LISTEN TOGETHER

SRS Document

Version	Written By	Reviewed By	Approved By	Date
0.X	Lamees Emad Matthew Emile	Matthew Emile Lamees Emad	Lamees Emad Matthew Emile	SEPTEMBER 12, 2018

SEPTEMBER 12, 2018

Introduced by Group 30

Table of Contents

Introduction

Executive Summary

This Software Requirements Specification (SRS) describes the nature of our mobile application in technical term as well as provides an overview of the entire purposes, abbreviations, and references of the application. The aim of this document is to gather and analyze and give an in-depth insight of the complete **Listen Together Mobile Application** by defining its functionality and design in detail. It also concentrates on the user interactions with the system. The detailed requirements of the **Listen Together Mobile Application** are provided in this document.

Document Overview

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

The remainder of this document includes six chapters which are: System Description, System Modules, System Functions, System Models, Non-functional Requirements and System Interfaces.

The first chapter, the System Description provides a quick summary about the basic system modules and what they do. The second chapter, the System Modules provides a clear and detailed description of the system modules' functionality assisted with the use of illustrations and diagrams such as Context Diagrams, Activity Diagrams, ER Diagrams and State Machine Diagrams. The third chapter, the System Functions shows a detailed list of every existent system function and show to which module each belongs. The fourth chapter, the System Models, shows models illustrating how the system is used by the user or what the system is further made up of. These models include Use Case Diagrams and Class Diagrams. The fifth module shows the Non-functional Requirements of the system. The sixth and last chapter which is the System Interfaces shows instances of the User Interfaces of our mobile application.

Definitions, Acronyms, and Abbreviations

Term	Definition
SRS	Software Requirements Specification
ER	Entity Relationship
HTTP	Hypertext Transfer Protocol
App	Application
STR	Storage
MRH	Main Request Handler
SCI	Social Integration
PLL	Playlist
RMS	Room Server
RSS	Room Server Spawner
MBE	Music Backend
NFR	Non-functional Requirement
FR	Functional Requirement

References

[1] https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database

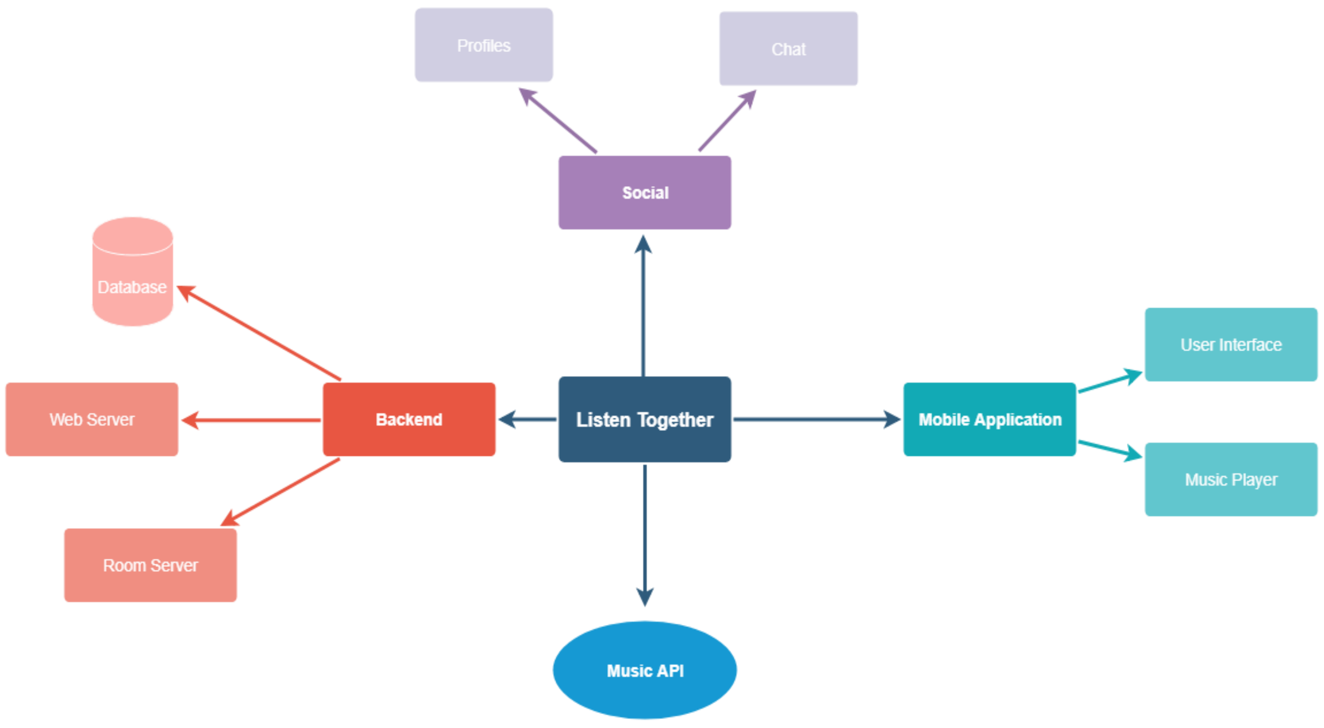
[2] https://sachinsdate.wordpress.com/2013/04/27/non-functional-requirements-in-mobile-applications/

[3] https://en.wikipedia.org/wiki/Software_requirements_specification

System Description

Introduction

The Listen Together System consists of a number of subsystems interacting together and with an external system. Our subsystems can be listed as: The ‘Social’ subsystem, the ‘Backend’ subsystem, and the ‘Mobile Application’ subsystem. The external system interacting with our system is the ‘Music API’. The ‘Social’ subsystem consists of two smaller subsystems which are the ‘User Profile’ and the ‘Chat’. The ‘Backend’ subsystem consists of three smaller systems which are the ‘Database’, the ‘Web Server’ and the ‘Room Server’. Finally, the ‘Mobile Application’ subsystem consists of two smaller subsystems which are the ‘User Interface’, and the ‘Music Player’.



Users

Our system has only of kind of user; the user which interacts with our mobile application, or the client.

Modules

1. Main Request Handler Module

2. Storage

- Store user authorization information
- Store user profile details
- Store playlists

3. Social Integration Module
   This module is in control of handling all user and user to user interactions such as:

- Authorizing old users (Sign in handler)
- Authorizing new users (Sign up handler)
  Linking friends to user accounts
  Retrieve friends online
- Retrieve invitations
- Retrieve application users

4. Playlist Module
   This module is in control of anything playlist related such as:
   - Defines what a playlist is
   - Creating playlists
   - Retrieving playlists
   - Editing playlist

5. Room Server Module
   This module is in control of handling music syncing within a single room; it can do the following:
   - Allow synced playing
   - Allow synced pausing and resuming
   - Allow synced skipping
   - Transmit track choices to all users in a room
   - Handles chat within a room
   - Handles the music backend calls

6. Room Server Spawner Module
   This module is in control of spawning new room servers upon request from a user; it can do the following:
   - Receives and handles room creation requests
   - Launches room servers with correct parameters
   - Receives and handles room join requests

7. Music Backend Module
   This module is in control of handling all music requests; it can do the following:
   - Query the available song choices by name
   - Respond to query by top songs that match
   - Respond with a song by name on first occurrence
   - Respond with a song by link if match of the link searched for by user found

8. Mobile Application Module
   This module serves the end user; it does the following:
     - Provide a logging in screen
     - Provide a sign up screen
     - Provide room screen

- Provide music player screen
- Provide a chatting interface
- Provide notification bar music handling
- Provide playlist interface
- Provide user profile interface
- Provide ability to join a room

System Modules

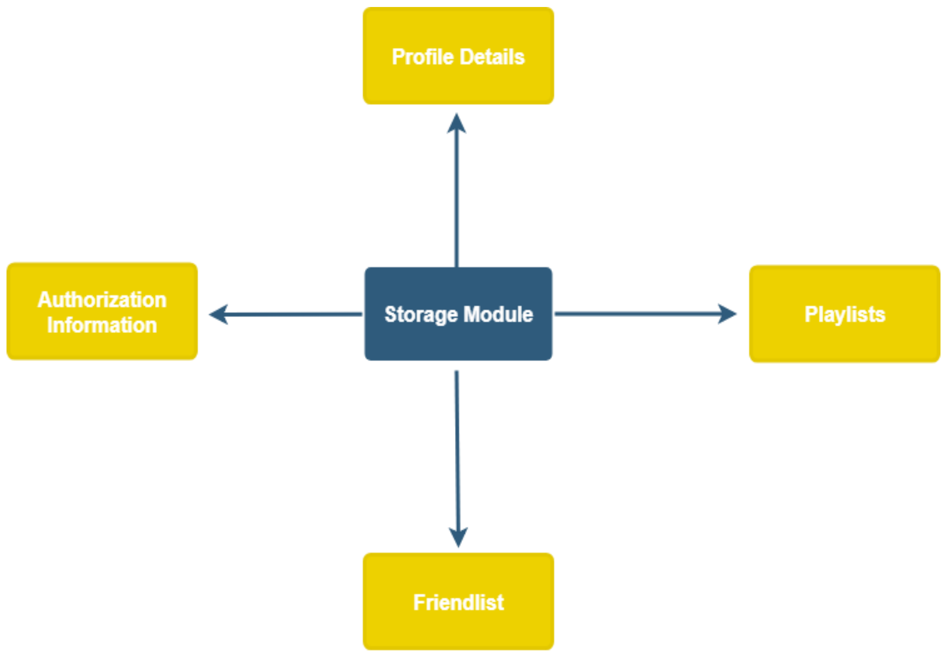
1. Main Request Handler Module

This module handles all kinds of http requests by receiving them from the Mobile App Module, passing each of them to a suitable system module then receiving a response from that module. It then sends the proper response back to the front-end.

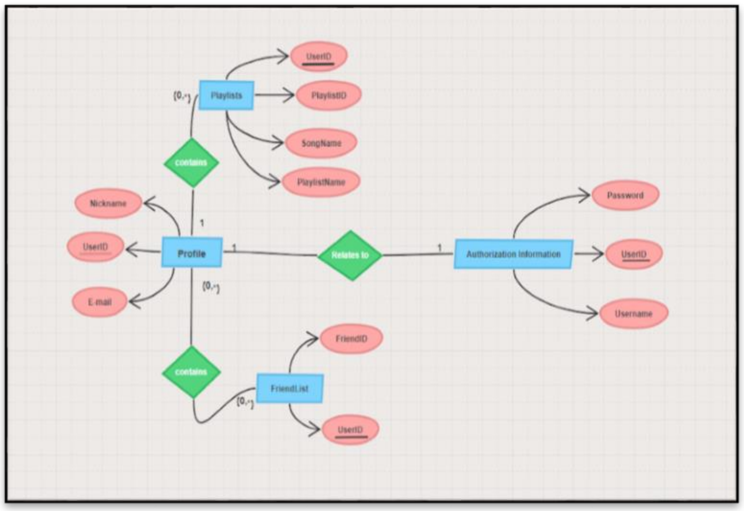
2. Storage Module

This module handles all types of storages in listen together system. It is basically a module describing our database and the kinds of data that will be stored in it.

- Storage Module: Context Diagram



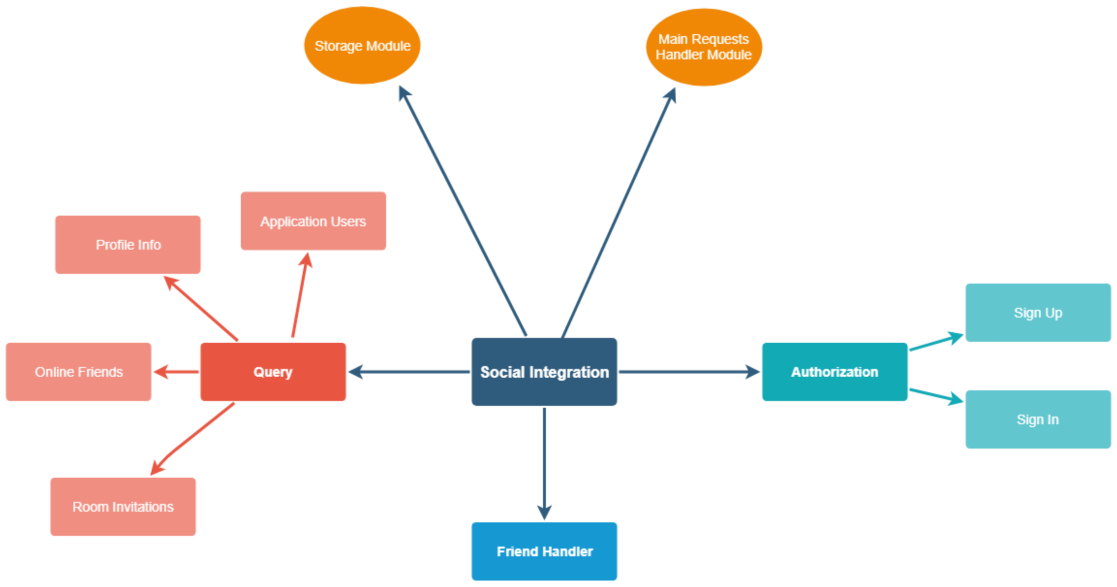
- **Storage Module: ER Diagram**



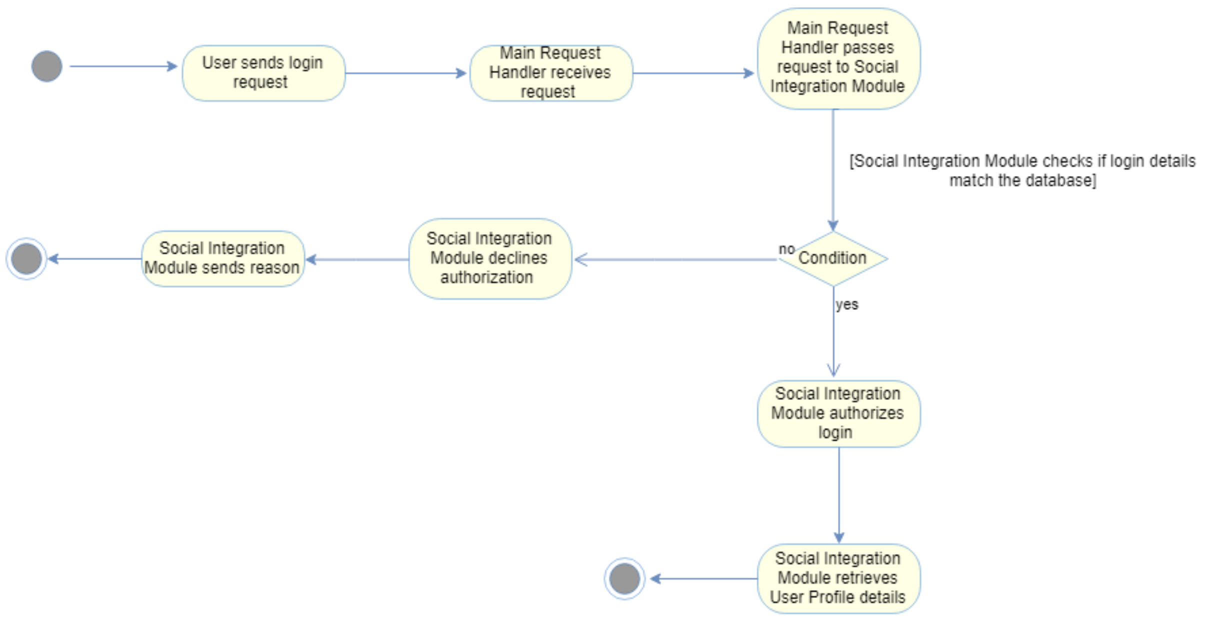
3. Social Integration Module

This module is in control of handling all user and user to user interactions such as signing users up, signing users in, linking friends to accounts, retrieving friends online, retrieving room invitations and retrieving application users.

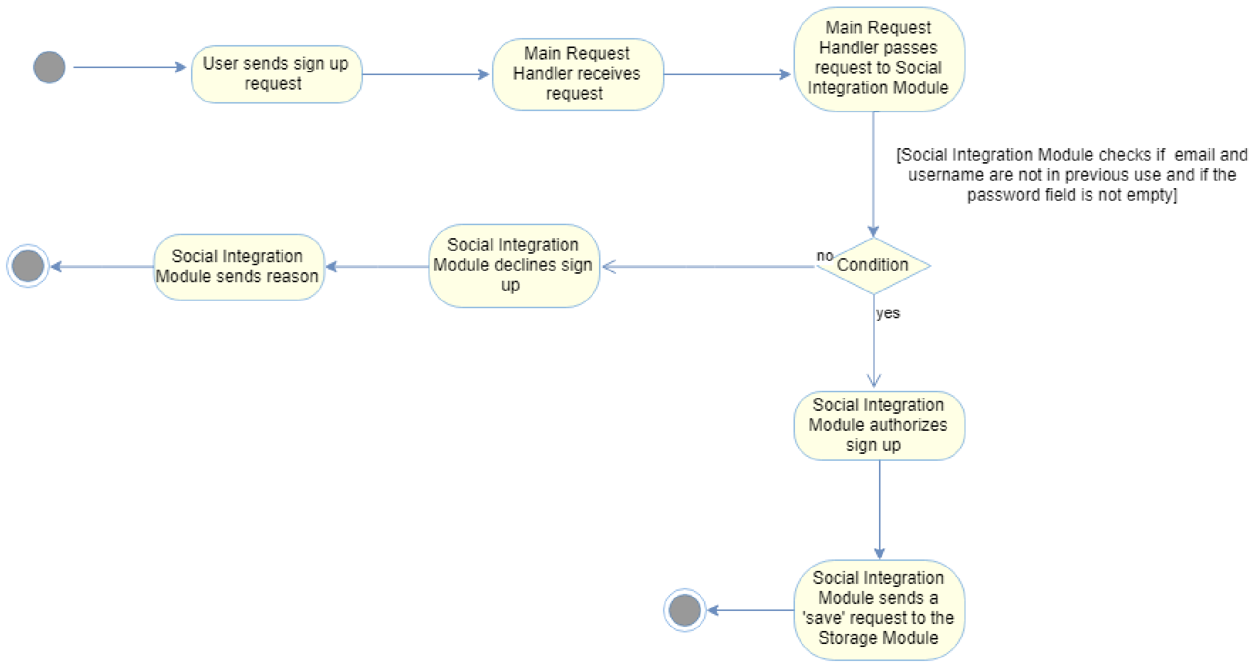
- **Social Integration Module: Context Diagram**



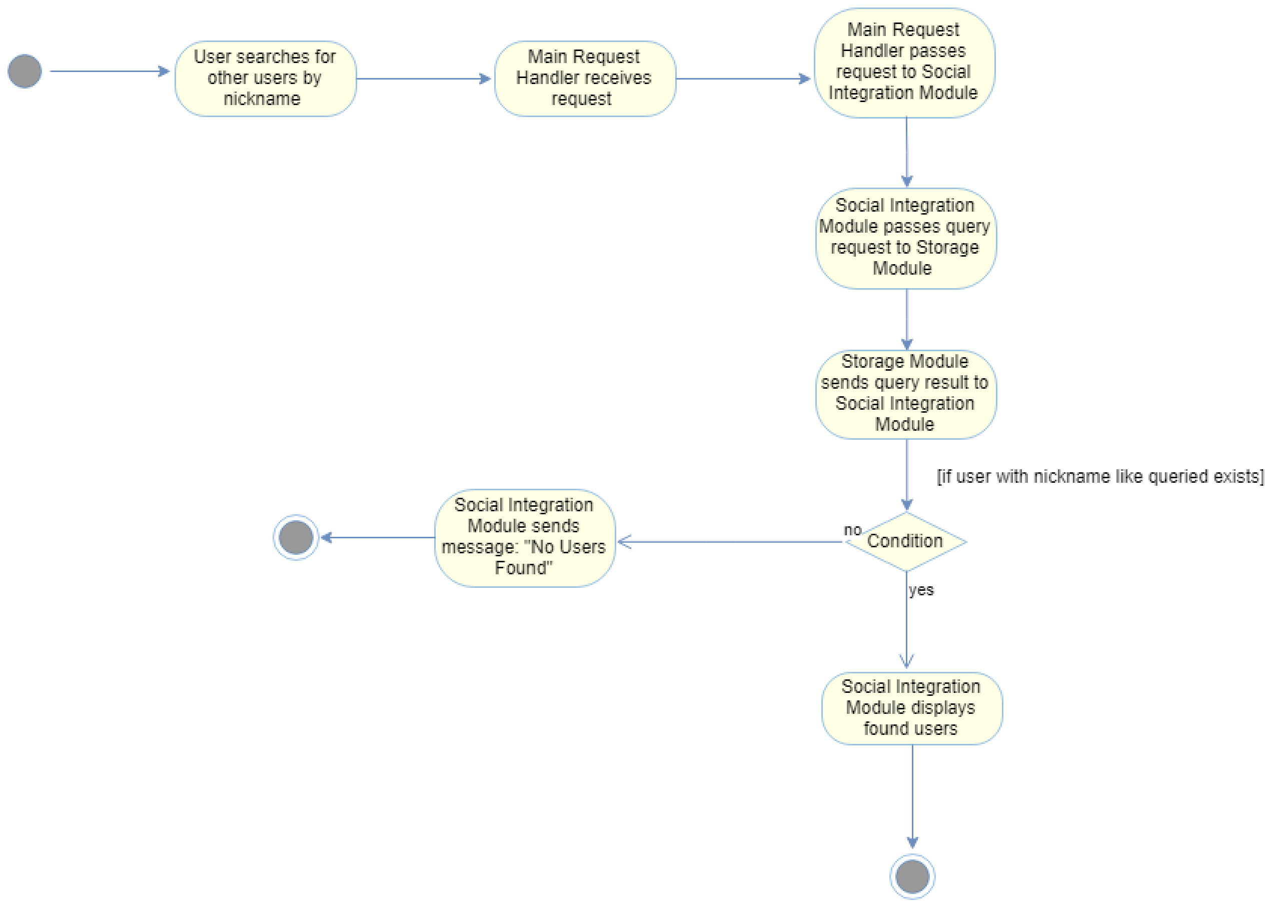
- ## Social Integration Module: User Login Activity Diagram


User sends login request
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
[Social Integration Module checks if login details match the database]
no
Condition
yes
Social Integration Module declines authorization
Social Integration Module sends reason
Social Integration Module authorizes login
Social Integration Module retrieves User Profile details

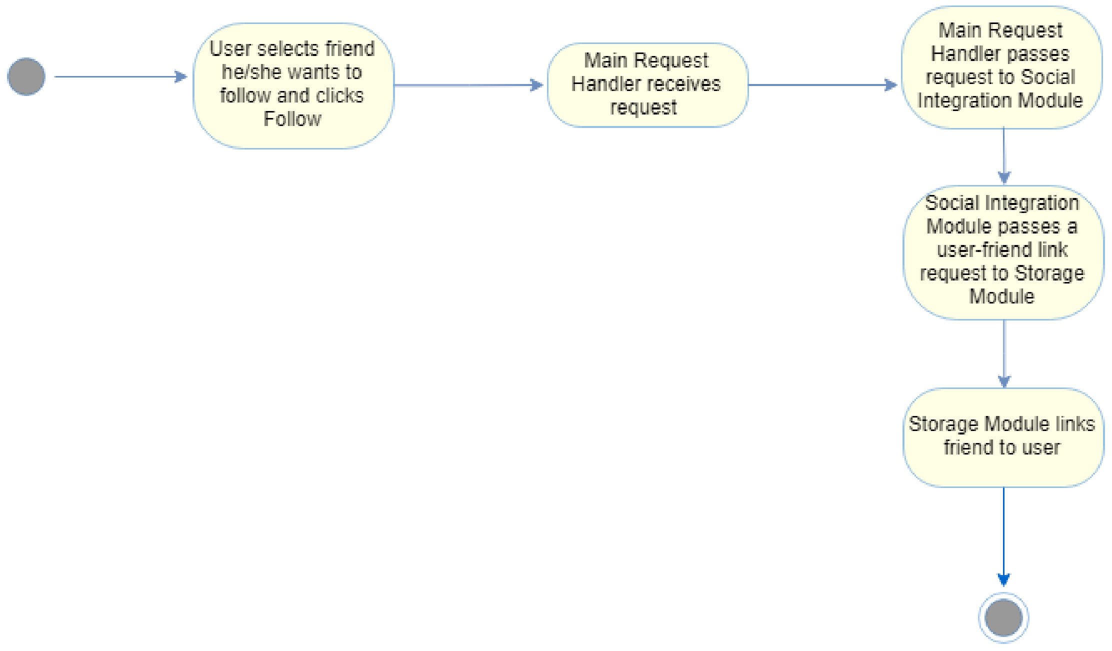
- ## Social Integration Module: User Sign Up Activity Diagram


User sends sign up request
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
[Social Integration Module checks if email and username are not in previous use and if the password field is not empty]
no
Condition
yes
Social Integration Module declines sign up
Social Integration Module sends reason
Social Integration Module authorizes sign up
Social Integration Module sends a 'save' request to the Storage Module

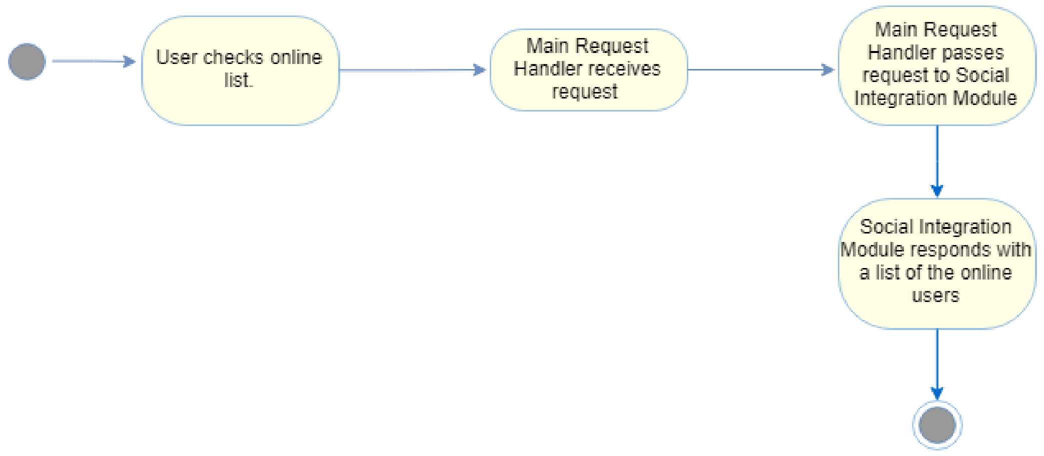
- **Social Integration Module: Search User Activity Diagram**


User searches for other users by nickname
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
Social Integration Module passes query request to Storage Module
Storage Module sends query result to Social Integration Module
[if user with nickname like queried exists]
Condition
no
yes
Social Integration Module sends message: "No Users Found"
Social Integration Module displays found users

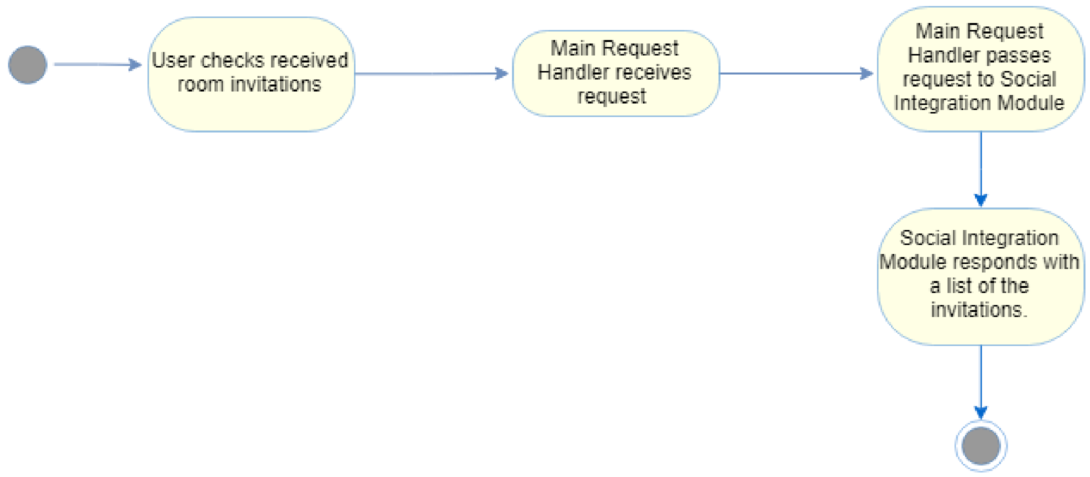
• Social Integration Module: Follow Friend Activity Diagram


User selects friend he/she wants to follow and clicks Follow
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
Social Integration Module passes a user-friend link request to Storage Module
Storage Module links friend to user

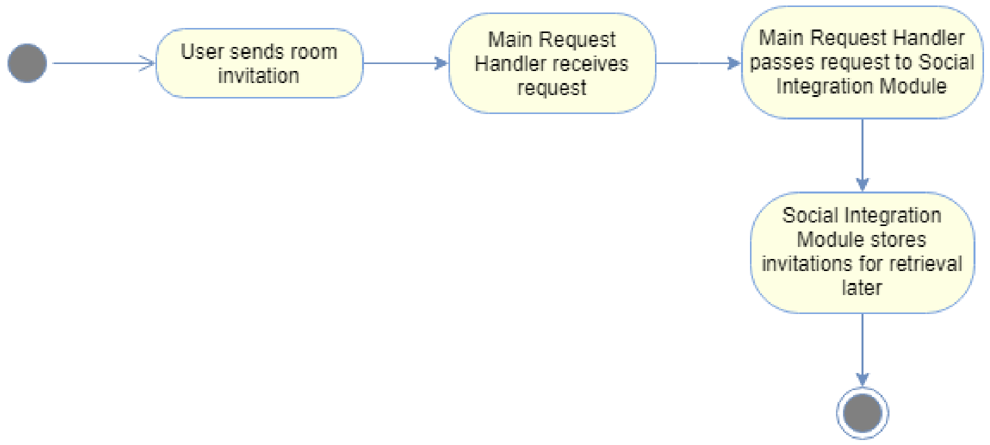
• Social Integration Module: Online Friends Activity Diagram


User checks online list.
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
Social Integration Module responds with a list of the online users

- **Social Integration Module: Retrieve Invitations Activity Diagram**


User checks received room invitations
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
Social Integration Module responds with a list of the invitations.

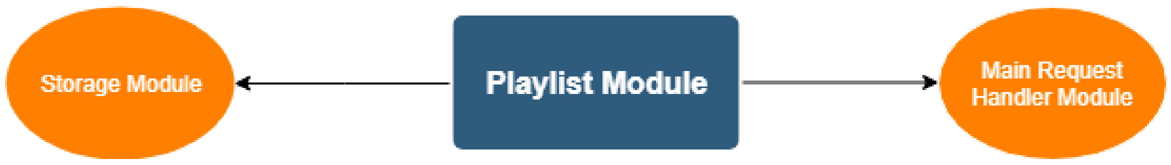
- **Social Integration Module: Send Invitations Activity Diagram**


User sends room invitation
Main Request Handler receives request
Main Request Handler passes request to Social Integration Module
Social Integration Module stores invitations for retrieval later

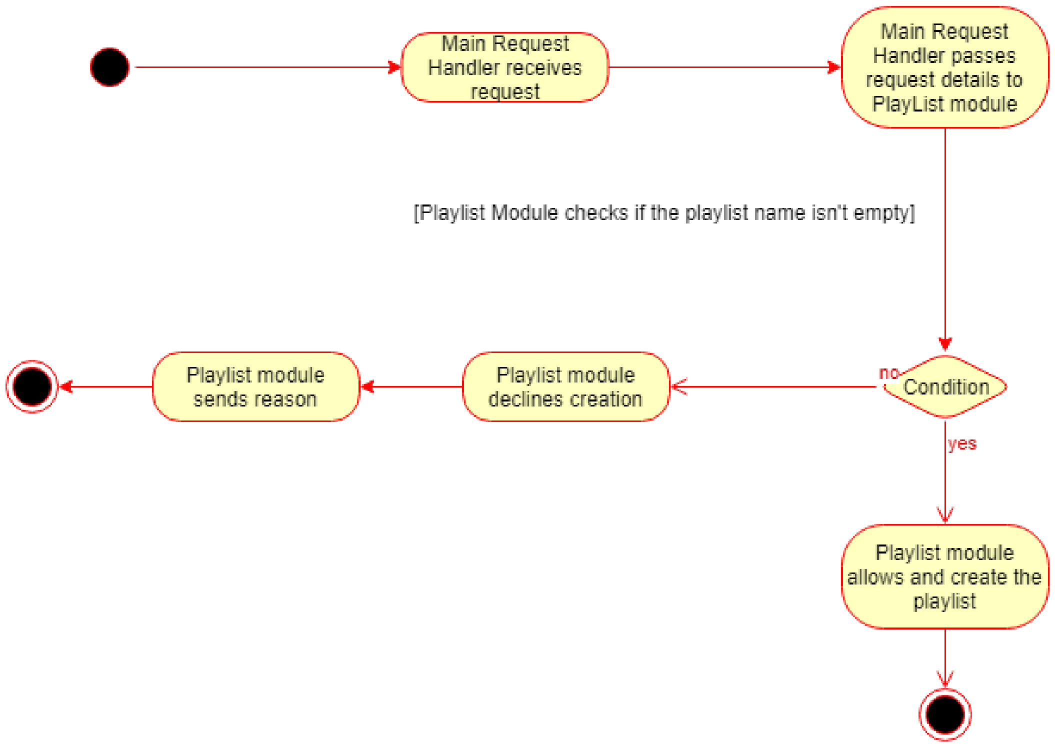
4. Playlist Module

This module is in control of anything playlist related such as defining, creating, editing and retrieving playlists.

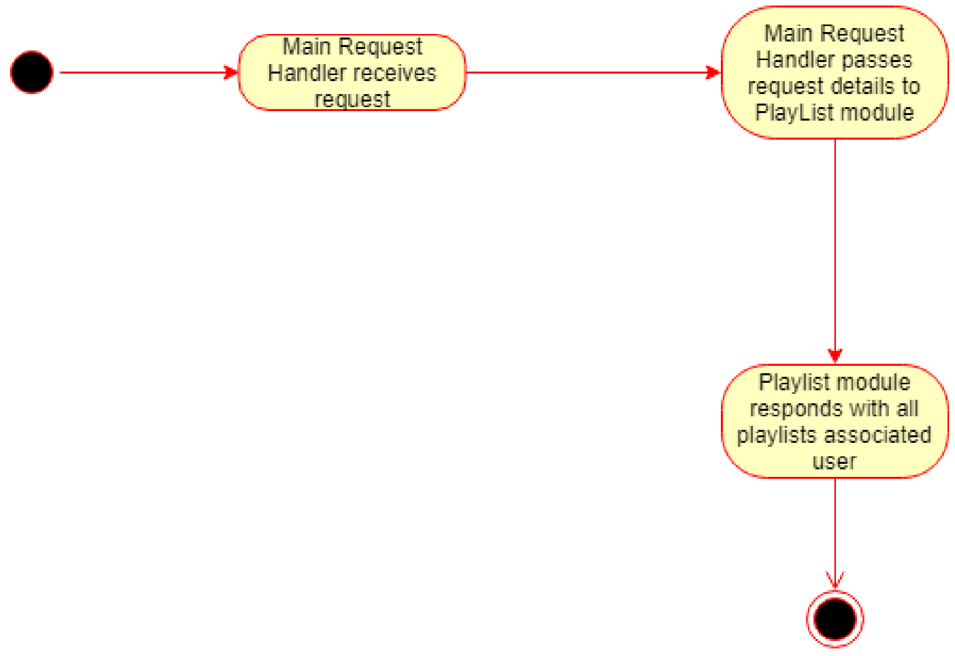
- **Playlist Module: Context Diagram**



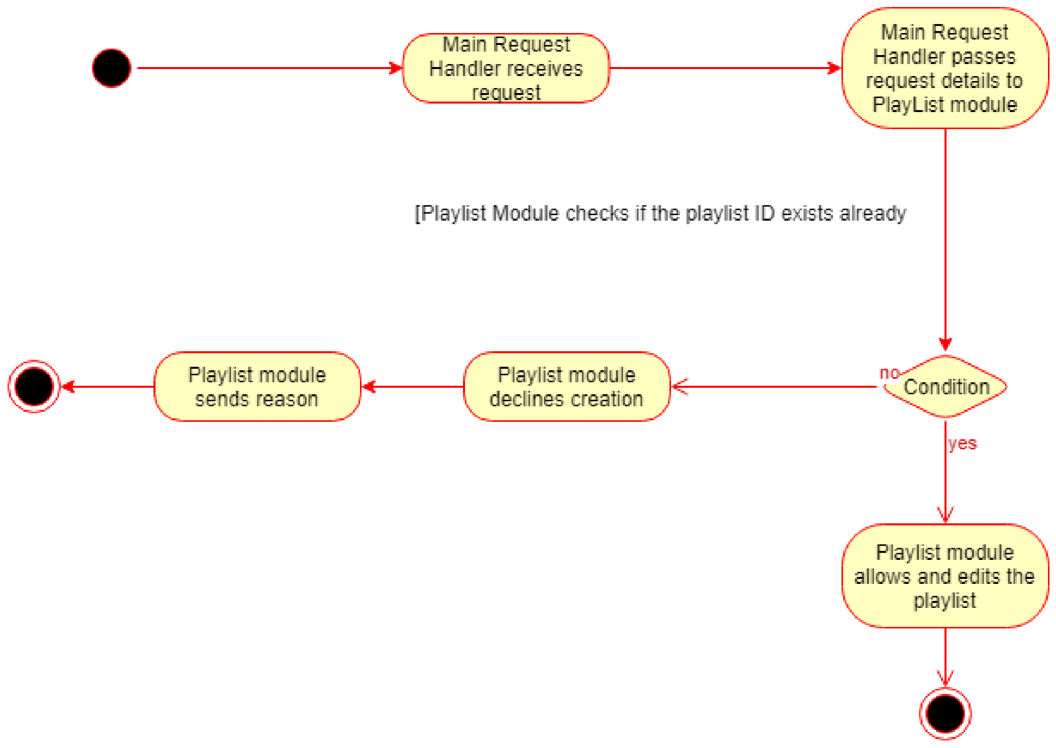
- **Playlist Module: Playlist Creation Activity Diagram**


Main Request Handler receives request
Main Request Handler passes request details to PlayList module
[Playlist Module checks if the playlist name isn't empty]
Condition
no
yes
Playlist module declines creation
Playlist module sends reason
Playlist module allows and create the playlist

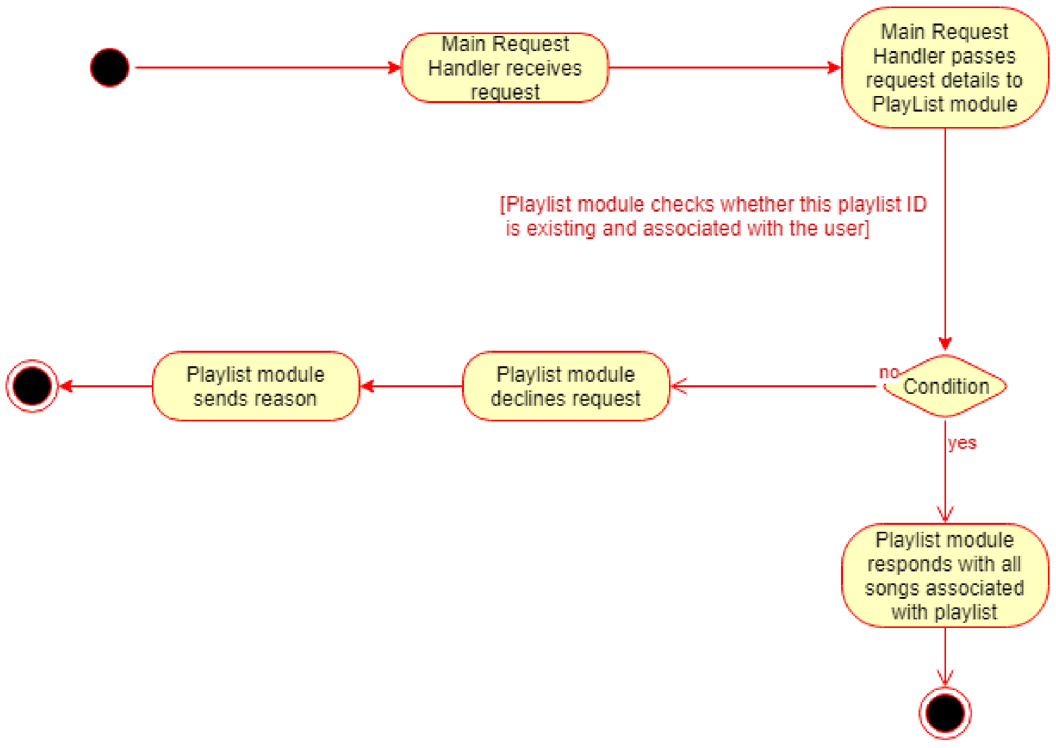
- **Playlist Module: Playlist Retrieval Activity Diagram**


Main Request Handler receives request
Main Request Handler passes request details to PlayList module
Playlist module responds with all playlists associated user

- **Playlist Module: Playlist Editing Activity Diagram**


Main Request Handler receives request
Main Request Handler passes request details to PlayList module
[Playlist Module checks if the playlist ID exists already
Condition
no
yes
Playlist module declines creation
Playlist module sends reason
Playlist module allows and edits the playlist

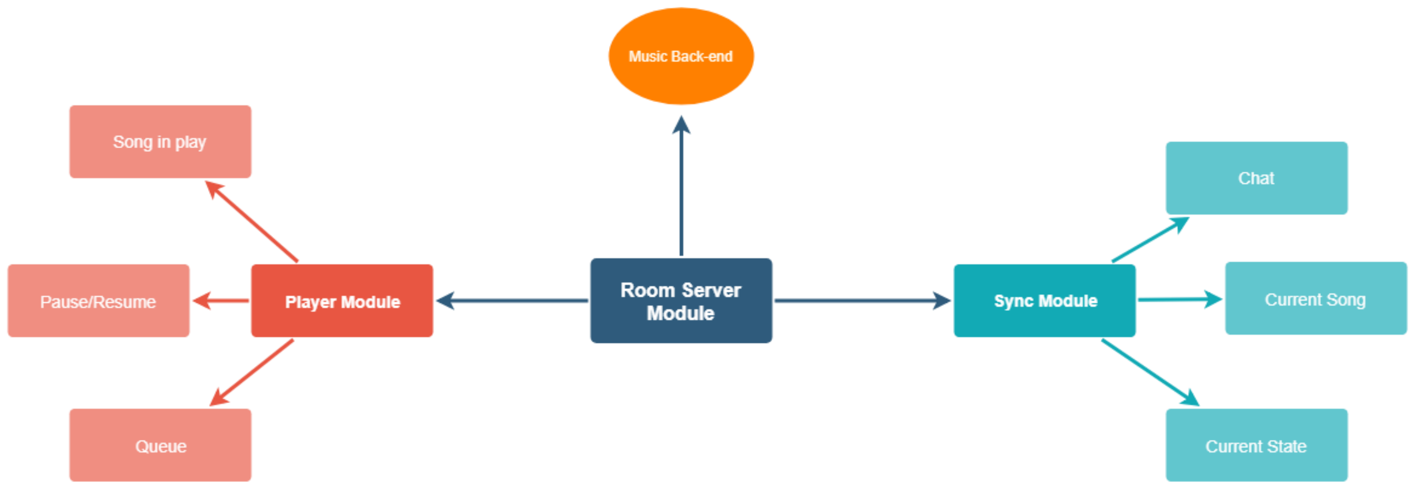
- **Playlist Module: Playlist Songs Retrieval Activity Diagram**


Main Request Handler receives request
Main Request Handler passes request details to PlayList module
[Playlist module checks whether this playlist ID is existing and associated with the user]
Condition
no
yes
Playlist module declines request
Playlist module sends reason
Playlist module responds with all songs associated with playlist

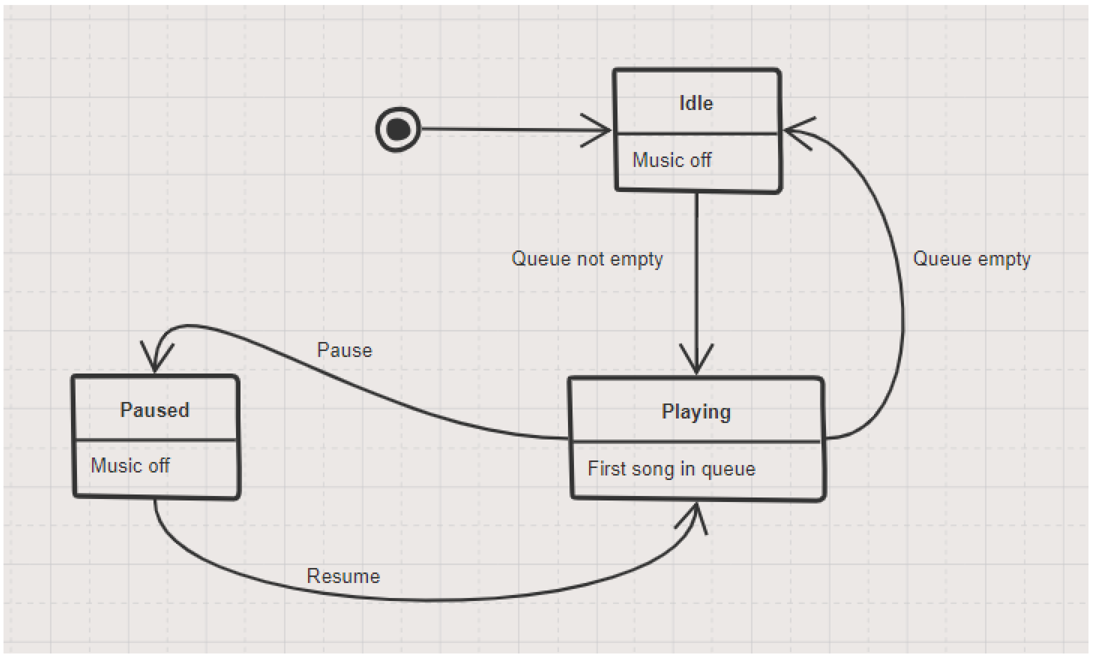
5. Room Server Module

This module is in control of handling music syncing within a single room; it can do all things including synced playing, stopping, pausing, and resuming. It also manages room chatting.

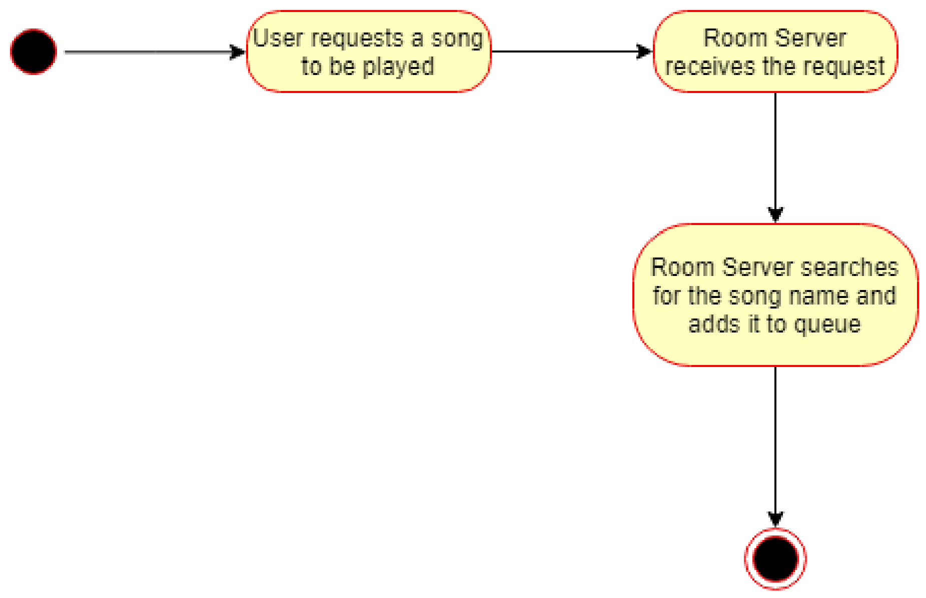
- **Room Server Module: Context Diagram**



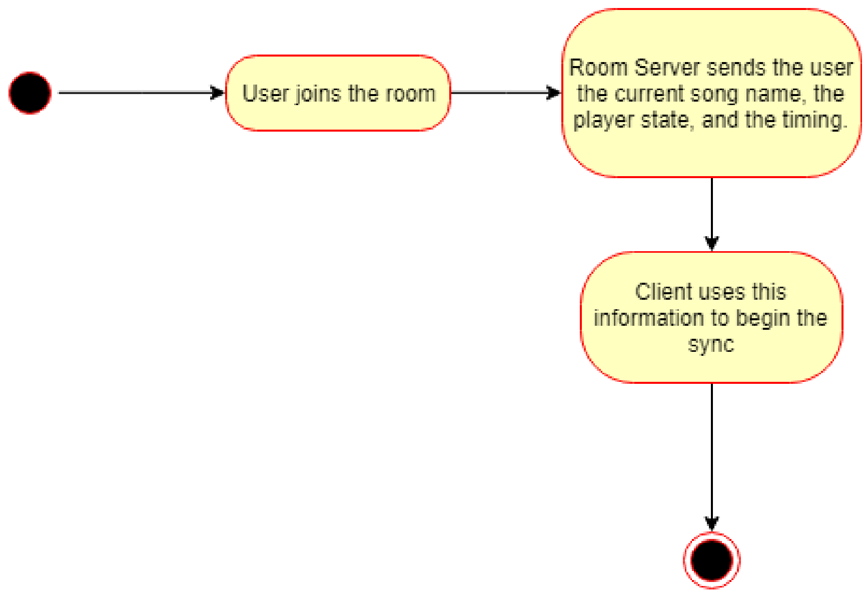
- **Room Server Module: Playing State Machine Diagram**



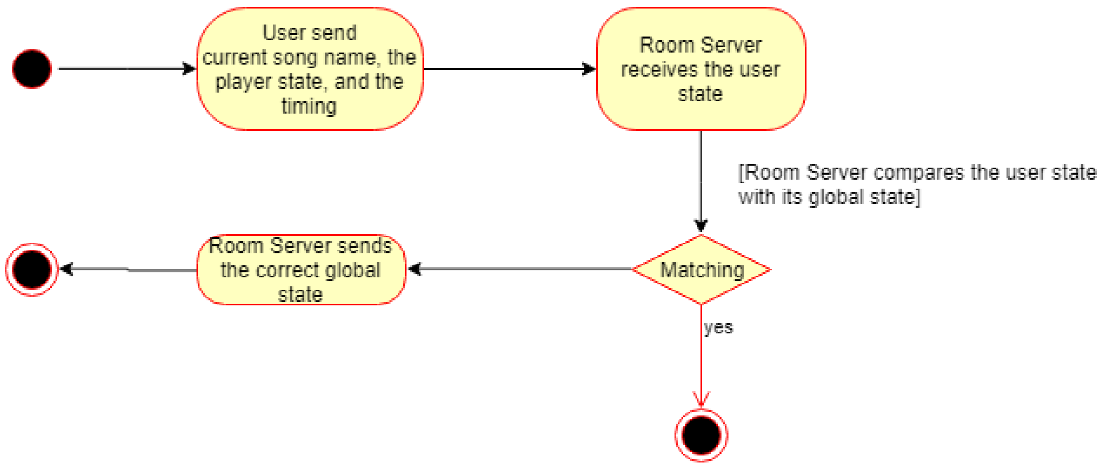
- **Room Server Module: Song Play Request Activity Diagram**


User requests a song to be played
Room Server receives the request
Room Server searches for the song name and adds it to queue

- **Room Server Module: [OnJoin] Activity Diagram**


User joins the room
Room Server sends the user the current song name, the player state, and the timing.
Client uses this information to begin the sync

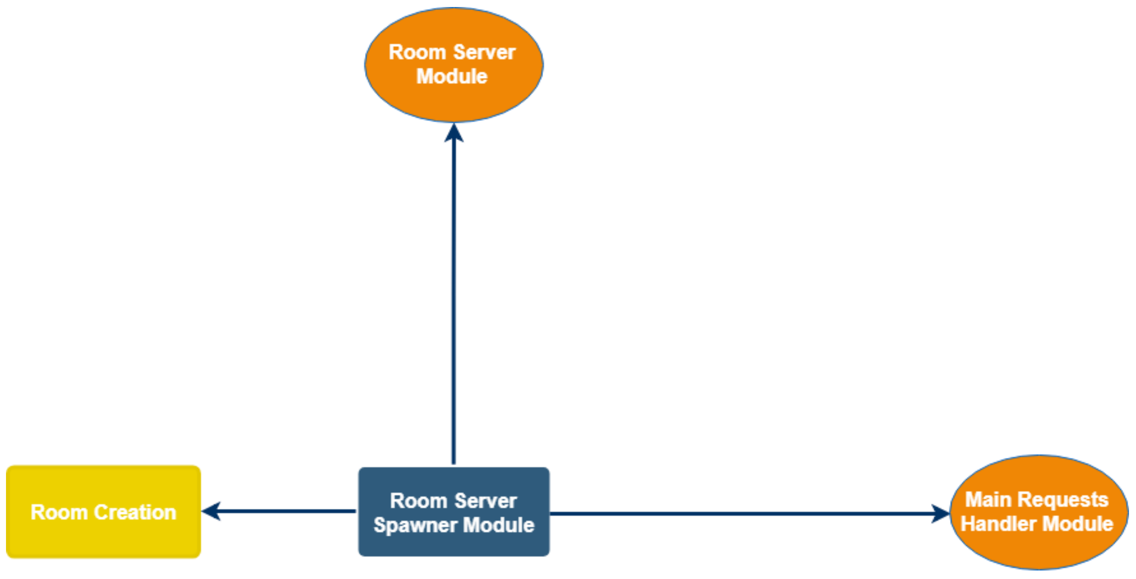
- **Room Server Module: Continuous Sync Activity Diagram**


User send current song name, the player state, and the timing
Room Server receives the user state
[Room Server compares the user state with its global state]
Matching
yes
Room Server sends the correct global state

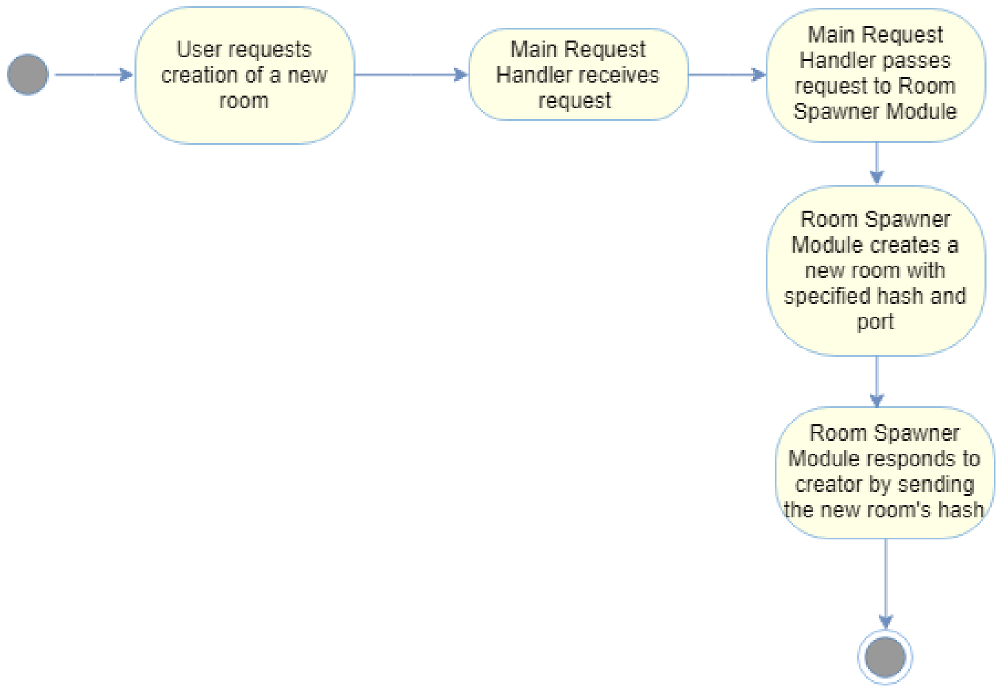
6. Room Server Spawner Module

This module is in control of spawning new room servers upon request from a user; it can do functions such as receiving and responding to room creation requests, launching room servers with correct parameters and handling room join requests.

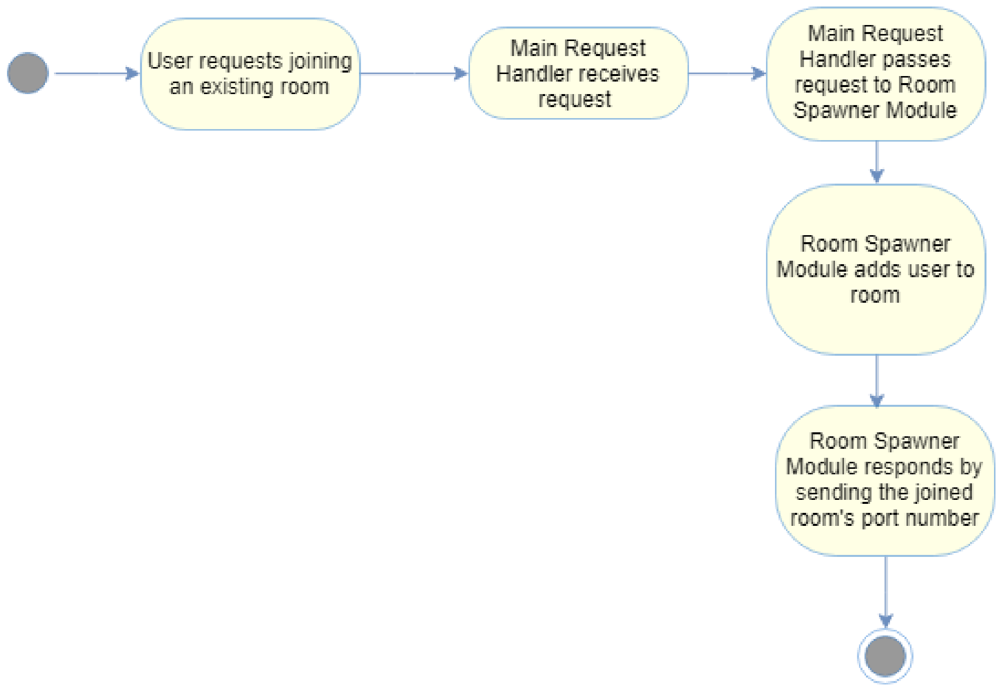
- **Room Server Spawner Module Context Diagram**



- **<u>Room Server Spawner Module: Room Creation Activity Diagram</u>**


User requests creation of a new room
Main Request Handler receives request
Main Request Handler passes request to Room Spawner Module
Room Spawner Module creates a new room with specified hash and port
Room Spawner Module responds to creator by sending the new room's hash

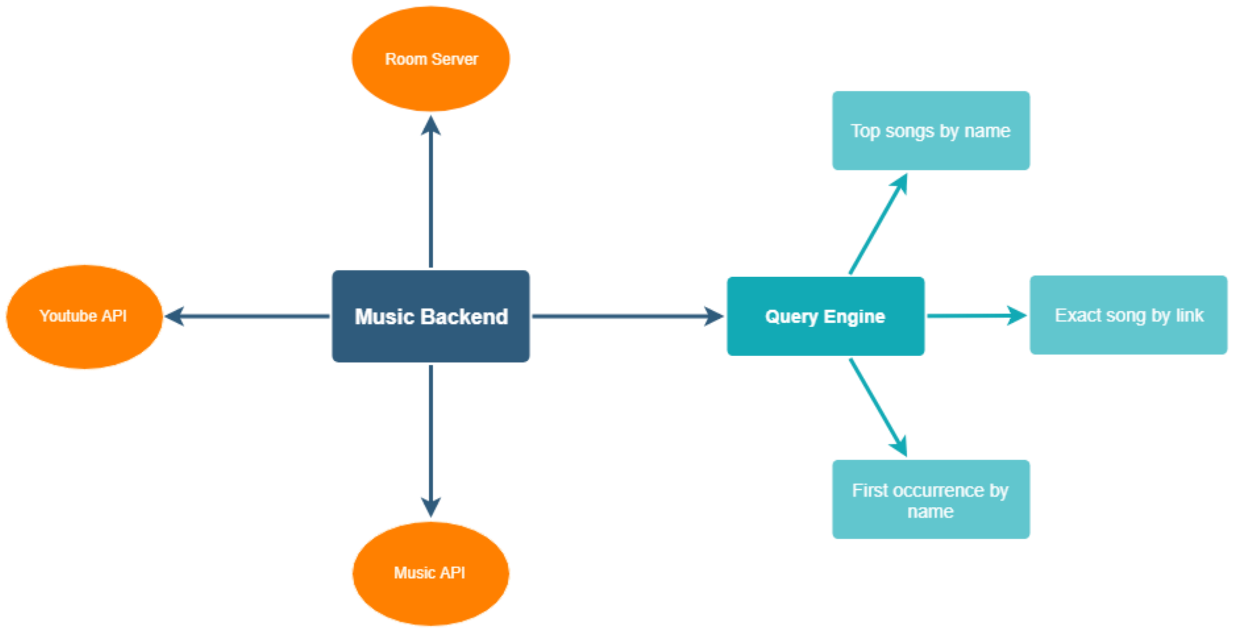
- **<u>Room Server Spawner Module: Room Joining Activity Diagram</u>**


User requests joining an existing room
Main Request Handler receives request
Main Request Handler passes request to Room Spawner Module
Room Spawner Module adds user to room
Room Spawner Module responds by sending the joined room's port number

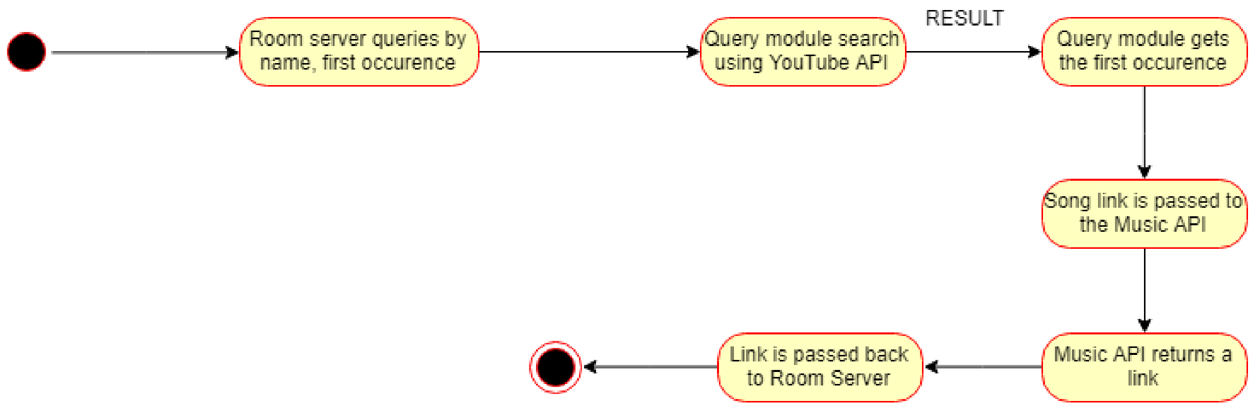
7. Music Backend Module

This module is in control of handling all music requests; it can do all things including querying a song by name, getting the top songs or the first occurrence, or even querying by link.

- <u>Music Backend Module: Context Diagram</u>



- <u>Music Backend Module: Basic Song Search Activity Diagram</u>


Room server queries by name, first occurence
Query module search using YouTube API
RESULT
Query module gets the first occurence
Song link is passed to the Music API
Music API returns a link
Link is passed back to Room Server

System Functions

[FR_MRH] Main Request Handler Module Functions

[FR_MRH_1] Basic Handler

Description: This function is the general handler to all requests, it then calls other specific handler as it see fits.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_2] Authorization Handler

Description: This function handles Authorize User requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_3] SignUp Handler

Description: This function handles SignUp requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_4] Send Invitation Handler

Description: This function handles room invitation requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_5] Retrieve Invitation Handler

Description: This function handles all retrieve invitation requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_6] Create Room Handler

Description: This function handles Create Room requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_7] Join Room Handler

Description: This function handles Join Room requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_8] Retrieve Profile Handler

Description: This function handles profile retrieval requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_9] Retrieve Playlists Handler

Description: This function handles playlist headers retrieval requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_10] Retrieve Playlist Handler

Description: This function handles songs in playlist retrieval requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MRH_11] Edit Playlist Handler

Description: This function handles playlist edit/creation requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MHR_12] Retrieve Friendlist Handler

Description: This function handles friendlist retrieval requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MHR_13] Follow Friend Handler

Description: This function handles following friend follow requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_MHR_14] Search User Handler

Description: This function handles user search requests.

Inputs: [string] URL

Outputs: void

Pre-conditions: none

Post-conditions: none

[FR_STR] Storage Module Functions

[FR_STR_1] Retrieve User

Description: This function creates and fills a User object through the database

Inputs: [string] Username/[int] UserID

Outputs: [User] return type. Returns a user object, or “null” if username isn’t found.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_2] Retrieve Profile

Description: This function creates and fills a Profile object through the database

Inputs: [int] UserID

Outputs: [Profile] return type. Returns a profile object, or “null” if UserID isn’t found.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_3] Save User

Description: This function saves/edits a User object to the database

Inputs: [User] user object

Outputs: [Boolean]. Returns a boolean, true if successful, false if not.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_4] Save Profile

Description: This function saves/edits a Profile object to the database

Inputs: [Profile] profile object

Outputs: [Boolean]. Returns a boolean, true if successful, false if not.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_5] Retrieve Playlists

Description: This function retrieves all user playlists.

Inputs: [User] user object OR [int] UserID

Outputs: [List<PlayListHeader>] return type. PlayListHeader is an object containing the playlist name, and its ID. Returns the list empty if user have no playlists, and null if UserID wasn't found.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_6] Retrieve Playlist

Description: This function retrieves a certain playlist

Inputs: [int] Playlist ID

Outputs: [Playlist] return type. Playlist is an object containing the playlist name, Playlist ID, and all songs included. Returns null if the Playlist ID couldn't be found.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_7] Saves Playlist

Description: This function saves or edit a playlist into the database

Inputs: [Playlist] object type.

Outputs: [Boolean] return type. "True" if succeeded, "False" if failed.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_7] Retrieve FriendList

Description: This function retrieves the user's FriendList

Inputs: [int] UserID OR [User] User object

Outputs: [FriendList] return type. FriendList is an object containing its owner UserID and all his/her friends UserIDs, and whether they are offline or online. Returns null if the UserID couldn't be found.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_8] Save Friend

Description: This function adds a friend to the User

Inputs: ([int] UserID OR [User] User object) AND ([int] FriendID OR [User] Friend object)

Outputs: [Boolean] return type. "True" if succeeded, "False" if failed.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_STR_9] Search Users

Description: This function retrieves users with a similar username

Inputs: [string] Username.

Outputs: [List<User>] return type. This is a list of User objects having similar Usernames. Will return "null" if no similar usernames exist.

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_SCI] Social Integration Module Functions

[FR_SCI_1] Authorize Login Function

Description: This function determines whether the logging in credentials are allowed or not

Inputs: [String] Username, [String] Password

Outputs: [Boolean] return type. This will be "True" if the user is Authorized, "False" if user isn't. [AuthorizationFailReason] return type, this shows exactly the reason the authorization failed.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_SCI_2] SignUp Function

Description: This function allows signing up.

Inputs: [String] Username, [String] Password, [String] Email.

Outputs: [Boolean] return type. This will be "True" if the user is Signed up, "False" if it couldn't sign the user up. [SignUpFailReason] return type, this shows exactly the reason the SignUp failed.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_SCI_2] Send Invitation Function

Description: This function registers a join-room invitation for another user.

Inputs: [string] InviterID, [string] InviteeID

Outputs: [Boolean] return type. Returns "True" if successful, and "False" if failed.

Pre-conditions: Database and Xampp both working.

Post-conditions: none.

[FR_SCI_3] Retrieve Invitations Function

Description: This function retrieves all pending invitations for the given user.

Inputs: ([int] UserID OR [User] User object)

Outputs: [List<Invitation>] return type. This is a list of Invitation objects. Will return "null" if no pending invitations exist.

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_SCI_4] Last Active Function

Description: This function is to be called every time the main request handler receives a request.

Inputs: [int] UserID

Outputs: void

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_SCI_5] Is Online Function

Description: This function determines whether the supplied UserID is online or not. (This will be used by the FriendList Class).

Inputs: [int] UserID

Outputs: [Boolean] return type. "True" if online, "False" if offline.

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_RMS] Room Server Module Functions

[FR_RMS_1] Add Song Function

Description: This function asks Music API for song by URL and plays it (if queue is empty) or adds it to the queue.

Inputs: [String] SongName

Outputs: [void]

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_RMS_1] OnJoin Function

Description: This function is triggered once a client joins the room, it sends the connection the latest synced state of the room [RoomState]. RoomState is an object containing the [String] song url, the [PlayerState] player state, and the [int] time.

Inputs: [Connection] ClientConnection

Outputs: [void]

Pre-conditions: Database and Xampp both working.

Post-conditions: none

[FR_RMS_2] Send Function

Description: This function sends a [List<byte>] packet to a certain connection.

Inputs: [Connection] ClientConnection, List<byte> Packet

Outputs: [void]

Pre-conditions: ClientConnection is established and working

Post-conditions: none

[FR_RMS_2] OnReceive Function

Description: This function receives a stream of bytes and turns them into a packet that is saved

Inputs: List<byte> stream

Outputs: [void]

Pre-conditions: none

Post-conditions: none

[FR_RMS_2] Parse Function

Description: This function parses the [RoomState] object into [List<byte>] packet.

Inputs: [RoomState] roomstate

Outputs: List<byte> return type. This will be “null” if conversion failed.

Pre-conditions: none

Post-conditions: none

[FR_RMS_3] Parse Function

Description: This function parses the [List<byte>] packet into [RoomState] object.

Inputs: List<byte> packet

Outputs: [RoomState] return type. This will be “null” if conversion failed.

Pre-conditions: none

Post-conditions: none

[FR_RMS_4] Pause Song Function

Description: This function pauses the current playing song

Inputs: None

Outputs: [void]

Pre-conditions: none

Post-conditions: none

[FR_RMS_5] Resume Song Function

Description: This function resumes the current playing song

Inputs: None

Outputs: [void]

Pre-conditions: none

Post-conditions: none

[FR_RSS] Room Server Spawner Module Functions

[FR_RSS_1] Generate Hash

Description: This function generates an unused hash to be associated with rooms

Inputs: None

Outputs: [string] hash

Pre-conditions: none

Post-conditions: none

[FR_RSS_2] Determine Port

Description: This function determines an unused port for the server to bind to and the client to listen to when a new room is created.

Inputs: None

Outputs: [ushort] port

Pre-conditions: none

Post-conditions: none

[FR_RSS_3] Create Room

Description: This function creates a room by making a [RoomServer] object

Inputs: [ushort] port, [string] hash

Outputs: [RoomServer] object. Returns “null” if failed.

Pre-conditions: none

Post-conditions: none

[FR_RSS_4] Get Port

Description: This function returns the port associated with a given hash for a user to use it to join a room.

Inputs: [string] hash

Outputs: [ushort] port

Pre-conditions: none

Post-conditions: none

[FR_MBE] Music Backend Module Functions

[FR_MBE_1] Query by Name

Description: This queries the YouTube API for top 10 songs for this name

Inputs: [string] name.

Outputs: [List<string>] urls. Returns a length of zero in case of no search results.

Pre-conditions: none

Post-conditions: none

[FR_MBE_2] Quick Query

Description: This returns the url of the first song in YouTube given its name

Inputs: [string] name.

Outputs: [string] url. Returns an empty string if no search results.

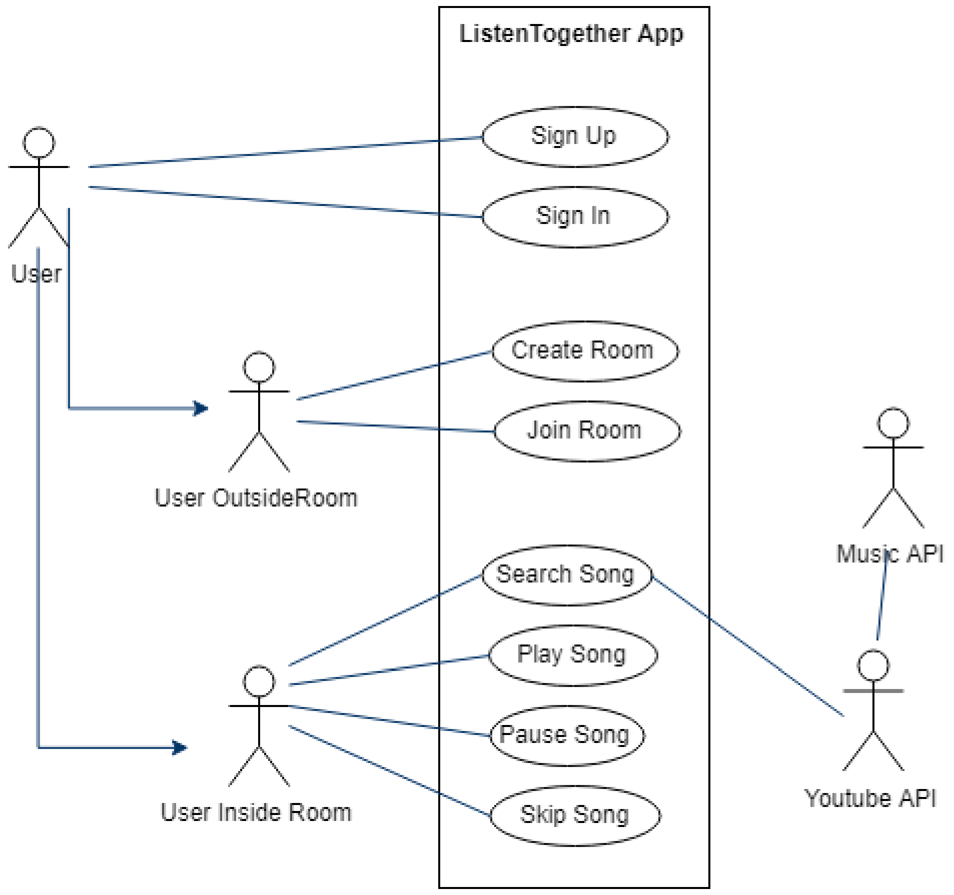
Pre-conditions: none

Post-conditions: none

System Models

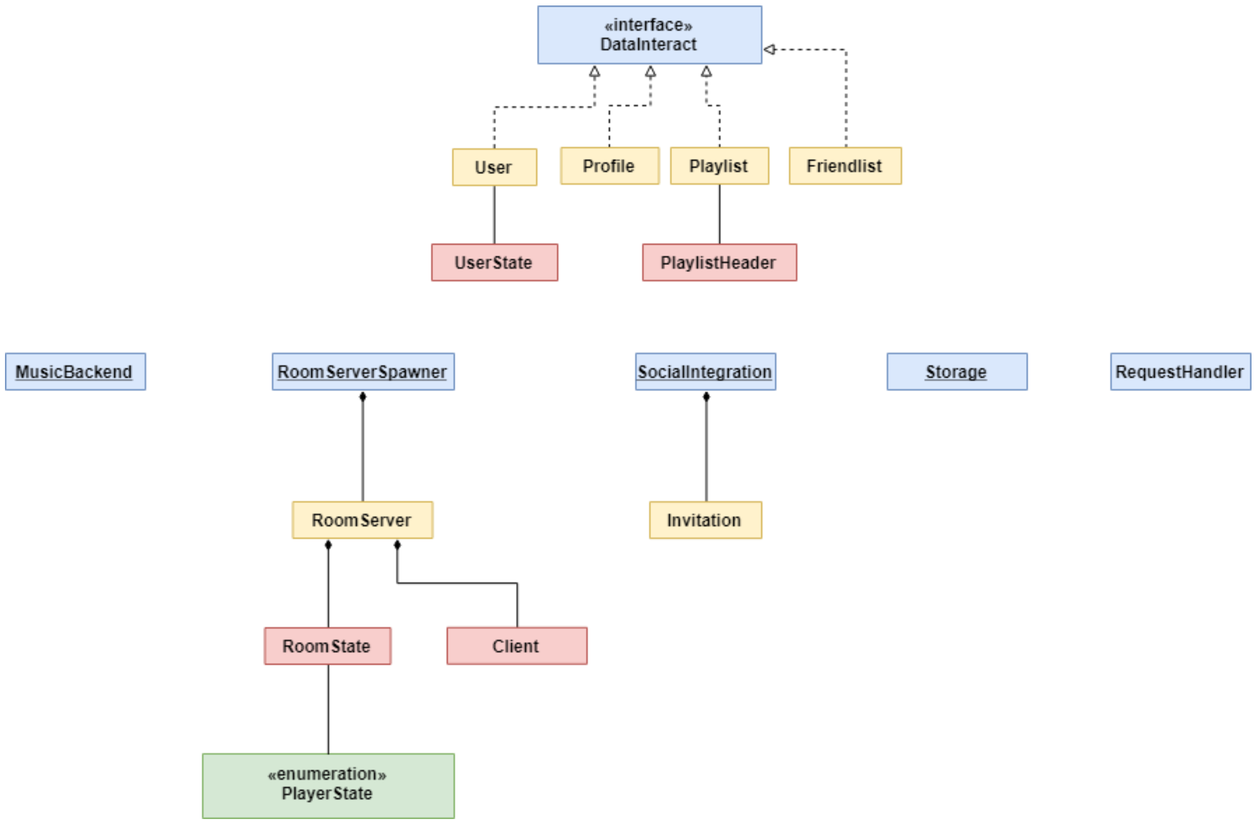
Use Case Diagrams

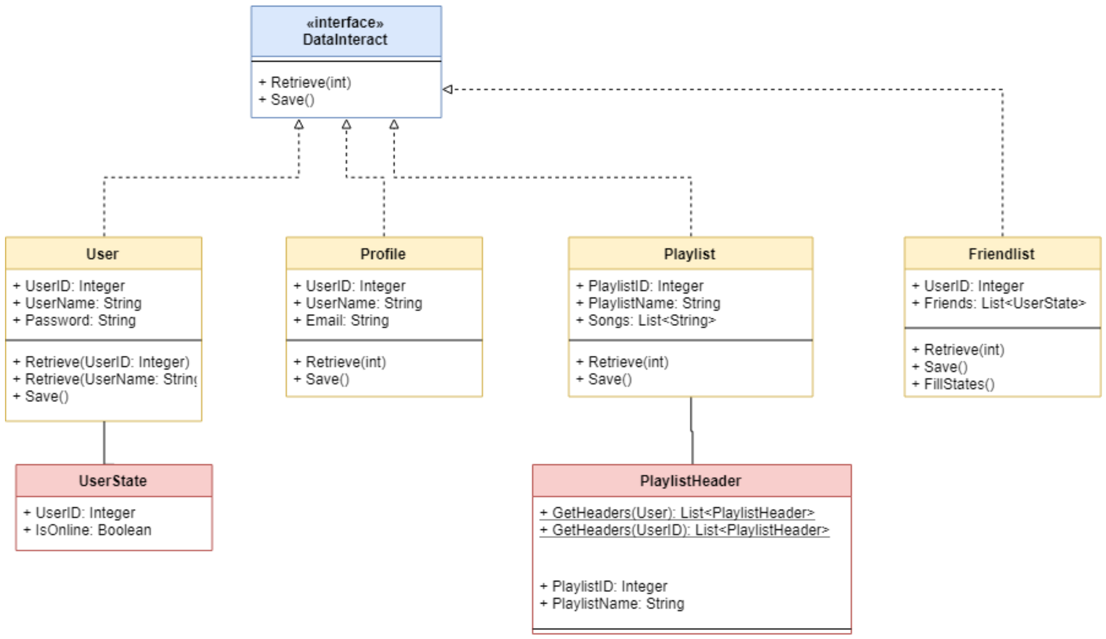
The following is a description of the use cases for Listen Together mobile application. In the first version of our system there is only one role, the role of the music listener, the mobile-end user.



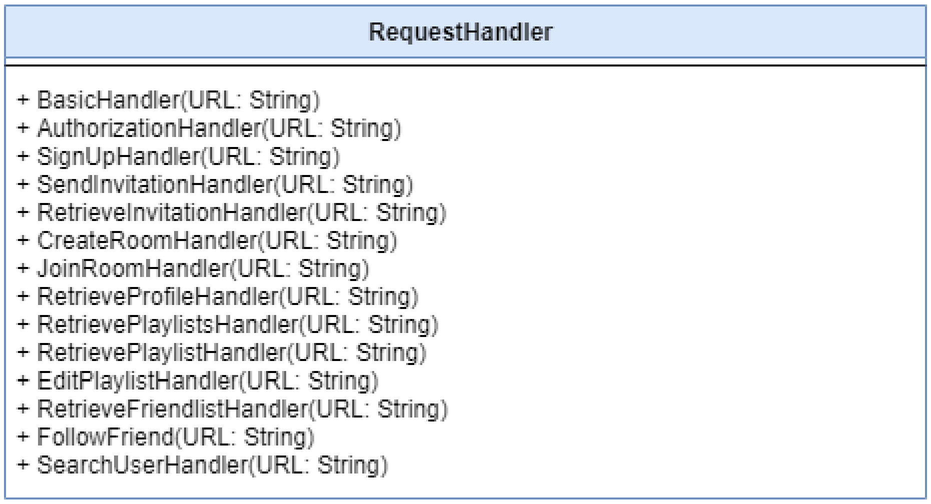
Class Diagram

Basic Diagrams



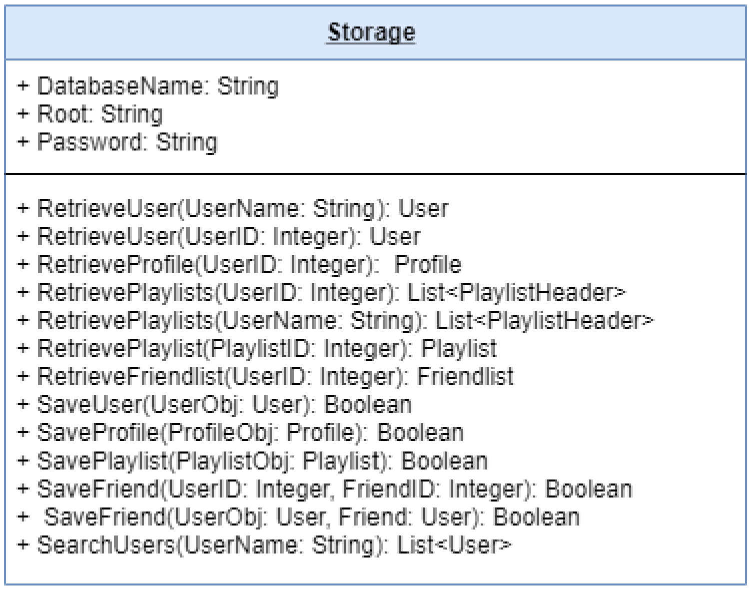


RequestHandler Class Diagram



+ BasicHandler(URL: String)
+ AuthorizationHandler(URL: String)
+ SignUpHandler(URL: String)
+ SendInvitationHandler(URL: String)
+ RetrieveInvitationHandler(URL: String)
+ CreateRoomHandler(URL: String)
+ JoinRoomHandler(URL: String)
+ RetrieveProfileHandler(URL: String)
+ RetrievePlaylistsHandler(URL: String)
+ RetrievePlaylistHandler(URL: String)
+ EditPlaylistHandler(URL: String)
+ RetrieveFriendlistHandler(URL: String)
+ FollowFriend(URL: String)
+ SearchUserHandler(URL: String)

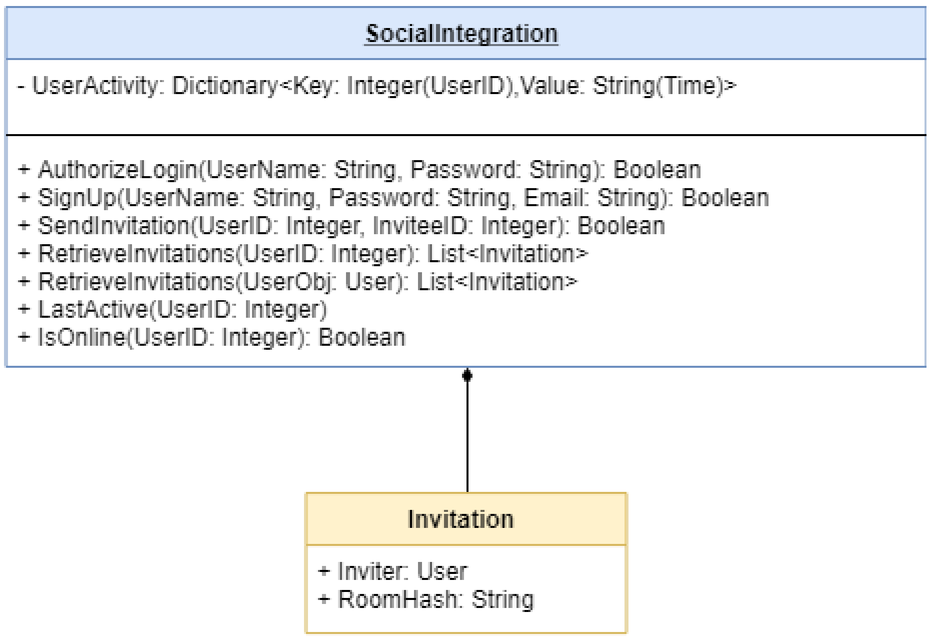
Storage Class Diagram



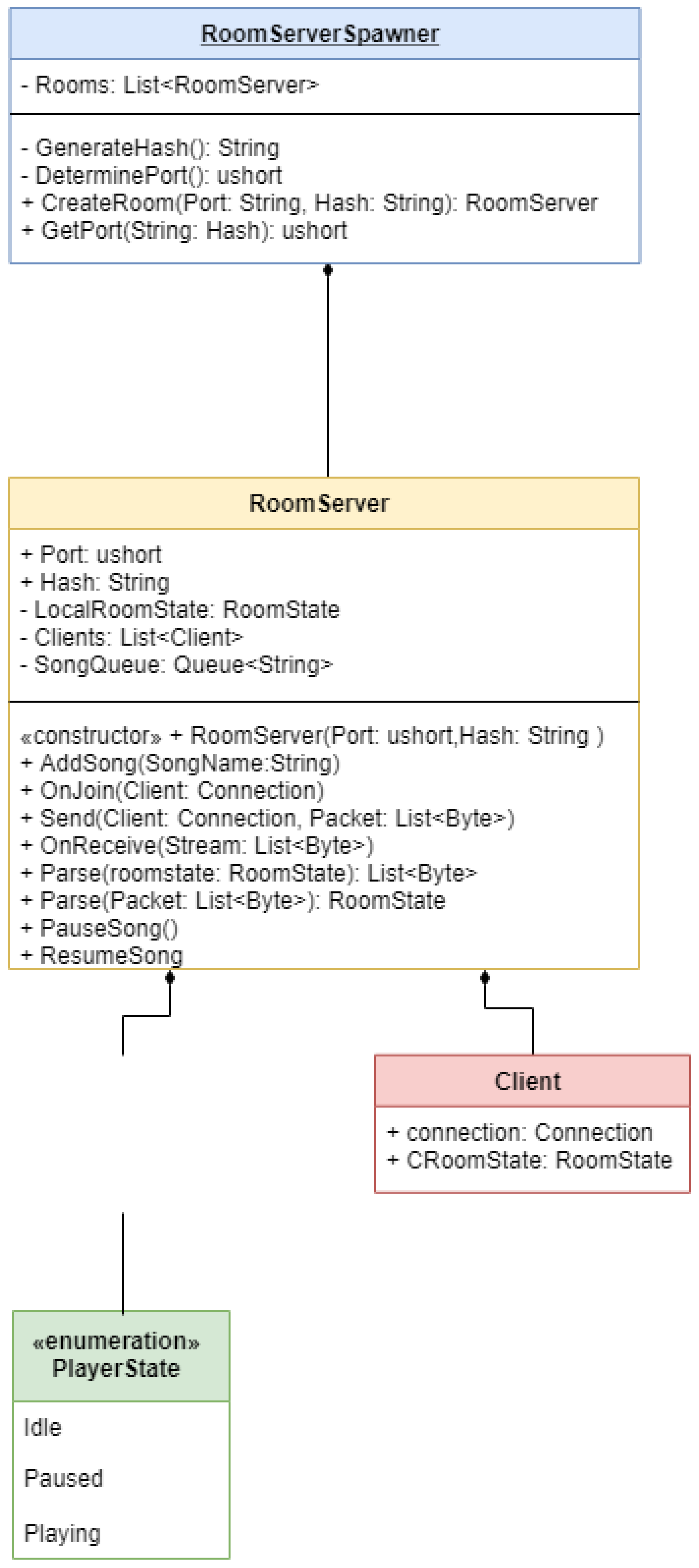
+ DatabaseName: String
+ Root: String
+ Password: String

+ RetrieveUser(UserName: String): User
+ RetrieveUser(UserID: Integer): User
+ RetrieveProfile(UserID: Integer): Profile
+ RetrievePlaylists(UserID: Integer): List<PlaylistHeader>
+ RetrievePlaylists(UserName: String): List<PlaylistHeader>
+ RetrievePlaylist(PlaylistID: Integer): Playlist
+ RetrieveFriendlist(UserID: Integer): Friendlist
+ SaveUser(UserObj: User): Boolean
+ SaveProfile(ProfileObj: Profile): Boolean
+ SavePlaylist(PlaylistObj: Playlist): Boolean
+ SaveFriend(UserID: Integer, FriendID: Integer): Boolean
+ SaveFriend(UserObj: User, Friend: User): Boolean
+ SearchUsers(UserName: String): List<User>

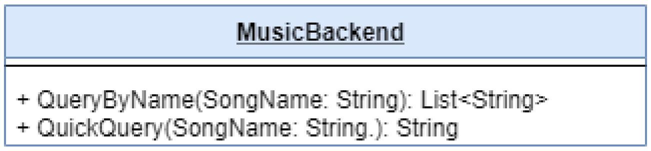
Social Integration Class Diagram


SocialIntegration
- UserActivity: Dictionary<Key: Integer(UserID),Value: String(Time)>
+ AuthorizeLogin(UserName: String, Password: String): Boolean
+ SignUp(UserName: String, Password: String, Email: String): Boolean
+ SendInvitation(UserID: Integer, InviteeID: Integer): Boolean
+ RetrieveInvitations(UserID: Integer): List<Invitation>
+ RetrieveInvitations(UserObj: User): List<Invitation>
+ LastActive(UserID: Integer)
+ IsOnline(UserID: Integer): Boolean
Invitation
+ Inviter: User
+ RoomHash: String

RoomServerSpawner Class Diagram


RoomServerSpawner
- Rooms: List<RoomServer>
- GenerateHash(): String
- DeterminePort(): ushort
+ CreateRoom(Port: String, Hash: String): RoomServer
+ GetPort(String: Hash): ushort
RoomServer
+ Port: ushort
+ Hash: String
- LocalRoomState: RoomState
- Clients: List<Client>
- SongQueue: Queue<String>
«constructor» + RoomServer(Port: ushort,Hash: String)
+ AddSong(SongName:String)
+ OnJoin(Client: Connection)
+ Send(Client: Connection, Packet: List<Byte>)
+ OnReceive(Stream: List<Byte>)
+ Parse(roomstate: RoomState): List<Byte>
+ Parse(Packet: List<Byte>): RoomState
+ PauseSong()
+ ResumeSong
Client
+ connection: Connection
+ CRoomState: RoomState
«enumeration»
PlayerState
Idle
Paused
Playing

MusicBackend Class Diagram


MusicBackend
+ QueryByName(SongName: String): List<String>
+ QuickQuery(SongName: String.): String

Non-Functional Requirements

[NFR_SC] Scalability Requirements

The system should be able to expand and shrink dependent on how many users are using the application at what times. By using smart load balancing and other advanced techniques, the system can withstand the high loads on the servers, while still shrinking at other times in order to avoid the costly hosting prices.

[NFR_ST] Security Requirement

The system should be able to be secure enough in order to avoid giving out data to non-allowed users. Careful encryption for passwords, guards on clients to avoid injections, and a lot of other techniques are to be expected here. The goal is to make everyone feel safe enough to enjoy using the application.

[NFR_U] Usability Requirements

The system should be usable by everyone, from 1 year olds to 100 year olds, everyone should be able to easily understand and play music without having to deal with the headache of long menus and complicated options. The idea is to make the user experience as simple as possible.

[NFR_P] <Performance> Requirements

The system should perform very well by any mobile phone not older than 3/4 years. No slowing between interfaces and button clicks, no long waiting for songs to be retrieved, played, or paused.

[NFR_P_1] CPU & Memory Requirements

When an application is developed to run on a particular software platform such as Java ME, Android, iOS etc. It can in theory be installed and run on any device that supports that OS platform. However, for any given OS, the supported devices could have a very wide range of capabilities in terms of CPU speed and available memory.

[NFR_P_2] Different Network Protocol Requirements

Mobile devices can communicate with the network on one or more protocols such as SMS, USSD, WiFi, EDGE, UMTS, LTE etc. Certain functions in your application may not perform well (or not perform at all) on certain protocols.

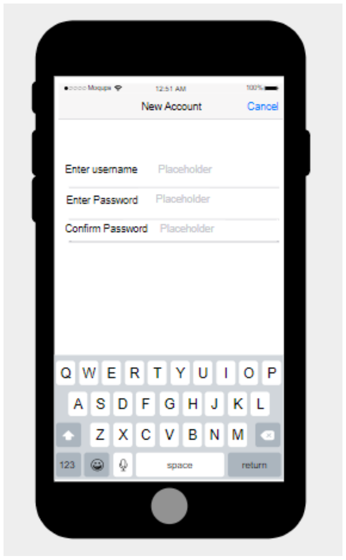
[NFR_P_3] Signal Strength Requirements

A mobile application is more likely to encounter a network drop or signal strength reduction situation than the desktop version of the application due to the inherently mobile nature of the platform. Some features may not be either network-fault tolerant and might not degrade or fail gracefully in such a situation.

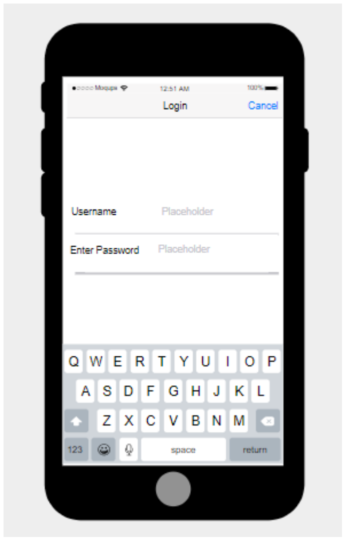
System Interfaces

User Interfaces

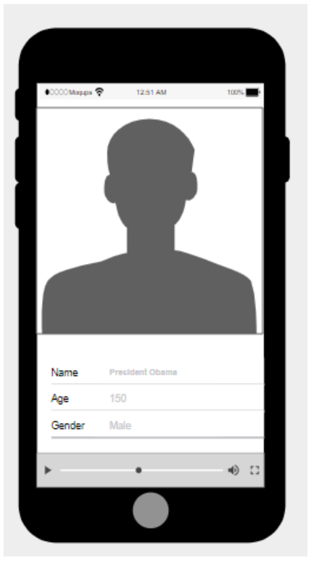
- Sign Up



- Sign In



- User Profile



Name	President Obama
Age	150
Gender	Male