Group01

Music Player
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
15/07/2022	1.0	Initial version	Trương Samuel Trần Hồng Minh Phúc Trần Thiện Tiến Phạm Ngọc Anh Thư

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

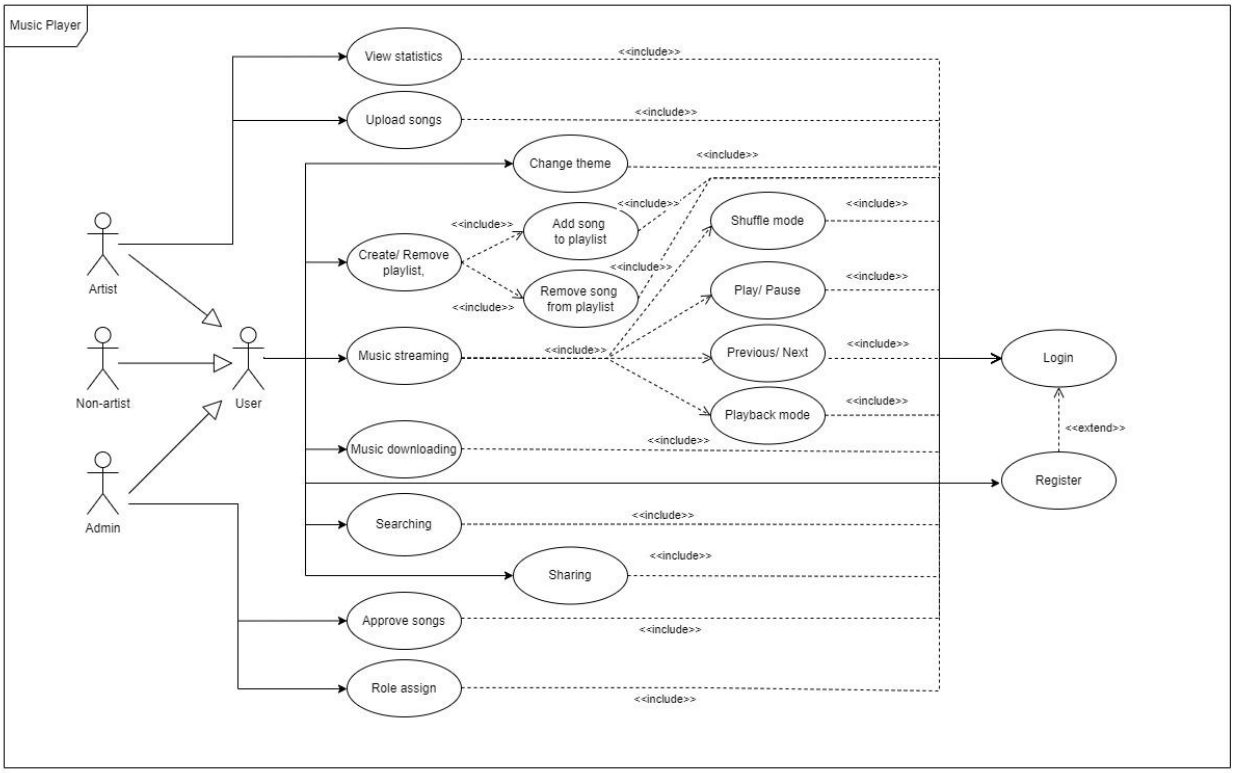
This Software Architecture Document applies to the Music Player System which will be developed by Group01.

2. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

- The delay when playing music should not be greater than 3 seconds
- The security of the account should have password salting.
- The maximum users access app at the same time should be 50.
- The application design should be easy and simple to use
- Language: Javascript (React Native), Go
- Environment: Android

3. Use-Case Model

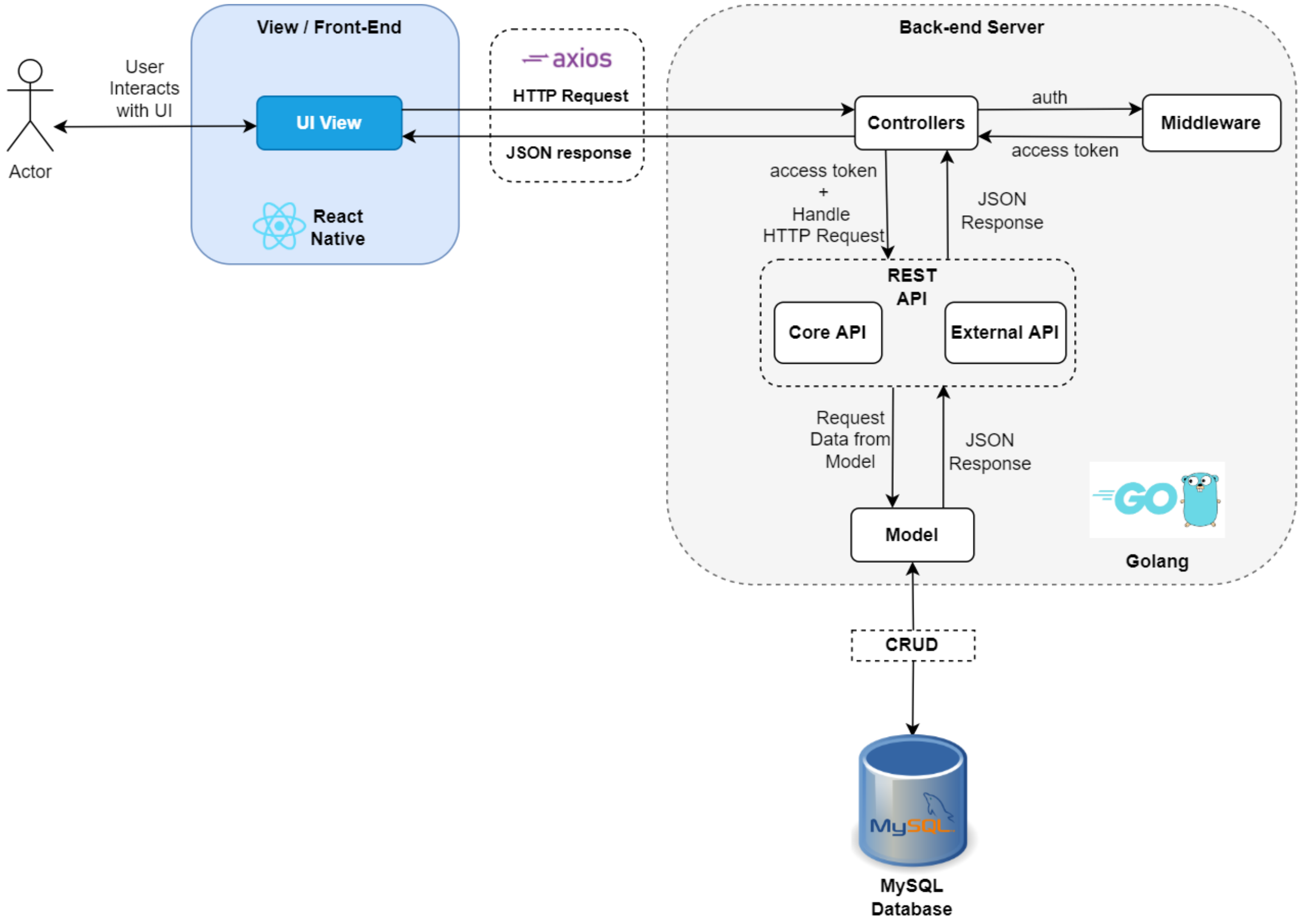


4. Logical View

A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers. Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

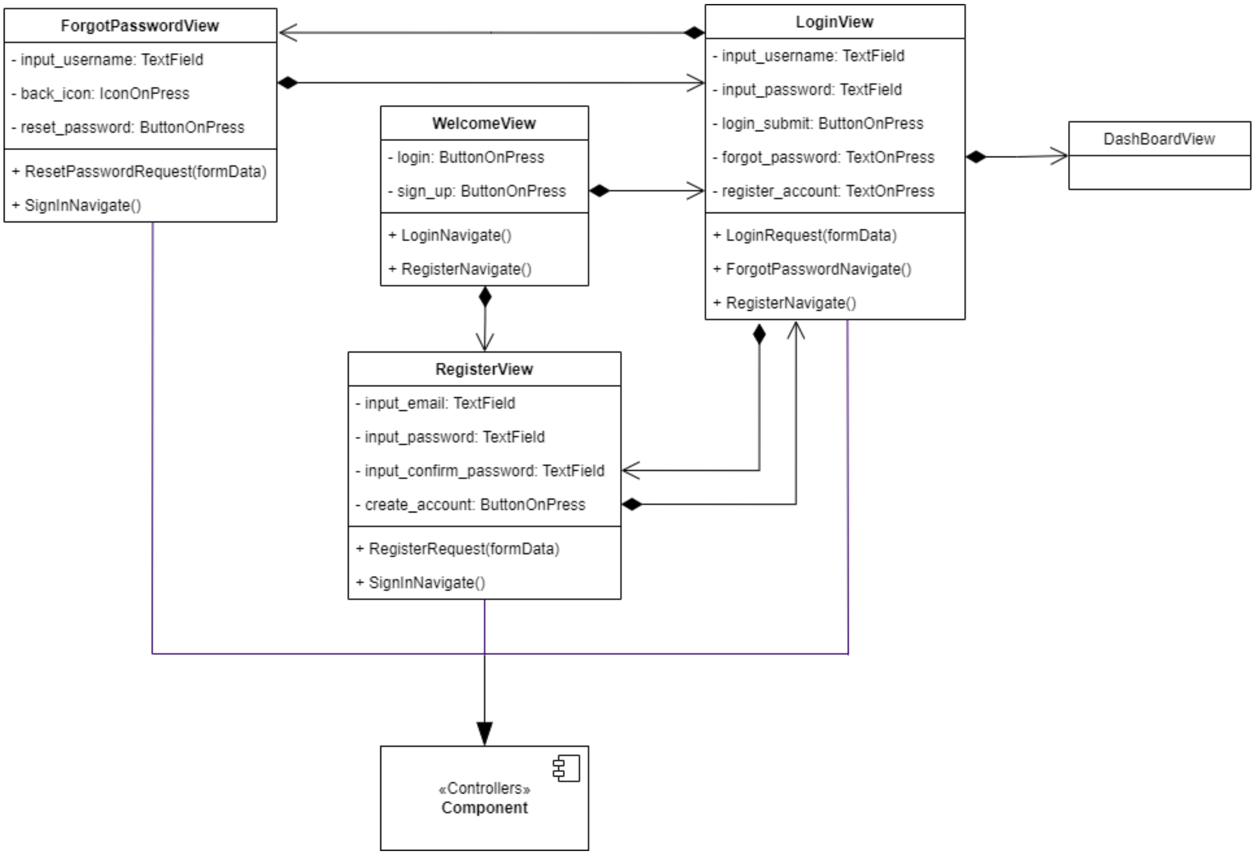
The logical view of the Music Player is implemented same as MVC and comprised of 4 main components:

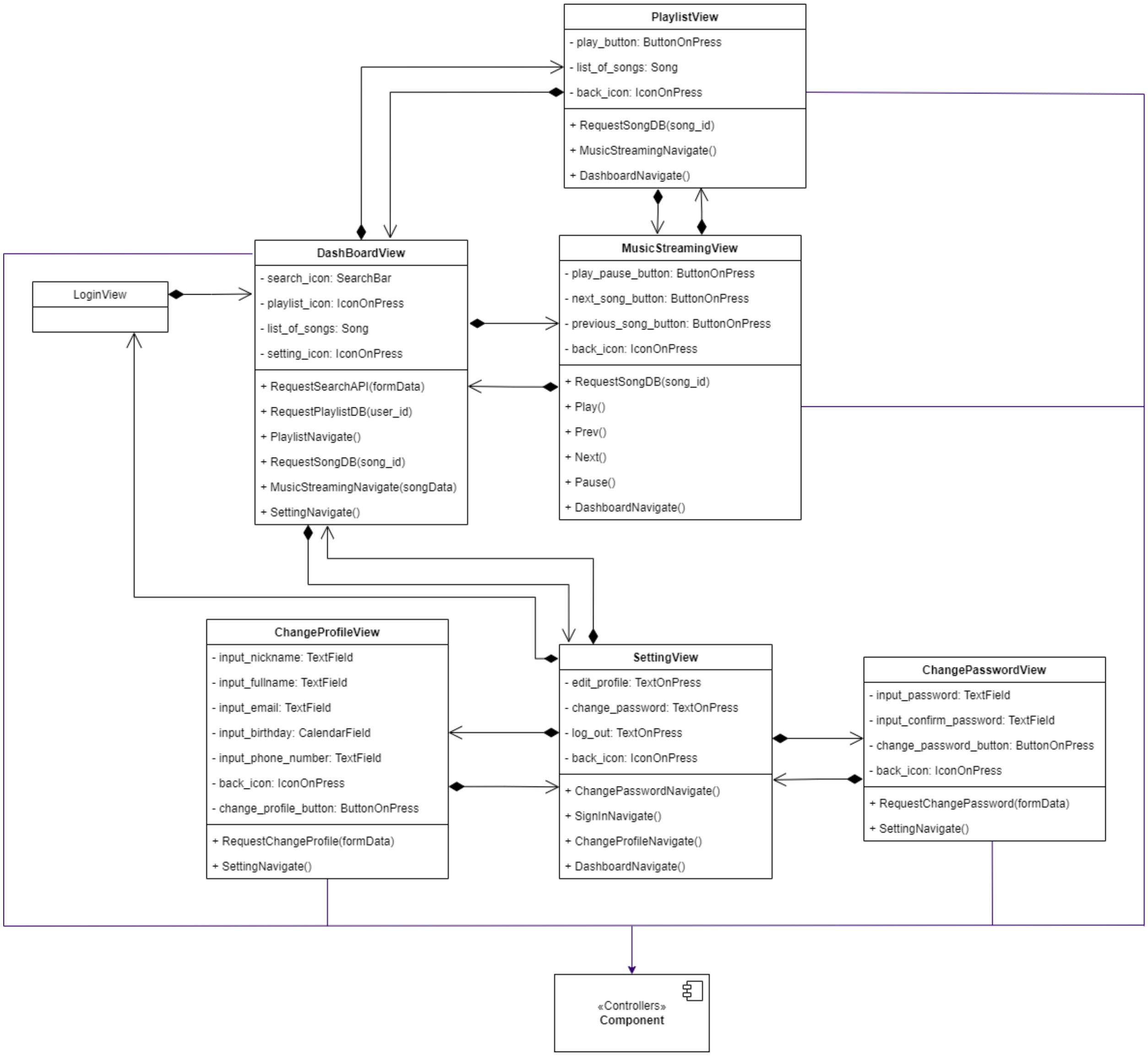
- **UI View:**
  - Responsibilities: Contains classes for each of the forms that the actors use to communicate with the system.
  - Service(s): Send HTTP Request to **Controllers** component when actors interact with UI.
- **Controllers:**
  - Responsibilities: handle application logic and request mapping.
  - Service(s): Handles HTTP Requests from User (**View** component) and forward to **Middleware** component for authentication. If the authentication is successful, continue to the **Model** component to request data through the built-in APIs (Core API) or external APIs. Otherwise, **Controllers** send an error response to the **View** component for display to the user and disallow access to the **Model** component
- **Middleware**
  - Responsibilities: Authenticate user and decide whether the user can access to the **Model** or not
  - Service(s): Receive HTTP Request from **Controller** and check if user is authenticated. If successful, **middleware** will give user access token to access **model.**
- **Model**
  - Responsibilities: Contains classes that represent all the data and data manipulation stored in MySQL database.
  - Service(s): Working with database through CRUD operations (create, read, update, delete)



4.1 Component: UI View

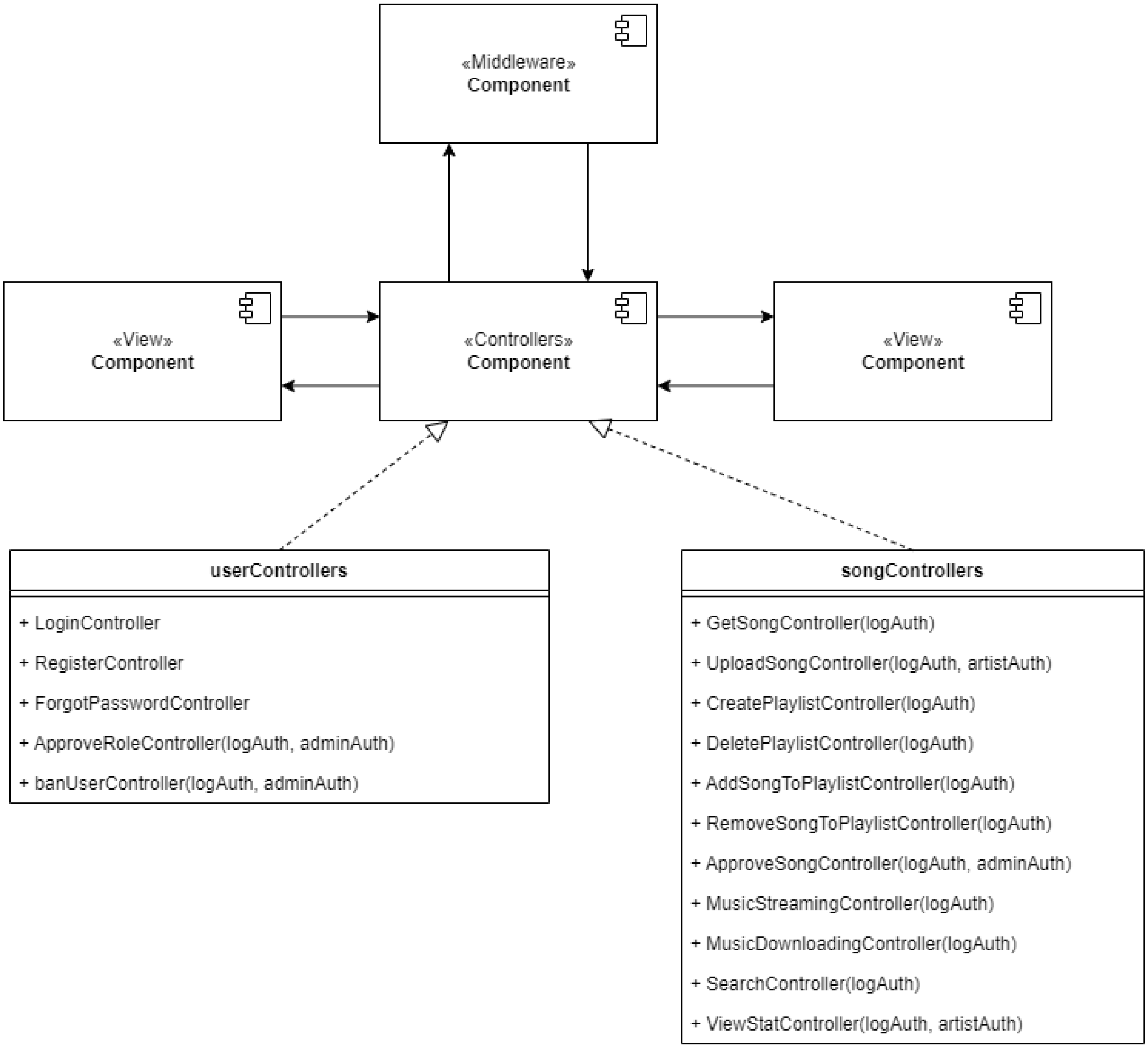
- Explain for all classes: Each class will represent buttons or text fields that users can interact.


ForgotPasswordView
- input_username: TextField
- back_icon: IconOnPress
- reset_password: ButtonOnPress
+ ResetPasswordRequest(formData)
+ SignInNavigate()
LoginView
- input_username: TextField
- input_password: TextField
- login_submit: ButtonOnPress
- forgot_password: TextOnPress
- register_account: TextOnPress
+ LoginRequest(formData)
+ ForgotPasswordNavigate()
+ RegisterNavigate()
DashBoardView
WelcomeView
- login: ButtonOnPress
- sign_up: ButtonOnPress
+ LoginNavigate()
+ RegisterNavigate()
RegisterView
- input_email: TextField
- input_password: TextField
- input_confirm_password: TextField
- create_account: ButtonOnPress
+ RegisterRequest(formData)
+ SignInNavigate()
«Controllers»
Component


PlaylistView
- play_button: ButtonOnPress
- list_of_songs: Song
- back_icon: IconOnPress
+ RequestSongDB(song_id)
+ MusicStreamingNavigate()
+ DashboardNavigate()
DashBoardView
- search_icon: SearchBar
- playlist_icon: IconOnPress
- list_of_songs: Song
- setting_icon: IconOnPress
+ RequestSearchAPI(formData)
+ RequestPlaylistDB(user_id)
+ PlaylistNavigate()
+ RequestSongDB(song_id)
+ MusicStreamingNavigate(songData)
+ SettingNavigate()
MusicStreamingView
- play_pause_button: ButtonOnPress
- next_song_button: ButtonOnPress
- previous_song_button: ButtonOnPress
- back_icon: IconOnPress
+ RequestSongDB(song_id)
+ Play()
+ Prev()
+ Next()
+ Pause()
+ DashboardNavigate()
LoginView
ChangeProfileView
- input_nickname: TextField
- input_fullname: TextField
- input_email: TextField
- input_birthday: CalendarField
- input_phone_number: TextField
- back_icon: IconOnPress
- change_profile_button: ButtonOnPress
+ RequestChangeProfile(formData)
+ SettingNavigate()
SettingView
- edit_profile: TextOnPress
- change_password: TextOnPress
- log_out: TextOnPress
- back_icon: IconOnPress
+ ChangePasswordNavigate()
+ SignInNavigate()
+ ChangeProfileNavigate()
+ DashboardNavigate()
ChangePasswordView
- input_password: TextField
- input_confirm_password: TextField
- change_password_button: ButtonOnPress
- back_icon: IconOnPress
+ RequestChangePassword(formData)
+ SettingNavigate()
«Controllers»
Component

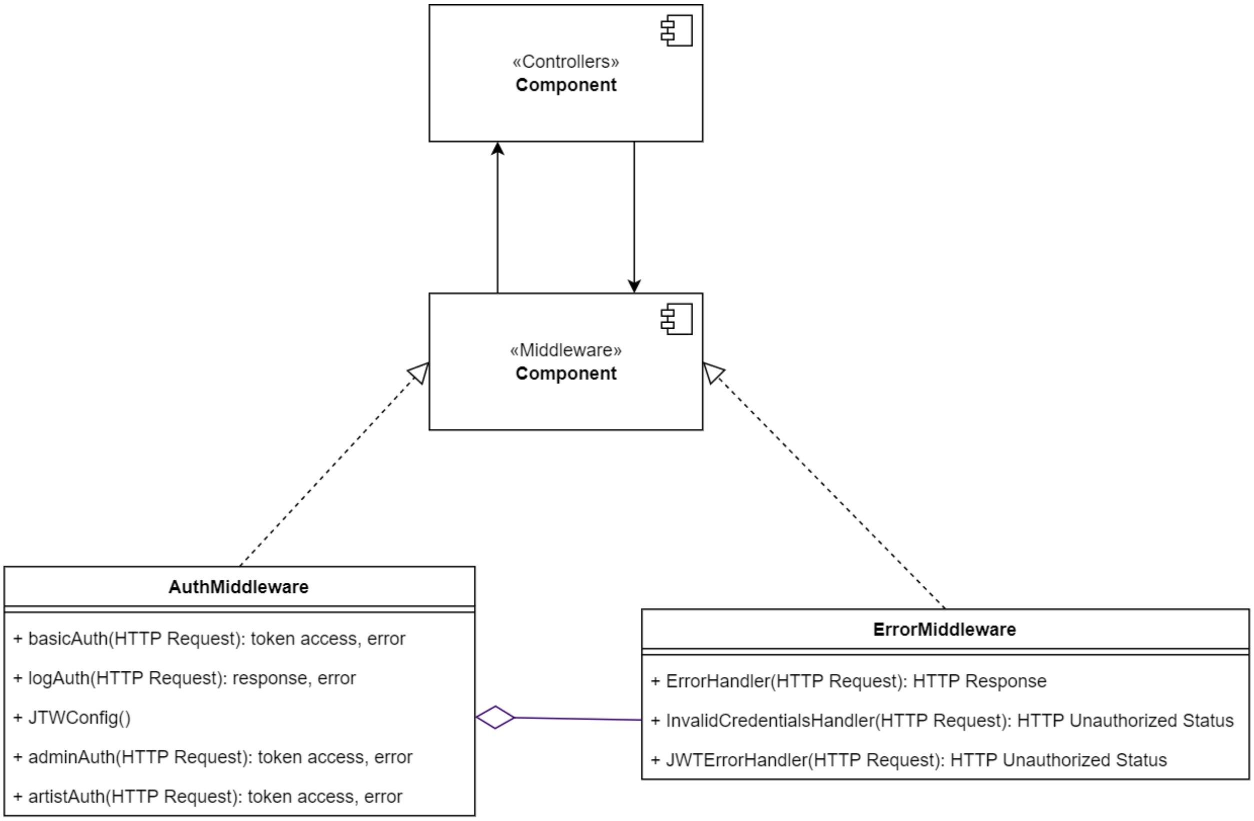
4.2 Component: Controllers

- userControllers: Handle user-related interactions on the system (such as login, register, change password,...)
- songControllers: Handle user and song related interactions (stream music, create playlist, add song to playlist,...)


«Middleware»
Component
«View»
Component
«Controllers»
Component
«View»
Component
userControllers
+ LoginController
+ RegisterController
+ ForgotPasswordController
+ ApproveRoleController(logAuth, adminAuth)
+ banUserController(logAuth, adminAuth)
songControllers
+ GetSongController(logAuth)
+ UploadSongController(logAuth, artistAuth)
+ CreatePlaylistController(logAuth)
+ DeletePlaylistController(logAuth)
+ AddSongToPlaylistController(logAuth)
+ RemoveSongToPlaylistController(logAuth)
+ ApproveSongController(logAuth, adminAuth)
+ MusicStreamingController(logAuth)
+ MusicDownloadingController(logAuth)
+ SearchController(logAuth)
+ ViewStatController(logAuth, artistAuth)

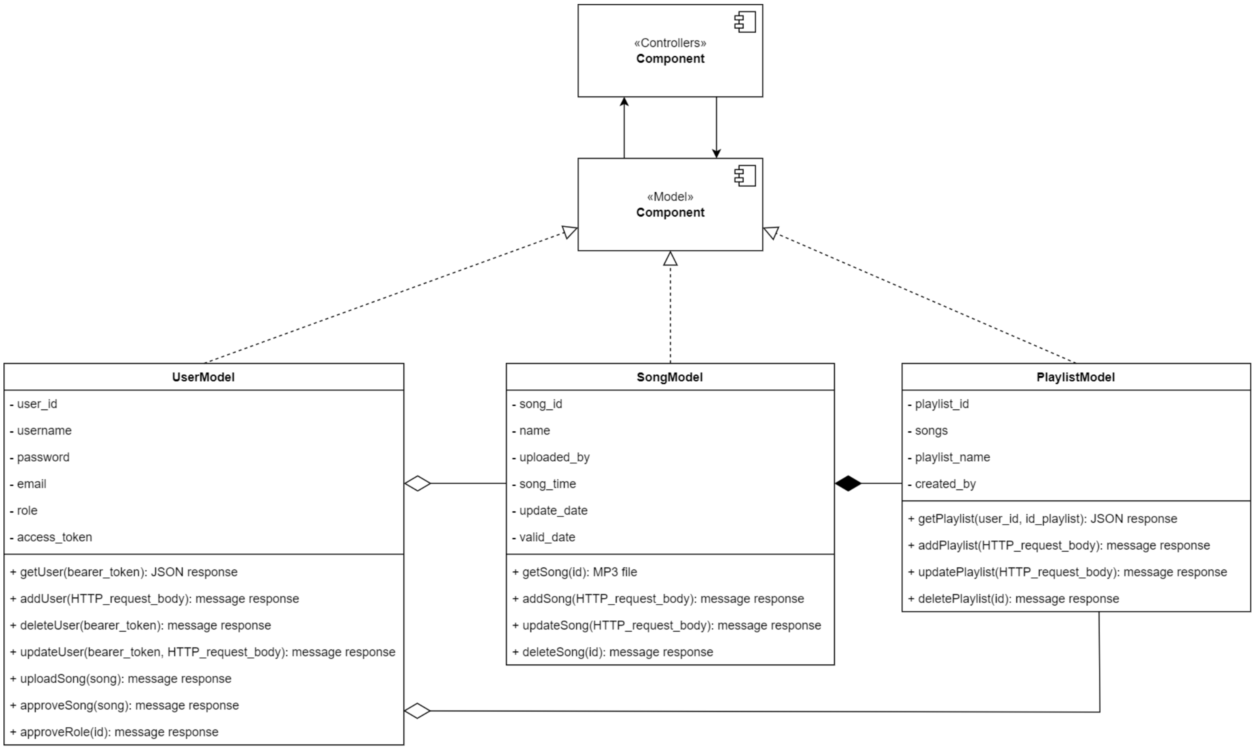
4.3 Component: Middleware

- AuthMiddleware: Handle user authentication (for example: check if the user is logged in before or if the user is admin/artist,...)
- ErrorMiddleware: Handle common errors and notify users of solutions (such as enter the wrong password, enter the wrong password more than 3 times, the user requested the wrong format,...)


«Controllers»
Component
«Middleware»
Component
AuthMiddleware
+ basicAuth(HTTP Request): token access, error
+ logAuth(HTTP Request): response, error
+ JTWConfig()
+ adminAuth(HTTP Request): token access, error
+ artistAuth(HTTP Request): token access, error
ErrorMiddleware
+ ErrorHandler(HTTP Request): HTTP Response
+ InvalidCredentialsHandler(HTTP Request): HTTP Unauthorized Status
+ JWTErrorHandler(HTTP Request): HTTP Unauthorized Status

4.4 Component: Model

- UserModel: data and data interactions on the database related to User (login - read, register - create, change password - update,...)
- SongModel: data and data interactions on the database related to Song (play music - read, delete song from playlist - delete, update name/title of song - update,...)
- PlaylistModel: data and data interactions on the database related to Playlist (create playlist - create, choose any playlist - read,...)



«Controllers»
Component
«Model»
Component
UserModel
- user_id
- username
- password
- email
- role
- access_token
+ getUser(bearer_token): JSON response
+ addUser(HTTP_request_body): message response
+ deleteUser(bearer_token): message response
+ updateUser(bearer_token, HTTP_request_body): message response
+ uploadSong(song): message response
+ approveSong(song): message response
+ approveRole(id): message response
SongModel
- song_id
- name
- uploaded_by
- song_time
- update_date
- valid_date
+ getSong(id): MP3 file
+ addSong(HTTP_request_body): message response
+ updateSong(HTTP_request_body): message response
+ deleteSong(id): message response
PlaylistModel
- playlist_id
- songs
- playlist_name
- created_by
+ getPlaylist(user_id, id_playlist): JSON response
+ addPlaylist(HTTP_request_body): message response
+ updatePlaylist(HTTP_request_body): message response
+ deletePlaylist(id): message response

5. Deployment

6. Implementation View