Software Requirements Specification

for

Music Streaming System

Version 1.0 approved

Prepared by Nguyễn Đức Khánh

Ngô Lê Hoàng

Đỗ Minh Quang

Group 5, 2324II INT2208E 23, VNU-UET

March 26, 2024

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Initial Draft	March 12th 2024	First draft with section details	0.1
Detail Enhancement	March 19th 2024	Subsection details updated Example figure given for each section.	1.0
More Detail Specification	March 26th 2024	Appendix section updated. Use Case specifications template updated.	2.0

1. Introduction

The digital revolution has transformed the way we experience music, bringing the vast universe of songs from around the globe to our fingertips. MusicStreaming stands at the forefront of this transformation, offering a seamless and personalized music streaming service to millions of users. This document delineates the requirements for MusicStreaming's software system, ensuring that it continues to meet the evolving needs of its users with efficiency, reliability, and innovation.

1.1. Purpose

The purpose of this Software Requirements Specification (SRS) is to establish a clear and detailed set of requirements for the MusicStreaming application. It serves as a contractual basis between stakeholders and the development team, ensuring a mutual understanding of functionalities, system capabilities, and performance criteria. This document will guide the design, development, and testing processes, facilitating a systematic approach to achieving project objectives. It also provides a framework for future maintenance and enhancements, ensuring the longevity and adaptability of the application in the dynamic landscape of digital music services.

1.2. Document Conventions

Headings	Arial/16 font size
Sub headings	Arial/12 font size
Body	Arial/11 font size

Numbered lists represent sequences or procedures.
Bulleted lists signify options or unordered sets.

1.3. Intended Audience and Reading Suggestions

This SRS document is intended for a diverse group of stakeholders, including but not limited to software developers, project managers, testers, marketing staff, document writers, and the end-users of the MusicStreaming application. The document is organized into 8 parts. 1. Introduction, 2. Overall Description, 3. External Interface Requirements, 4. System Features, 5. Other Nonfunctional Requirements, 6. Other Requirements and t appendix parts. All the parts are independent however reading the document sequentially helps the reader understand the MusicStreaming system better.

1.4. Project Scope

The scope of this project encompasses the development and enhancement of the MusicStreaming application, a premier music streaming service. The project aims to deliver a user-friendly platform that provides access to a vast library of music, podcasts, and videos from creators all over the world.

The key objectives include:

- Delivering high-quality audio streaming to users globally.
- Providing personalized content recommendations based on user preferences.
- Ensuring seamless integration across various devices and platforms.
- Maintaining a robust and scalable infrastructure to support a growing user base.
- Incorporating social features that allow for sharing and discovering content within user communities.

This SRS will detail the requirements necessary to achieve these objectives, including user interface design, system functionalities, performance requirements, and security measures. The document will also address the constraints, dependencies, and assumptions related to the project.

1.5. References

The following references provide the foundation and guidelines for the preparation of this Software Requirements Specification:

- IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998).
- IEEE Guide to Software Requirements Specifications (IEEE Std 1233, 1998 Edition).
- Spotify Developer Documentation and API Reference.
- Previous version documents and user feedback reports related to the Spotify application.
- Software Engineering course slides provided by lecturers.

As the project progresses, additional resources may be identified and documented in this section.

2. Overall Description

As we delve into the Overall Description, we present a bird's-eye view of the MusicStreaming application, its capabilities, and its interaction with the broader digital ecosystem. This section will pave the way for a granular look at the product's role in the competitive landscape of music streaming services

2.1. Product Perspective

The Music Streaming system is envisioned as a comprehensive digital platform designed to provide users with access to a vast library of musical content. It stands as an independent, full-featured service that allows for the streaming of music tracks, albums, and artist-curated playlists. The system is conceptualized to seamlessly integrate with users' digital ecosystems, offering compatibility across various devices and operating systems.

2.2. Product Functions

- *Music Library Access*: Users can stream music from a vast collection of tracks from various artists, genres, and languages. The service provides access to millions of songs, catering to diverse musical tastes.
- *Personalized Recommendations*: Leveraging data analytics, MusicStreaming offers personalized music recommendations based on individual listening habits, helping users discover new music and curated playlists.
- *Playlist Creation*: Users can create, share, and follow playlists, which can be customized and saved for offline listening.
- *Offline Listening*: Premium users have the option to download music and listen offline, providing flexibility and convenience.
- *High-Quality Audio*: MusicStreaming offers high-quality audio streaming options, ensuring an optimal listening experience for premium subscribers.
- *User Interface*: The application features a user-friendly interface with easy navigation, search functions, and access to different sections like 'Home', 'Search', and 'Your Library.

2.3. User Classes and Characteristics

- *General Users*: This class includes individuals who use MusicStreaming for personal entertainment. They are characterized by their diverse musical tastes and preferences, and they utilize MusicStreaming's free or premium services. General users are the largest group and include various demographics.

- *Premium Users*: These users pay a subscription fee for enhanced features such as ad-free listening, offline playback, and high-quality audio. They value uninterrupted and high-fidelity music experiences.

- *Artists and Content Creators*: This group uses MusicStreaming as a platform to publish and monetize their music and podcasts. They benefit from MusicStreaming's tools for artists, which provide insights into listener demographics and engagement.
- *Administrators:* Administrators oversee the system’s operations, manage server infrastructure, security, and updates. Administrators ensure compliance with guidelines, handle reported content; address user inquiries, troubleshoot issues, and maintain a positive user experience.

2.4. Operating Environment

2.4.1. Hardware Platform

MusicStreaming Web is accessible through web browsers, eliminating the need for specific hardware requirements. It supports integration with various commercial hardware for music streaming, such as smart speakers and AV receivers, through MusicStreaming Connect.

2.4.2. Operating Systems and Versions

MusicStreaming is compatible with major desktop and mobile operating systems, ensuring broad accessibility. It supports:

+ Desktop: Windows 7 and above, macOS Yosemite 10.10 and above, and Linux distributions via the web browser.
+ Mobile: Available on Android and iOS devices with web browser support.

2.4.3. Co-existing Software

The web application is compatible with the latest versions of major web browsers such as Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge. It also integrates with various browser extensions and plugins that do not interfere with its functionality.

2.4.4. Infrastructure Requirements

Standard web server and database configurations will suffice for the initial deployment:

Disk capacity	>=64GB
Memory (RAM)	>=2GB
Front-end	HTML, CSS , ReactJS
Back-end	FastAPI, PostgreSQL (latest versions as of March 2024)

2.5. Design and Implementation Constraints

The system runs under Windows 7 / 8 / 8.1 / 10 / 11, macOS 10.10 and above or any version of Linux.
The application is developed on ReactJS platform as frontend and FastAPI as backend.

2.6. User Documentation

- *User Manual:*
    - A comprehensive guide that explains how to use the app.
    - Includes step-by-step instructions for common tasks (e.g., searching for music, creating playlists).
    - Describes navigation, buttons, and icons.
    - Provides troubleshooting tips.
- *FAQs:*
    - Addresses common queries and issues.
    - Covers topics like account management, playback, and settings.
    - Helps users find quick solutions.
- *Tutorials and How-To Guides:*
    - Short videos or written guides demonstrating specific features.
    - Useful for visual learners.
- *Contextual Help:*

  - In-app tooltips or pop-ups that provide information relevant to the current screen.
  - Helps users learn on the go.
- *Release Notes:*
  - Summarizes changes in each app update.
  - Highlights new features, bug fixes, and improvements.
- *Contact Information:*
  - Provides details for customer support or feedback.
  - Includes email addresses, chat support, or community forums.

2.7. Assumptions and Dependencies

2.7.1. Assumptions

- *User Connectivity*: It is assumed that users have access to stable and high-speed internet connectivity to stream music without interruptions.
- *Device Capability*: Users' devices are assumed to be capable of running the MusicStreaming application, which includes having sufficient processing power, memory, and a compatible operating system.
- *Content Licensing*: It is assumed that MusicStreaming will continue to secure and maintain licensing agreements with music labels, artists, and publishers to offer a wide range of content.

2.7.2. Dependencies

- *Third-Party Services*: MusicStreaming's functionality depends on various third-party services, including payment gateways for subscription management, social media platforms for sharing content, and cloud services for hosting the application and data.
- *Hardware Compatibility*: MusicStreaming relies on the compatibility of hardware devices, such as smartphones and smart speakers, to ensure users can access the service across different platforms.
- *Regulatory Compliance*: The service is dependent on adhering to international laws and regulations related to digital services, data protection, and copyright.

3. External Interface Requirements

3.1. User Interfaces

a) **Logical Characteristics of Interface:**

- The web application shall have a responsive design capable of adapting to various screen sizes and resolutions.
- Required screen formats shall include a home page featuring recommended playlists, trending tracks, and user-specific recommendations.
- Page layouts should prioritize easy navigation with clear categories for browsing music genres, artists, albums, and playlists.
- The content of reports or menus should be customizable, allowing users to personalize their music browsing experience.

- Playback controls shall be easily accessible on the interface, including play, pause, skip, repeat, and volume adjustment functionalities.
- The interface should support the creation, management, and sharing of playlists, with options for editing and rearranging track order.

b) **Optimization for User Interaction:**

- The interface shall prioritize ease of use and intuitive navigation, ensuring that users can find and play music effortlessly.
- Do's and Don'ts guidelines shall be provided to optimize user experience, including clear labeling of buttons, intuitive icons, and consistent navigation patterns.
- Error messages shall be concise and informative, offering options for both long and short error messages based on user preferences.
- Users shall have the option to customize their interface preferences, such as choosing between light and dark themes or adjusting font sizes for better readability.
- The system shall provide tooltips or contextual help to guide users through unfamiliar features or functionalities.
- Verifiable requirements shall be established, such as specifying that a user with basic training can create a playlist and add tracks within a certain time frame.

3.2. Hardware Interfaces

- The web application shall support interfaces with various hardware components, including audio output devices such as speakers, headphones, and sound cards.
- It shall support multiple audio output channels for stereo and surround sound configurations.
- Supported devices shall include desktop and laptop computers, smartphones, tablets, and smart TVs with compatible web browsers.
- Compatibility with Bluetooth audio devices shall be provided, allowing users to stream music wirelessly to Bluetooth-enabled speakers and headphones.

3.3. Software Interfaces

- The Music Streaming system’s web interface is crafted using modern web development technologies such as HTML, CSS, and JavaScript, ensuring compatibility across various operating systems like Unix, Linux, Mac, and Windows. The system utilizes PostgreSQL databases located in data centers worldwide to store and manage vast amounts of song data, user profiles, and artist information.

- The databases hold extensive details about music tracks, including album art, track names, genres, artists, release dates, and user reviews and ratings. This data is meticulously curated by the content management system to provide users with a rich and engaging experience.

- The system also integrates with a user account management system to oversee user subscriptions, preferences, and settings. This system is responsible for managing user data, including account creation dates, subscription types, and user-generated content like playlists and favorites.

- Music Streaming system communicates with content delivery networks (CDNs) to ensure efficient distribution of music files, enabling fast and reliable streaming services. The CDN system manages the complex task of delivering high-quality audio content to users around the globe, adapting to various network conditions and user demands.

3.4. Communications Interfaces

- **Local Network Protocols:**
  - The web application shall support communication over local network protocols such as TCP/IP, UDP, and HTTP.
  - It shall utilize standard network sockets for establishing connections and transmitting data between the client and server components.
  - Support for IPv4 and IPv6 addressing schemes shall be provided to ensure compatibility with diverse network environments.
- **Data Exchange Formats:**
  - The application uses standardized data exchange format: JSON (JavaScript Object Notation) for transmitting structured data over the network.
  - Metadata for music tracks, playlists, and user preferences shall be encoded using JSON format for efficient communication between client and server.
- **API Endpoints:**
  - The web application exposes RESTful API endpoints for client-server communication, allowing clients to perform CRUD (Create, Read, Update, Delete) operations on music-related resources.
  - API endpoints shall be documented with clear specifications for request and response formats, authentication mechanisms, and error handling procedures.

4. System Features

4.1. User Accounts

4.1.1. Description and Priority

- This feature allows users to create accounts, log in, and manage personalized music experiences. Users can create playlists, save favorite tracks and albums, and potentially enjoy additional features like customized recommendations or offline listening (if applicable).
- High Priority

Benefit (High - 8):

- Increased user engagement: Logged-in users are more likely to return and spend more time on the platform.

- Personalized experience: Users can curate their own listening experience with playlists and saved favorites.
- Potential for premium features: Creates a foundation for offering paid subscriptions with additional benefits like ad-free listening or higher quality audio.

Penalty (Medium - 5):

- Limited functionality for non-registered users: Without accounts, users may have limited access to features or personalization options.
- Potential barrier to entry: Login requirements might discourage some users from casually exploring the music library.

Cost (Medium - 5):

- Development effort: Implementing a user login system, playlist management functionality, and database integration requires development resources.
- Ongoing maintenance: Maintaining the user account system and ensuring data security requires ongoing effort.

Risk (Medium - 4):

- Security considerations: Protecting user data (e.g., usernames, passwords) necessitates robust security measures.
- Login complexity: A complex login process can frustrate users. It's important to strike a balance between security and ease of use.

4.1.2. Stimulus/Response Sequences

Use Case 1: User Creates an Account

+ **User Action:** Navigates to the signup page and enters email address, password, and potentially other information (username, display name etc.).
+ **System Response:**
    - Validates user input (e.g., email format, password strength).
    - Checks for existing email address.
    - If valid, creates a new user account in the database and sends a confirmation email (optional).
    - Displays a success message and redirects the user to the login page or dashboard.

Use Case 2: User Logs In

+ **User Action:** Navigates to the login page and enters email address and password.
+ **System Response:**
    - Verifies email address and password against stored credentials.
    - If successful, creates a session for the user and redirects them to their personalized dashboard or music library.
    - If unsuccessful, displays an error message with guidance (e.g., incorrect email/password, account disabled).

Use Case 3: User Logs Out

+ **User Action:** User clicks the logout button and confirms logout.
+ **System Response:**
    - System displays a logout confirmation message (optional).
    - System removes the user's login information from memory.
    - System redirects the user to the home page or login page.

Use Case 4: Modify Account Details

+ **User Action:** Click on the “Edit account details” link, after that enter new information and finally click on save button.
+ **System Response:**
    - System displays the edit account details form.
    - System validates the new information.
    - System updates the user's account information.
    - System displays a confirmation message.

Use Case 5: View User Information

+ **User Action:** Click on the “View user information” link, after that users view their informations
+ **System Response:**
    - System displays the account details page.
    - The account details page displays the user's personal information, such as name, email, account creation date, etc.

User Case 6: Premium Account

+ **User Action:**
    - User clicks on the "Subscribe to Premium" button.
    - User selects the desired subscription plan.
    - User enters payment information.
    - User clicks on the "Pay" button.
+ **System Response:**
    - System displays the Premium subscription page.
    - System displays the available subscription plans with their corresponding features and access.
    - System processes the user's payment information.
    - System activates the Premium function for the user.
    - System displays a confirmation message.

4.1.3. Functional Requirements

- Requirement 1: User Registration
    - Author: User
    - Input: User fills sign in fields: username, email, password, captcha, etc. User clicks on the submit button.
    - Server: Validate information, add new user records to database.
    - Output: If the account creation is successful, redirects the user to the login view, indicating successful registration.
    - Alternative Flow: If there are validation errors or database constraints (e.g., username or email already exists), display appropriate error messages to the user, prompting them to correct the information and resubmit the registration form.
- Requirement 2: User Log In
    - Author: User
    - Input: User provides login credentials, including email and password, in the designated login fields. User clicks on the login button.

    - Server: Verifies if the provided email exists in the database and if the associated password matches the one provided by the user.
    - Output: If the authentication is successful, redirect the user to the application's main dashboard or designated landing page
    - Alternative flow: If the email or password is incorrect, display an error message indicating authentication failure and prompt the user to retry logging in with the correct credentials.
- Requirement 3: User Log Out
    - Author: User
    - Input: User initiates the logout process by clicking on the logout button or link within the application interface.
    - Server: Clears the user's session data and authentication tokens associated with the current session.
    - Output: Redirects the user to the application's login page or a designated landing page indicating successful logout.
- Requirement 4: Modify Account Details
    - Author: User
    - Input: User navigates to the account settings or profile management section within the application. User provides updated account details such as username, email, password, or any other customizable fields. User submits the updated account details by clicking on the "Save Changes" or similar button.
    - Server: Validates the updated account details to ensure they meet the required criteria (e.g., valid email format, password strength). Updates the user's account records in the database with the provided modifications.
    - Output: If the account details are successfully updated, display a confirmation message indicating the changes have been saved.
    - Alternative Flow: If there are validation errors or any issues preventing the update, display appropriate error messages to the user, prompting them to correct the information and resubmit the form.
- Requirement 5: View User Information
    - Author: User
    - Input: User accesses the profile or account settings section within the application.
    - Server: Retrieves the user's information from the database, including username, email and any other relevant details.
    - Output: Provides options for the user to edit or modify their account details if desired. Ensures that sensitive information is not displayed in plaintext and is appropriately masked or hidden.
    - Alternative Flow: Displays an error message indicating that the user is not authorized to view the requested information.

4.2. Track feature

4.2.1. Description and Priority

- This feature enables artists to upload tracks, submit their music or audio content, and allows users to play and download tracks.
- High Priority

4.2.2. Stimulus/Response Sequences

Use Case 1: Upload Track

+ **User Action:** Artist navigates to the Upload Track section within their account and provides metadata for the track
+ **System Response:**
    - The system validates the file format, checks for duplicates, and ensures that all required information is provided.
    - The track is uploaded to Spotify’s database.
    - The artist receives a confirmation message indicating successful upload.

Use Case 2: Download Track

+ **User Action:**
    - User clicks on the "Download Track" button.
    - System displays a confirmation dialog.
    - User clicks on the "Download" button in the confirmation dialog.
+ **System Response:**
    - System checks if the user has permission to download the track.
    - If the user has permission, the system starts downloading the track.
    - System displays a download progress bar.
    - When the download is complete, the system notifies the user.
    - System opens the folder containing the downloaded file.

Use Case 3: Modify Track Information

+ **User Action:**
    - User clicks on the "Modify Track Information" button.
    - User enters new information for the track (name, artist, album, etc.).
    - User clicks on the "Save" button.
+ **System Response:**
    - System displays the edit track information form.
    - System validates the new information.
    - System updates the track information.
    - System displays a confirmation message.

Use Case 4: Play Track

+ **User Action:** Click on the button “▶” to play the track.
+ **System Response:**
    - System starts playing the track.
    - System displays the information of the track being played (name, artist, album, etc.).
    - System displays playback controls (play, pause, rewind, fast forward).

4.2.3. Functional Requirements

- Requirement 1: Track upload
    - Author: Artist
    - Input: Track file (MP3, MP4, WAV) and information about the uploaded track, such as title, artist name, genre, duration, etc.
    - Server: validate file format, store uploaded tracks

- Output: If successful, display message “Track successfully uploaded.” If the uploaded file format is unsupported, prompt the user with a message explaining the supported formats and request re-upload. If the upload process is interrupted due to network issues or server downtime: Display a message informing the user about the interruption and advise them to retry the upload later.

- Requirement 2: Track download
    - Author: User
    - Input: User clicks on download button, then frontend sends request which contains the ID of the track to be downloaded.
    - Server: Access control to ensure only authorized users can download tracks. Bandwidth management to handle concurrent downloads efficiently.
    - Output: Upon successful authorization and bandwidth allocation, the server delivers the requested track to the user, ensuring a seamless download experience. If error occurs, provide options for the user to try downloading the track again later or to explore other available tracks.

- Requirement 3: Modify Track Information
    - Author: Artist
    - Input: The artist navigates to the track management section within the application and selects a specific track to modify. The artist provides updated information such as track title, album, genre, or any other relevant details.
    - Server: The server validates the updated track information to ensure it meets the required criteria and adheres to any constraints (e.g., character limits, data formats). It then updates the corresponding record in the database with the provided modifications.
    - Output: If the track information is successfully updated, the server confirms the changes and notifies the user with a success message. Additionally, it may display the updated track details to the user for verification. If there are any validation errors or issues preventing the update, the server notifies the user with appropriate error messages, indicating the nature of the problem and prompting them to correct the information accordingly.

- Requirement 4: Play Track
    - Author: User
    - Input: The user selects a specific track to play from the application's library or playlist. This selection can be made by clicking on the track's title, album cover, or a designated play button.
    - Server: Upon receiving the play request from the frontend, the server checks the availability and readiness of the selected track for playback. It may also authenticate the user's access rights to ensure they are authorized to play the track.
    - Output:If the track is available and the user is authorized to play it, the server streams the audio data to the user's device in real-time. This enables seamless

playback without the need for downloading the entire track beforehand.In case the track is unavailable or there are any playback errors (e.g., network issues, corrupted audio data), the server notifies the user with an appropriate error message. It may also provide suggestions or alternative tracks for playback.

4.3. Playlist feature

4.3.1. Description and Priority

- This feature allows users to manage personalized music experiences. Users can create playlists, save favorite tracks and albums, and potentially enjoy additional features like customized recommendations or offline listening (if applicable).
- High priority

4.3.2. Stimulus/Response Sequences

Use Case 1: Create Playlist

+ **User Action:** User selects Create New Playlist option and enters a unique playlist name
+ **System Response:**
    - The system prompts the user to provide a playlist name and an optional description for the new playlist.
    - The system validates the playlist name and description.
    - If valid, it creates an empty playlist with the specified name and description.

Use Case 2: Add Tracks to a Playlist

+ **User Action:** Selects tracks from the music library and adds them to a chosen playlist.
+ **System Response:**
    - Updates the playlist data in the database to include the newly added tracks.
    - Reflects the changes in the user interface, showing the added songs within the playlist.

Use Case 3: Manages Playlist

+ **User Action:** Edits a playlist name, description, or rearranges the order of tracks within a playlist.
+ **System Response:**
    - Updates the playlist data in the database based on user actions.
    - Reflects the changes in the user interface, displaying the updated playlist details.
    - Optionally allows deleting tracks from the playlist or deleting the entire playlist.

4.3.3. Functional Requirements

- Requirement 1: Create playlist
    - Author: User
    - Input: The user navigates to the playlist creation section within the application interface and initiates the process of creating a new playlist. The user provides a name for the playlist and may optionally add a description or select a cover image.
    - Server: Upon receiving the playlist creation request from the frontend, the server validates the provided playlist name to ensure it meets any required criteria (e.g.,

minimum length, uniqueness). It also checks the user's authentication status to ensure they are logged in and authorized to create playlists. If the provided playlist name is valid and the user is authenticated, the server creates a new playlist record in the database. It associates the playlist with the user's account and stores any additional metadata provided by the user, such as description or cover image.

  - Output: Upon successful creation of the playlist, the server confirms the creation process and notifies the user with a success message. It may also redirect the user to the newly created playlist's page or display a confirmation within the application interface. If there are any validation errors or issues preventing the creation of the playlist (e.g., duplicate name, invalid characters), the server notifies the user with an appropriate error message. It prompts the user to correct the input and resubmit the playlist creation form.

- Requirement 2: Add track to playlist
  - Author: User
  - Input: The user selects a specific track from the application's library or playlist and chooses the option to add it to an existing playlist. The user may be presented with a dropdown menu or a list of their existing playlists to choose from.
  - Server: Upon receiving the request to add a track to a playlist, the server verifies the user's authentication status to ensure they are logged in and authorized to modify playlists. It also checks the validity of the selected track and the target playlist. If the user is authenticated and the selected track and playlist are valid, the server updates the database to associate the track with the selected playlist. It may also handle any additional metadata associated with the track in the context of the playlist, such as track order or user-specific preferences.
  - Output: Upon successful addition of the track to the playlist, the server confirms the action and notifies the user with a success message. It may provide feedback within the application interface, such as updating the playlist view to reflect the newly added track. If there are any issues preventing the addition of the track to the playlist (e.g., invalid track or playlist, authorization failure), the server notifies the user with an appropriate error message. It prompts the user to correct the issue and retry the operation.
- Requirement 3: Manage playlist
  - Author: User
  - Input: The user navigates to the playlist management section within the application interface, where they can perform various actions such as renaming playlists, changing playlist settings, reordering tracks, or deleting playlists.
  - Server: Upon receiving the user's request to manage a playlist, the server verifies the user's authentication status to ensure they are logged in and authorized to modify playlists. It also checks the validity of the requested operation and the target playlist. Depending on the specific action requested by the user (e.g., renaming playlist, reordering tracks), the server updates the database

accordingly. It ensures data integrity and consistency while applying the requested changes to the playlist.

- Output: Upon successful completion of the requested playlist management action, the server confirms the operation and notifies the user with a success message. It may also provide feedback within the application interface, such as updating the playlist view to reflect the changes made. If there are any issues preventing the completion of the requested playlist management action (e.g., authorization failure, database error), the server notifies the user with an appropriate error message. It prompts the user to correct the issue and retry the operation.

4.4. The search facility

4.4.1. Description and Priority

- The search facility is a critical component of the music streaming system, allowing users to discover and access their favorite music efficiently. Users can search for specific tracks, albums, artists, genres, playlists, and more. The search functionality enhances user experience by providing relevant results based on user queries.
- High Priority

4.4.2. Stimulus/Response Sequences

Use Case 1: Basic Track, Genre-Based, Artist Search

+ **User Action:** User enters a search query, a specific genre or an artist's name in the search bar.
+ **System Response:**
    - The user sees a list of track results, including titles, artists, and album information.
    - Users can explore popular tracks or discover lesser-known gems within that genre.
    - The system retrieves the artist's profile, including their top tracks, albums, and related artists.

Use Case 2: Advanced Filters

+ **User Action:** User applies advanced filters (e.g., release date, popularity, explicit content) to their search.
+ **System Response:**
    - The system refines the search results based on the selected filters.
    - Users can find recently released tracks, popular hits, or family-friendly content.

Use Case 3: History Search Bar

+ **User Action:** User scrolls through the history or uses search filters (type some words in search bar) and selects a specific track from the history
+ **System Response:**
    - The system displays a chronological list of recently played tracks.
    - The system starts playing the chosen track, allowing the user to enjoy their music.

4.4.3. Functional Requirements

- Requirement 1: Track, Genre-Based, Artist Search
    - Author: User
    - Input: The user enters a search query into the application interface, specifying whether they are searching for a track, a genre, or an artist.
    - Server: Upon receiving the user's search query, the server processes the input and determines the type of search requested (track, genre-based, or artist). It then retrieves relevant data from the database, including tracks, genres, and artists.
    - Output: The server presents the user with search results based on the type of search performed. These results include relevant information such as track titles, artists, albums, genres, or other related content. The presentation ensures clarity and ease of navigation for the user.
- Requirement 2: Advanced Filter
    - Author: User
    - Input: The user accesses the advanced filter option within the application interface, which allows them to refine search results or narrow down content based on specific criteria. This could include filters for genre, release date, duration, popularity, and more.
    - Server: Upon receiving the user's request to apply advanced filters, the server processes the selected filter criteria and retrieves relevant data from the database. It applies the specified filters to the dataset, ensuring that only content matching the user's criteria is included in the results.
    - Output: The server presents the user with filtered search results based on the selected criteria. It ensures that the filtered content meets the user's preferences and provides options for further refinement or exploration. If there are no search results matching the user's selected filter criteria, the server notifies the user with a message indicating that there are no matches found. It may suggest adjusting the filter settings or provide alternative suggestions for finding relevant content.
- Requirement 3: History Search Bar
    - Author: User
    - Input: The user interacts with the history search bar feature within the application interface. This feature allows the user to access their search history and quickly repeat previous searches.
    - Server: Upon user interaction with the history search bar, the server retrieves the user's search history data from the database. It organizes the search history in chronological order, with the most recent searches appearing at the top of the list.
    - Output: The server presents the user with their search history, displayed in the history search bar dropdown menu or a separate section of the interface. Each entry in the search history includes details such as the search query and the date/time of the search.

4.5. Recommendation System Feature

4.5.1. Description and Priority

- The Recommendation System is a component of the Music Streaming web application. It enhances user engagement by providing personalized music recommendations based on their listening history, preferences, and behavior. Users receive tailored song suggestions, discover new artists, and enjoy a curated music experience.

- Medium Priority

4.5.2. Stimulus/Response Sequences

Use Case 1: Recommended Playlists

+ **User Action:** User logs in to the web and navigates to the "Recommended Playlists" section.
+ **System Response:**
    - The system analyzes the user's listening history, favorite genres, and recently played tracks.
    - It generates personalized playlists containing recommended songs based on the user's preferences.

Use Case 2: Genre Exploration

+ **User Action:** User selects a genre from the genre filter.
+ **System Response:**
    - The system suggests popular tracks within the chosen genre.
    - Users can discover new songs and expand their musical horizons.

Use Case 3: Continuous Learning

+ **User Action:** User skips a track they dislike or thumbs-up a song they enjoy.
+ **System Response:**
    - The system adapts its recommendations based on user feedback.
    - It learns from interactions to refine future suggestions.

4.5.3. Functional Requirements

- Requirement 1: Recommend Playlist
    - Author: User
    - Input: The user navigates to the playlist recommendation section within the application interface, or the system automatically suggests playlists based on the user's listening history, preferences, or behavior.
    - Server: Based on the analysis, the server generates a list of recommended playlists that are likely to be of interest to the user. It ensures that the recommended playlists are diverse and cater to different moods, genres, or themes.
    - Output: The server presents the user with a list of recommended playlists, along with brief descriptions or preview tracks to entice the user's interest.
- Requirement 2: Genre exploration
    - Author: User
    - Input: The user expresses interest in exploring different music genres within the application interface. This can be done by selecting a genre from a dropdown menu, browsing genre-specific sections, or searching for specific genres.
    - Server: Upon receiving the user's request for genre exploration, the server retrieves genre-related data from the database, including information about artists, albums, and tracks associated with the selected genre. It may also utilize

external APIs or databases to enhance the genre exploration experience with additional content and recommendations.

- Output: The server presents the user with a curated selection of content related to the chosen genre, such as top tracks, featured artists, and popular albums. It ensures that the content is relevant to the selected genre and aligns with the user's preferences and interests.

4.6. Payment

4.6.1. Description and Priority

- Payment component handles financial transactions related to user subscriptions, premium accounts, and in-app purchases. Users can securely make payments, upgrade their accounts, and enjoy premium features seamlessly.
- Medium Priority

4.6.2. Stimulus/Response Sequences

Use Case 1: Upgrade account

+ **User Action:** User navigate to the account settings and select the “Upgrade to Premium” option.
+ **System Response:**
    - The system prompts the user to choose a billing plan (monthly or yearly).
    - The user enters their payment details (credit card information or other payment methods).
    - The system processes the payment and updates the user’s account to premium status.

Use Case 2: Transaction History

+ **User Action:** User choose the payment history section in Setting
+ **System Response:**
    - The system displays a list of past transactions, including dates, amounts, and descriptions.
    - The user can download receipts for each transaction.

4.6.3. Functional Requirements

- Requirement 1: Upgrade account
    - Author: User
    - Input: The user navigates to the account settings or subscription management section within the application interface and selects the option to upgrade their account. The user may be presented with different subscription tiers or upgrade options to choose from.
    - Server: Upon receiving the user's request to upgrade their account, the server verifies the user's authentication status to ensure they are logged in and authorized to perform account upgrades. It also checks the validity of the selected upgrade option and ensures that the user's current subscription status allows for upgrades. If the user is authenticated and eligible for an account upgrade, the server processes the upgrade request and updates the user's

    account records in the database accordingly. This may involve changing the user's subscription tier, updating billing information, or applying any additional features or benefits associated with the upgraded account.
  - Output: Upon successful completion of the account upgrade process, the server confirms the upgrade and notifies the user with a success message. It may also provide details about the upgraded subscription tier and any new features or benefits that come with it. If there are any issues preventing the completion of the account upgrade process (e.g., payment failure, server error), the server notifies the user with an appropriate error message. It prompts the user to correct the issue and retry the upgrade process or provides assistance in resolving the issue.
- Requirement 2: Transaction History
  - Author: User
  - Input: The user navigates to the transaction history section within the application interface to view a record of their past transactions.
  - Server: Upon receiving the user's request for transaction history, the server verifies the user's authentication status to ensure they are logged in and authorized to access their transaction records. It retrieves the relevant transaction data from the database based on the user's account information.
  - Output: The server presents the user with a list or summary of their past transactions, displaying relevant details for each transaction such as payment dates, amounts, transaction IDs, and descriptions. It ensures that the transaction history is presented in a clear and organized manner for easy comprehension by the user.

5. Other Nonfunctional Requirements

5.1. Performance Requirements

The application shall adhere to the following performance requirements to ensure a responsive and reliable user experience:

- *Response Time*: Display the initial content within 2 seconds of user interaction under standard operating conditions. Subsequent content, such as songs and playlists, shall load within 3 seconds under the same conditions.

- *System Availability:* The service shall be available 24/7, with a monthly uptime of at least 99.99%, excluding scheduled maintenance windows.

- *Scalability:* Support up to 1000 concurrent users without degradation of performance.

- *Latency*: Have a network latency of less than 150 milliseconds for 95% of all transactions.

- *Capacity*: The database shall handle at least 10 thousand songs, with the ability to scale as the library grows.

- *Resource Utilization*: The application shall optimize resource usage, with the backend services not exceeding 70% CPU utilization and 80% memory utilization under peak load conditions.

5.2. Safety Requirements

- *Content Moderation:* Implement robust content moderation policies to prevent the distribution of harmful or inappropriate content. This includes automated filtering, user reporting mechanisms, and a dedicated review team.

- *Data Protection:* User data shall be protected in accordance with the highest industry standards, including encryption of sensitive data, secure data storage solutions, and regular security audits.

- *User Privacy:* Adhere to strict privacy policies, ensuring that user data is not shared without explicit consent and that users have control over their personal information.

- *Safety Features:* offer safety features such as parental controls, content filters, and privacy settings, allowing users to customize their experience and manage risks.

- *Safety Education:* Provide resources and guidance to educate users about safe online practices and the responsible use of the platform.

- *Incident Response:* Have a clear and accessible process for users to report safety concerns and incidents, with a commitment to timely and appropriate action.

5.3. Security Requirements

- *User Authentication and Authorization*
  - Use strong password policies and securely hash passwords.
  - Define access control rules based on user roles.
- *Data Protection*
  - Encrypt sensitive data during transmission (HTTPS) and at rest.
  - Use industry-standard encryption algorithms.
  - Validate and sanitize user inputs to prevent attacks.
- *Secure APIs*
  - Implement rate limiting and authenticate API requests.
  - Validate input parameters.
- *Infrastructure Security*
  - Regularly update server software.

  - Harden server configurations.
  - Set up intrusion detection systems and incident response procedures.
- *Third-Party Dependencies*
  - Use reputable libraries.
  - Monitor vulnerabilities.

5.4. Software Quality Attributes

- *Design Qualities:*
  - Modifiability: This refers to how easily the software can be modified or updated without causing unintended side effects. A well-designed software system should be flexible and allow for changes to be made efficiently.
  - Maintainability: This is the ease with which the software can be maintained over time. It includes aspects such as code readability, documentation, and adherence to coding standards.
  - Testability: Testability measures how easily the software can be tested to ensure its correctness and reliability. A well-designed software system should facilitate comprehensive testing at various levels, from unit tests to system tests.

- *Runtime Qualities:*
  - Performance: Performance relates to the speed and responsiveness of the software while executing tasks. It includes factors such as latency, throughput, and resource utilization.
  - Scalability: Scalability refers to the ability of the software to handle increasing workload or user demand by adding resources such as servers or processing power.
  - Reliability: Reliability measures the ability of the software to perform consistently under varying conditions without failures or errors. It includes aspects such as fault tolerance and error recovery mechanisms.

- *System Qualities:*
  - Security: Security involves protecting the software from unauthorized access, data breaches, and other malicious activities. It encompasses authentication, authorization, encryption, and other security measures.
  - Availability: Availability refers to the accessibility of the software to users when needed. It involves minimizing downtime and ensuring continuous operation through redundancy and fault tolerance mechanisms.
  - Interoperability: Interoperability measures the ability of the software to interact and integrate seamlessly with other systems or components. It includes support for standard protocols and data formats.

- *User Qualities:*

- Usability: Usability refers to how easy and intuitive the software is to use for its intended users. It involves aspects such as user interface design, navigation, and user feedback.
- Accessibility: Accessibility ensures that the software can be used by people with disabilities, including those with visual, auditory, motor, or cognitive impairments. It involves providing alternative means of interaction and adhering to accessibility standards.
- Acceptability: Acceptability measures the extent to which the software meets the expectations and requirements of its users. It involves gathering user feedback and incorporating user preferences into the design and development process.

5.5. Business Rules

- *User Roles and Permissions:*
  - Users can search for and play music, create and manage playlists, follow artists and other users and personalize their music recommendations
  - Artists and Content Creators can upload and manage their music content, view analytics related to their tracks and have additional features (e.g., artist profiles, promotional tools).
  - Administrators need to manage user accounts and permissions, handle copyright infringement claims and monitor system performance and security
- *Content Licensing and Copyright:*
  - Ensure that only licensed music content is available for streaming.
  - Enforce takedown procedures for unauthorized content.
- *User Behavior and Community Guidelines:*
  - Enforce community guidelines to prevent abusive behavior (e.g., hate speech, harassment).
  - Implement reporting mechanisms for inappropriate content.
  - Address user complaints and disputes.
- *Quality of Service (QoS):*
  - Prioritize premium users during peak times to ensure smooth streaming.
  - Limit free users’ access to maintain service quality.
  - Implement adaptive streaming based on network conditions.
- *Privacy and Data Protection:*
  - Protect user data (e.g., personal information, listening history).
  - Obtain explicit consent for data collection and personalized recommendations.

6. Other Requirements

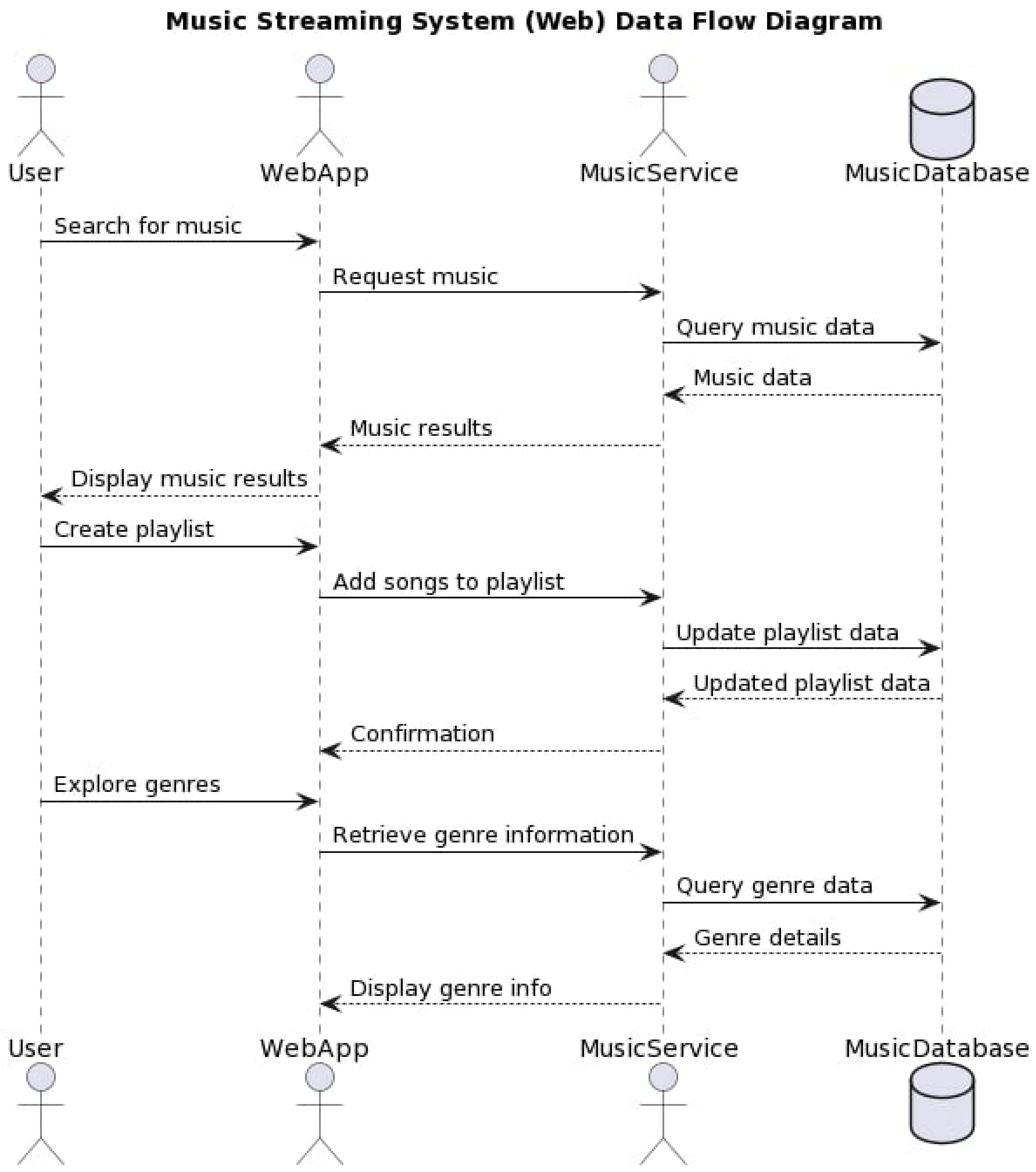
The system will incorporate stringent security protocols, including regular audits, to safeguard user and system data. Content management will be streamlined through an advanced backend, supporting copyright adherence and royalty distribution. Scalability and performance are key, with the system designed to handle growth and maintain service quality. Lastly, legal compliance and ethical considerations will be at the forefront, ensuring content legality and data transparency.

Appendix A: Glossary

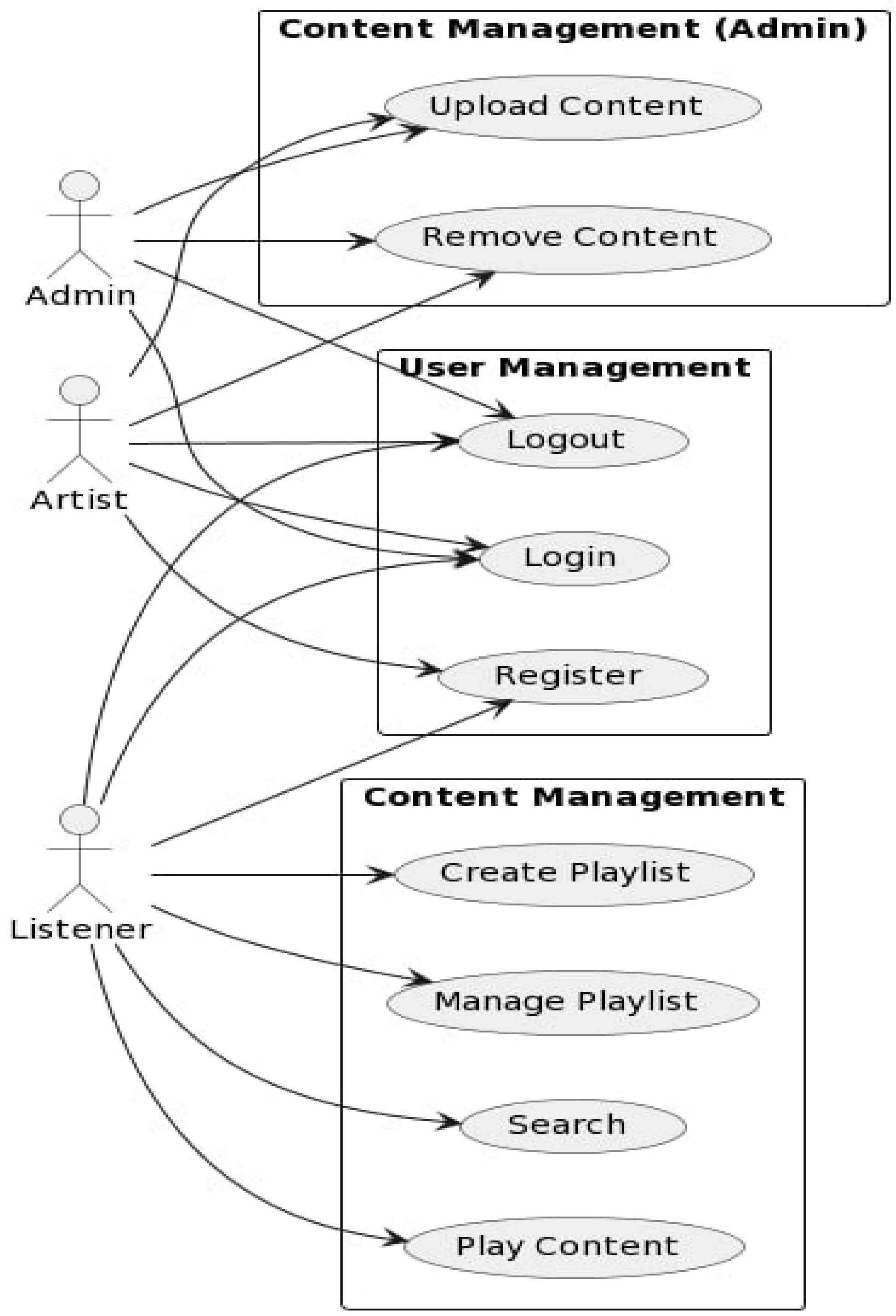
Term	Definition
API	Application Programming Interface. A communication protocol enabling interaction between servers.
Authentication	The process of proving that you are registered and have the relevant permissions to access specific data.
FAQs	Frequently Asked Questions. A list of common questions and answers that address typical user concerns and issues related to the topic.
Content Delivery Networks	A network of servers distributed globally that work together to provide fast delivery of internet content, including music streaming.

Appendix B: Analysis Models

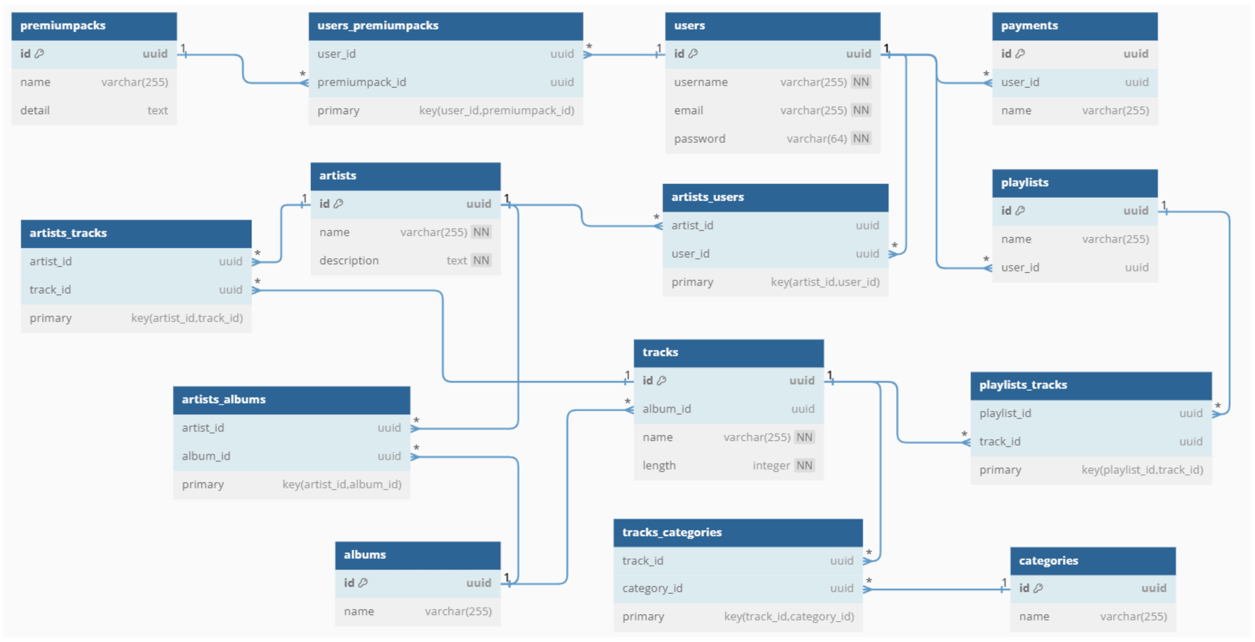
I. Data flow diagram



II. Overall Use Case Diagram



III. Database Design


premiumpacks
id uuid
name varchar(255)
detail text
users_premiumpacks
user_id uuid
premiumpack_id uuid
primary key(user_id,premiumpack_id)
users
id uuid
username varchar(255) NN
email varchar(255) NN
password varchar(64) NN
payments
id uuid
user_id uuid
name varchar(255)
artists
id uuid
name varchar(255) NN
description text NN
artists_tracks
artist_id uuid
track_id uuid
primary key(artist_id,track_id)
artists_users
artist_id uuid
user_id uuid
primary key(artist_id,user_id)
playlists
id uuid
name varchar(255)
user_id uuid
tracks
id uuid
album_id uuid
name varchar(255) NN
length integer NN
playlists_tracks
playlist_id uuid
track_id uuid
primary key(playlist_id,track_id)
artists_albums
artist_id uuid
album_id uuid
primary key(artist_id,album_id)
albums
id uuid
name varchar(255)
tracks_categories
track_id uuid
category_id uuid
primary key(track_id,category_id)
categories
id uuid
name varchar(255)
1
*