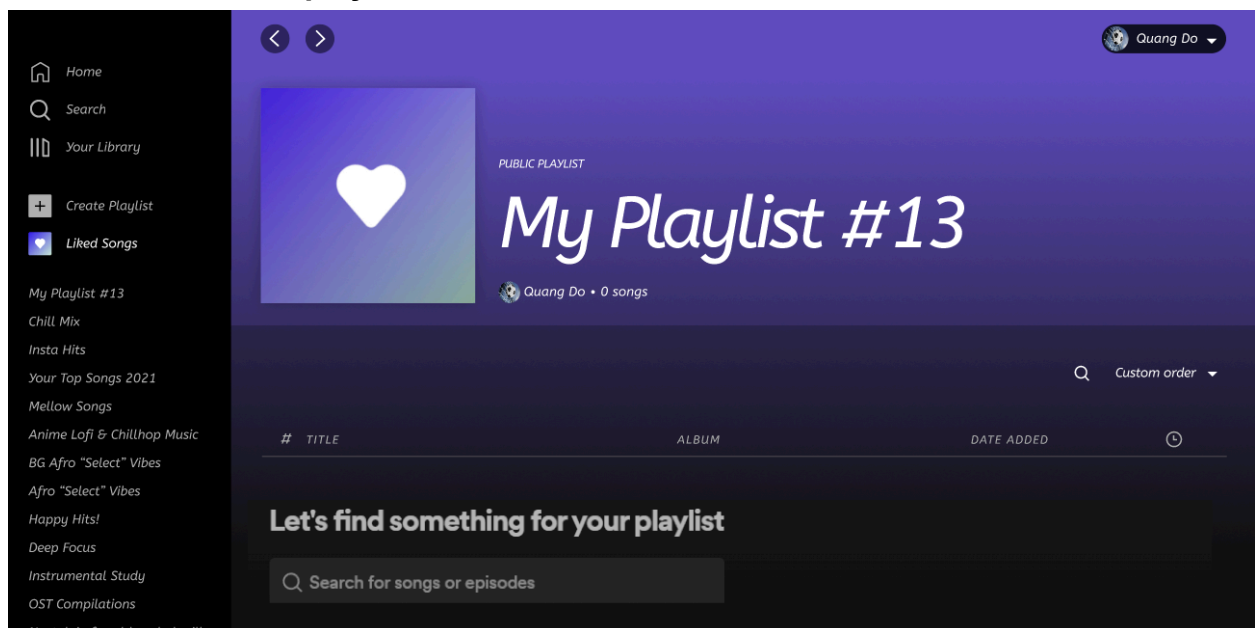


Figure : Demo create playlist

5.6. Manage playlist

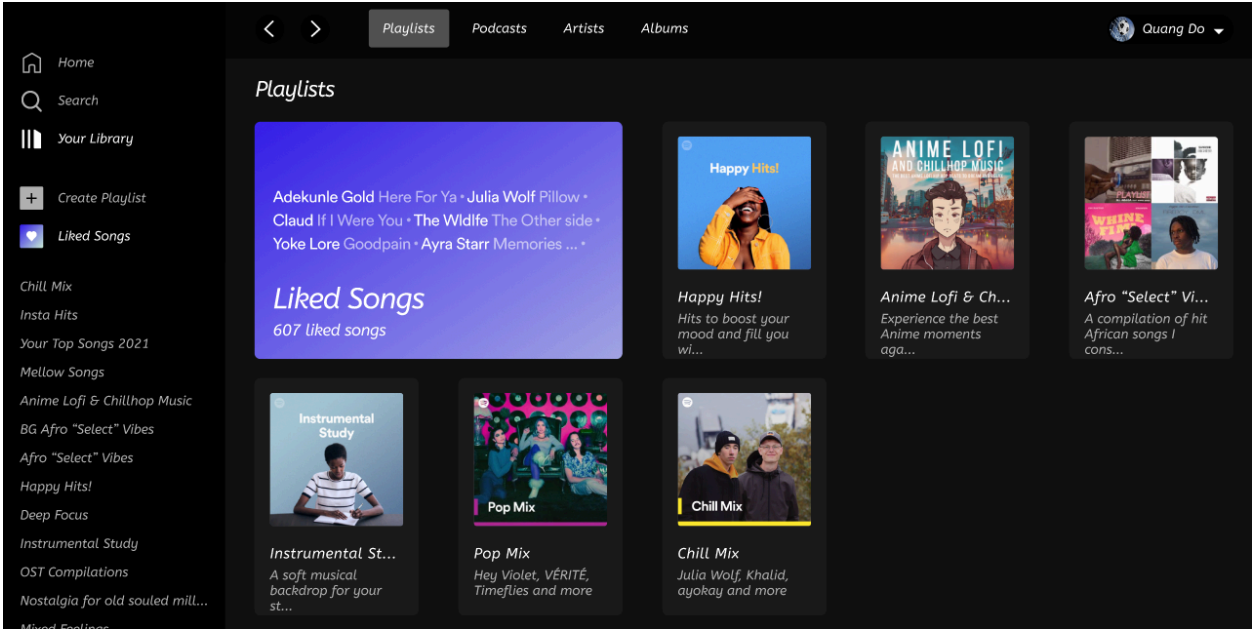


Figure : Demo playlist management

5.7. Download track

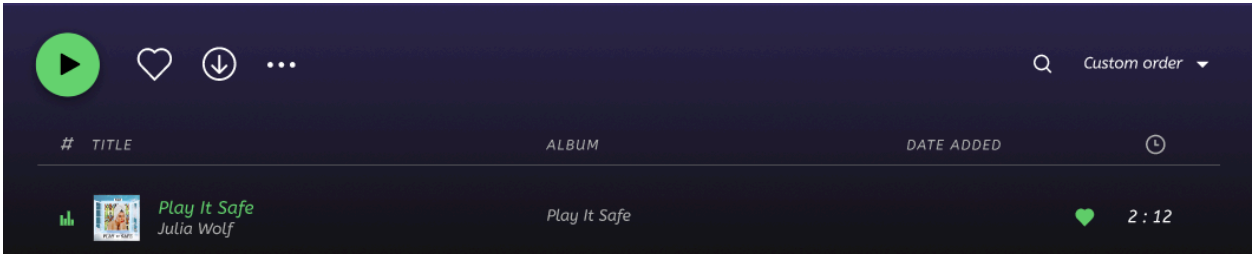


Figure : Demo download track

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 1600x1600px.
- 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)

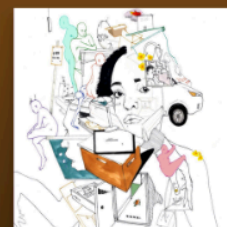
Add artists who collaborated on this entire release.

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 : Demo artist upload and manage track

6. Database Diagrams

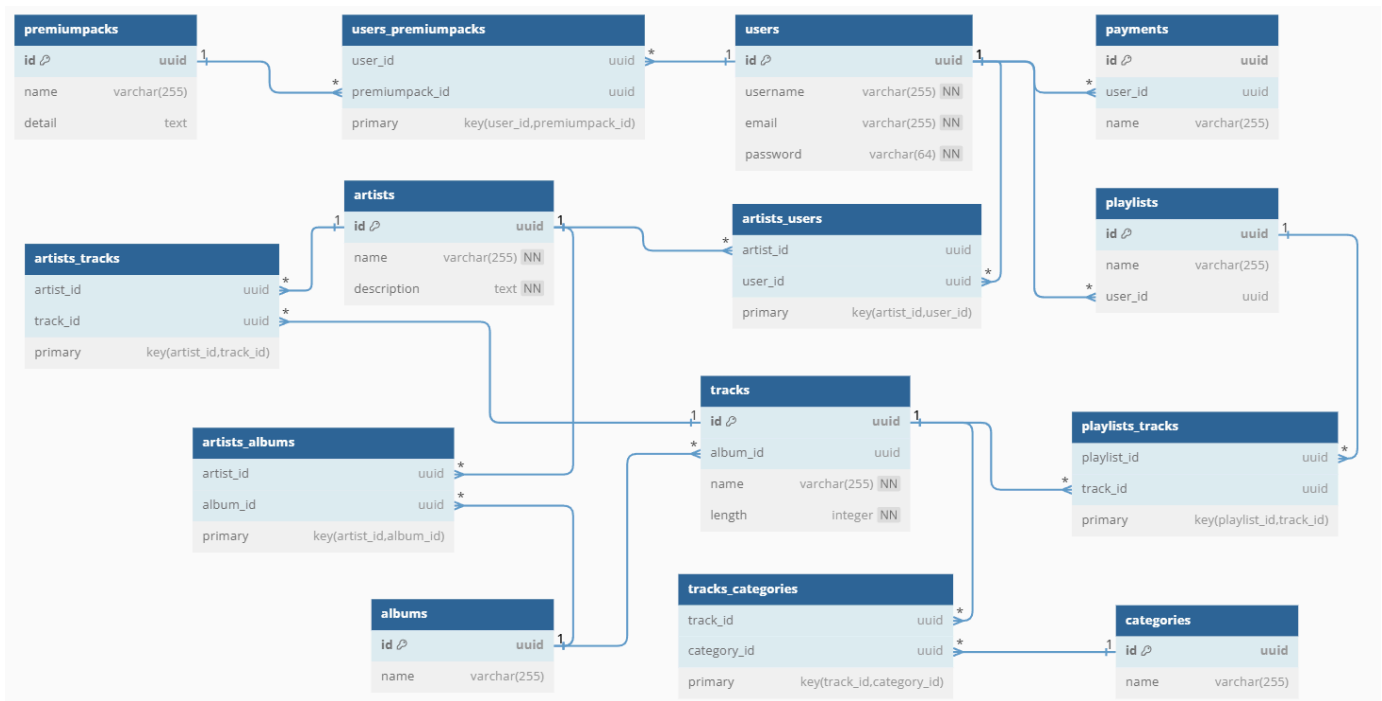


Figure : Database Diagram

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.