Group 05 - AlgoRhythm

CANARY MUSIC STREAMING SERVICE
Software Architecture Document

Version 1.3

Revision History

Date	Version	Description	Author
17/11/2024	1.0	● Initialization of document ● Architectural goals & constraints ● Use-case models	Khưu Thành Thiện
19/11/2024	1.1	● Implementation view ● Deployment	Khưu Thành Thiện
20/11/2024	1.2	● Logical view	Khưu Thành Thiện
12/12/2024	1.3	● Included modules in Implementation view	Khưu Thành Thiện

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document outlines the design and structure of the system, as well as significant decisions being made to the development process. It is intended to facilitate general understanding of the system's architecture, guiding its development, implementation, and future enhancements.

1.2 Scope

This **System Architecture Document** provides an architectural overview of the **Canary Music Streaming Service** - a group project by **AlgoRhythm.**

2. Architectural Goals and Constraints

This section covers the key requirements and system constraints that have substantial impact on the architecture:

1. **Performance**
    - The system must be able to handle at least 1000 accesses per second with less than 4 seconds of response time.
    - The system must be stable on multiple web browsers, prioritizing the more popular ones (Google Chrome, Opera, Microsoft Edge, Firefox, etc.)
    - The system is available for accessing 24/7.
    - The system response time for basic features must take no more than 3 seconds.
    - User’s personal information must be taken into account and securely encrypted.
    - Legal terms and policies must be respected.
    - The system’s database must be able to contain at least 10,000 songs.
    - User Interface must be comprehensible and responsive.
2. **Development Platform**

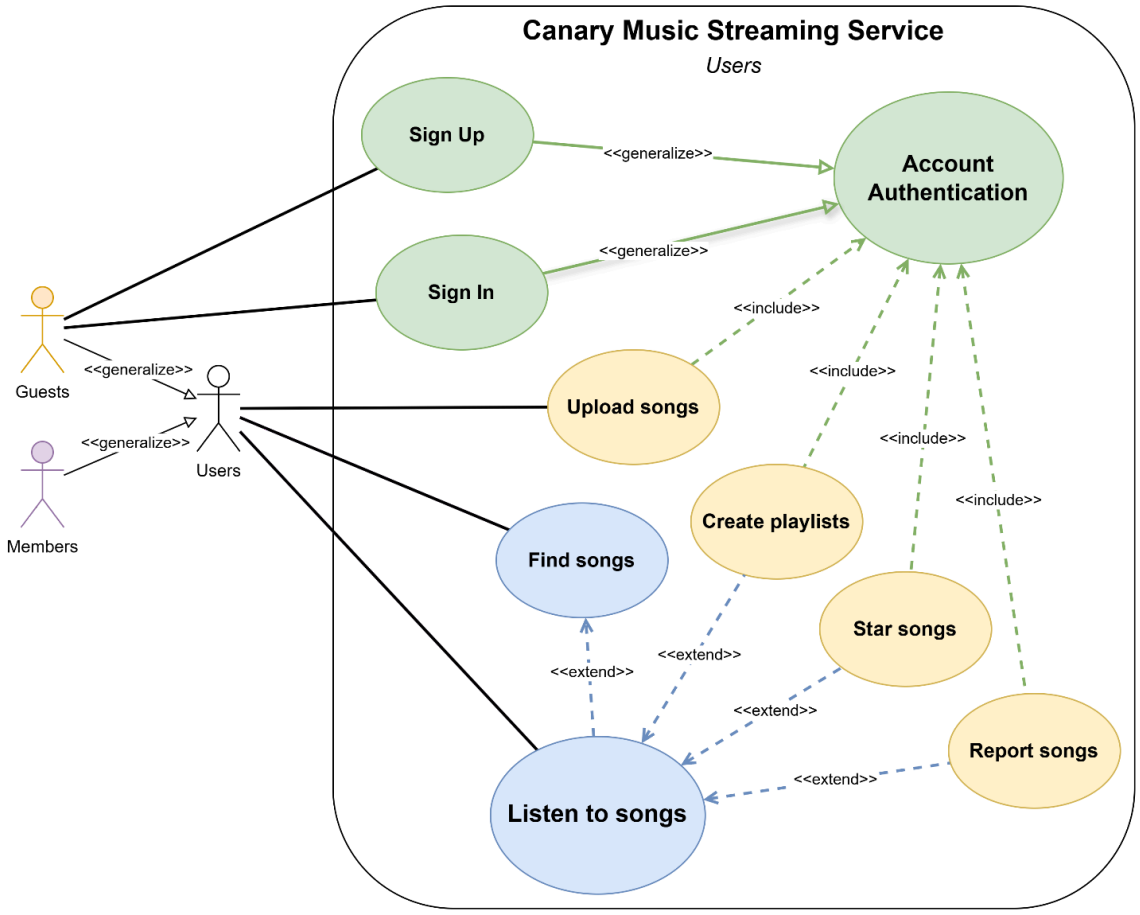
    This project uses **Wix** - a cloud-based, no-code website builder for the entire development process, including database management. **Wix Studio** will be used as the development environment.
3. **Constraints**

    Non-registered accounts will have limited access to the service.

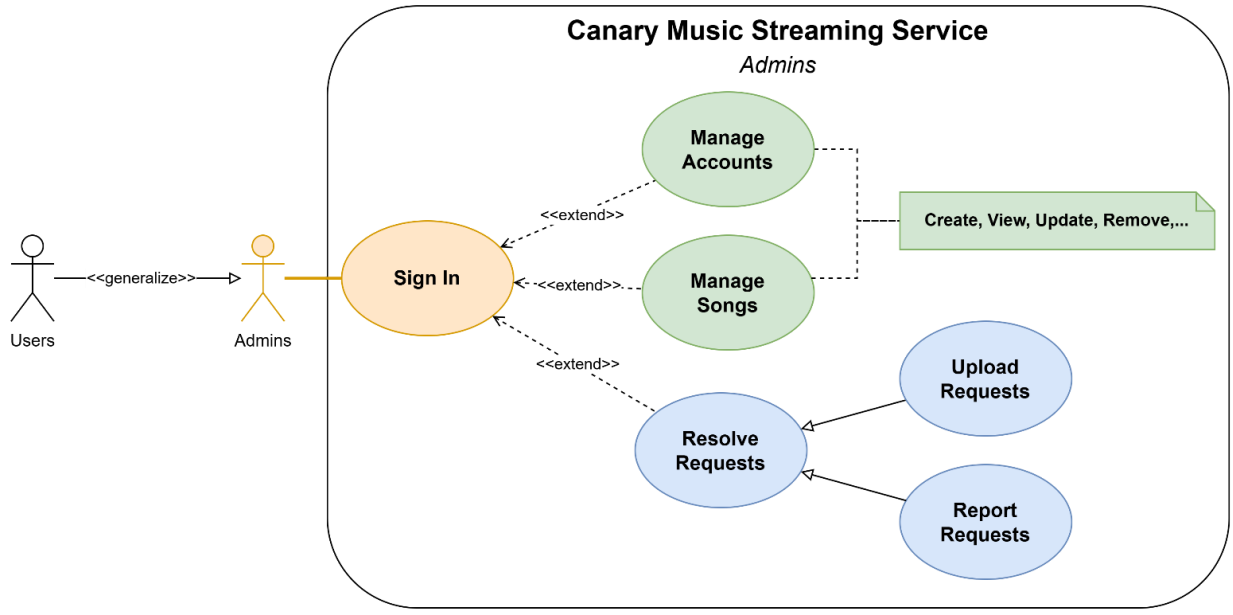
    Usernames are unique and are required by the system on registration.

3. Use-Case Model

3.1 Client Use-case model



3.2 Admin Use-case model

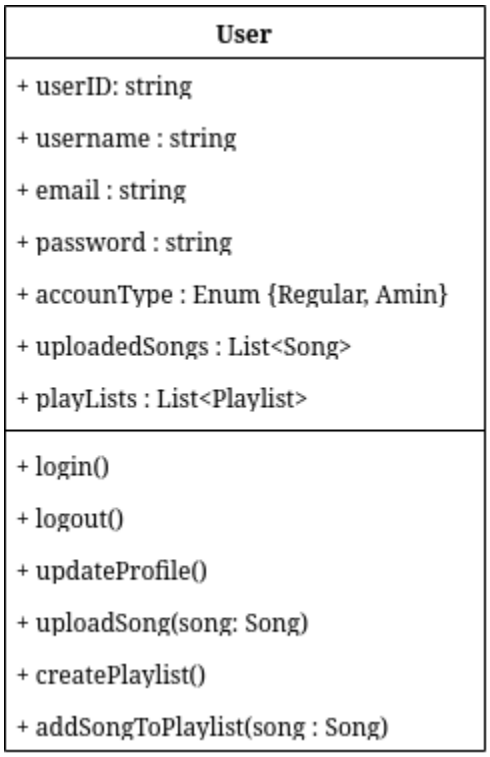


4. Logical View

Due to our project being built with Wix website builder, there is no applicable Architecture Design because everything has been handled internally by Wix. Therefore, we could only provide a general view to how our website and its features work as well as try to implement a design similar to Modular Architecture.

- Homepage:
  - Music recommendation
  - Trending tracks
- Music:
  - Control (play/pause/next/previous, shuffle, etc)
  - Artist info
  - Derived album
  - Similar songs
- UserProfile:
  - Account info
  - Update profile
- Library:
  - Created playlists
  - Uploaded songs
  - Most recently listened
- Search:
  - Searching for songs
  - Filter by artist, albums, etc.
- Admin dashboard
  - User management (ban, report tools)
  - Request moderation (upload requests, report requests)
  - Analytics (traffics, active users, upload requests)

4.1 Component: User



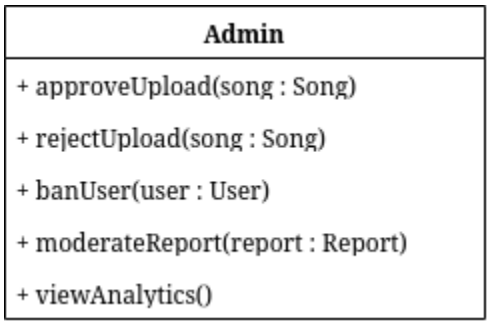
Attributes:

- **userID:** A unique identifier for each user, likely a string or a number.
- **username:** The user's chosen username, a string.
- **email:** The user's email address, a string.
- **password:** The user's password, stored securely (likely hashed or encrypted).
- **accountType:** An enum indicating whether the user is a regular user or an admin.
- **uploadedSongs:** A list of songs uploaded by the user, references to Song objects.
- **playLists:** A list of playlists created by the user, references to Playlist objects.

Methods:

- **login():** Authenticates the user based on their username/password combination.
- **logout():** Logs the user out of the system.
- **updateProfile():** Allows the user to modify their profile information (username, email, etc.).
- **uploadSong(song: Song):** Uploads a song to the user's account and adds it to their uploadedSongs list.
- **createPlaylist():** Creates a new playlist for the user and adds it to their playlists list.
- **addSongToPlaylist(song: Song):** Adds a song to an existing playlist in the user's playlists list.

4.2 Component: Admin (derived from User)



Methods:

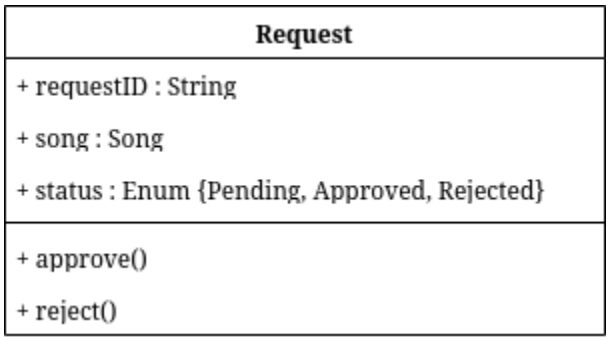
- **approveUpload(song: Song)**: Approves an uploaded song that is pending moderation.
- **rejectUpload(song: Song)**: Rejects an uploaded song that is pending moderation.
- **banUser(user: User)**: Bans a user from the platform, preventing them from accessing or uploading content.
- **moderateReport(report: Report)**: Investigates and takes appropriate action on a user-reported issue (e.g., copyright infringement, harassment).
- **viewAnalytics()**: Accesses and analyzes platform usage statistics and user behavior.

4.3 Component: Song

Attributes:

- **songID:** A unique identifier for the song, likely a string or a number.
- **source:** The source of the song (e.g., uploaded, from a library, etc.), a string.
- **title:** The title of the song, a string.
- **artist:** A reference to the User object who uploaded or created the song.
- **album:** A reference to the Playlist object that the song belongs to (if any).
- **genre:** The genre of the song, a string.
- **duration:** The duration of the song, likely stored as a time object or a number representing seconds.
- **coverImgPath:** The path to the cover image for the song, a string.

4.4 Component: Request



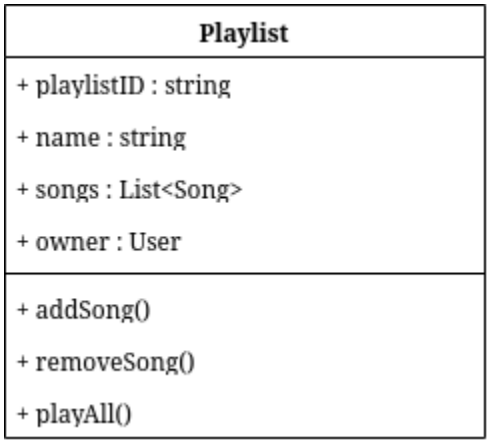
Attributes:

- **requestID:** A unique identifier for the request, likely a string or a number.
- **song:** A reference to the Song object that the request is associated with.
- **status:** An enum indicating the current status of the request (Pending, Approved, Rejected).

Methods:

- **approve():** Changes the request status to "Approved."
- **reject():** Changes the request status to "Rejected."

4.5 Component: Playlist



Attributes:

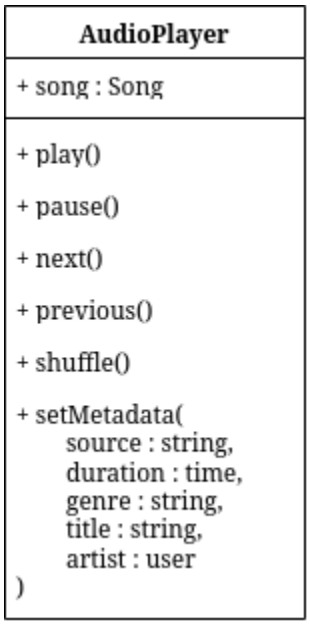
- **playlistID**: A unique identifier for the playlist, likely a string or a number.
- **name**: The name of the playlist, a string.
- **songs**: A list of songs that belong to the playlist, likely references to Song objects.
- **owner**: A reference to the User object who created the playlist.

Methods:

- **addSong()**: Adds a song to the playlist's list of songs.
- **removeSong()**: Removes a song from the playlist's list of songs.

- **playAll()**: Plays all songs in the playlist in sequence.

4.6 Component: AudioPlayer



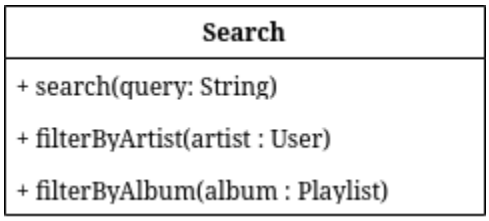
Attributes:

- **song**: A reference to the Song object that is currently being played.

Methods:

- **play()**: Starts playing the current song.
- **pause()**: Pauses the playback of the current song.
- **next()**: Plays the next song in the playlist (if available).
- **previous()**: Plays the previous song in the playlist (if available).
- **shuffle()**: Shuffles the order of songs in the playlist.
- **setMetadata(source: string, duration: time, genre: string, title: string, artist: user)**: Sets the metadata for the currently playing song, including its source, duration, genre, title, and artist.

4.7 Component: Search

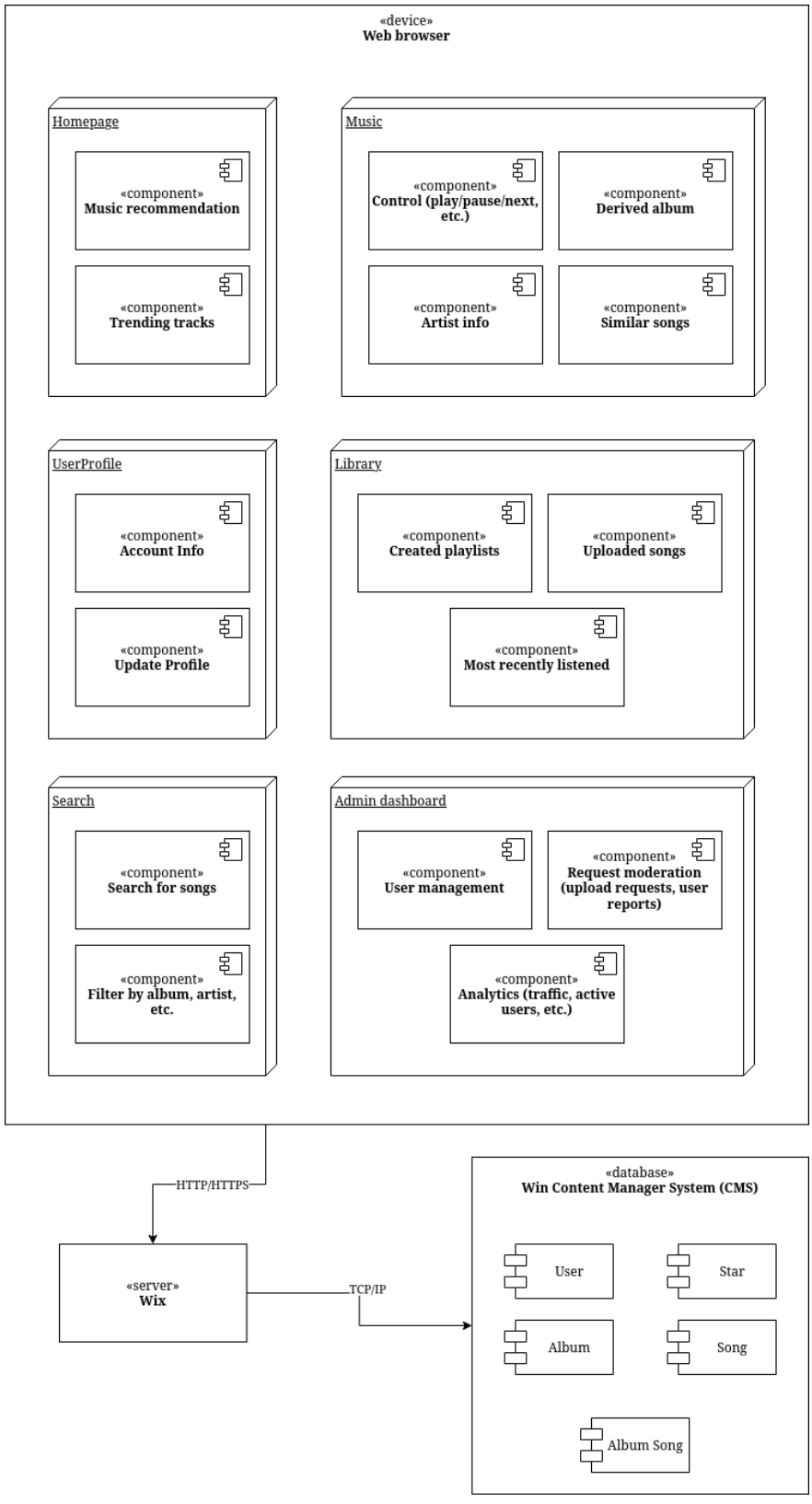


Methods:

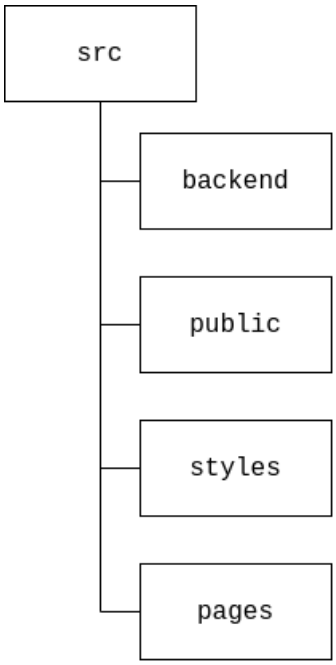
- **search(query: String)**: Searches for songs, playlists, or users based on the provided query string.

- **filterByArtist(artist: User)**: Filters search results to only include songs or playlists by the specified artist.
- **filterByAlbum(album: Playlist)**: Filters search results to only include songs from the specified album.

5. Deployment



6. Implementation View



6.1 Main folder: *backend/*

This folder implements the basic logic and platform of the application as well as defining behaviors for interaction with the database.

6.2 Main folder: *public/*

This folder contains code for handling globally visible components (header, footer, etc.)

6.3 Main folder: *styles/*

Is used for general styling of the application interface

6.4 Main folder: *pages/*

Contains layout for pages of the application and functions for each page’s interactive components.