Listen Together

SRS

Introduced By G30

Version	Written By	Reviewed By	Approved By	Date
0.X	Matthew Emile Lamees Emad	Lamees Emad Matthew Emile		

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.

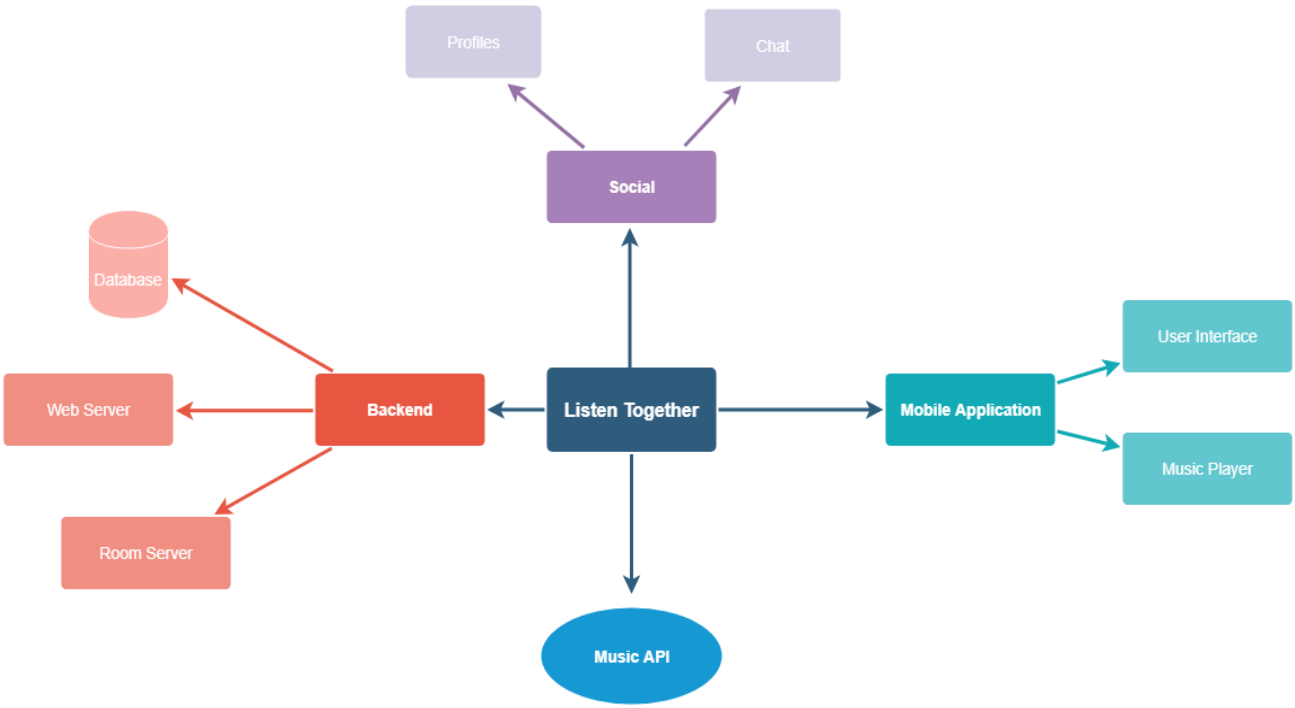
References

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

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

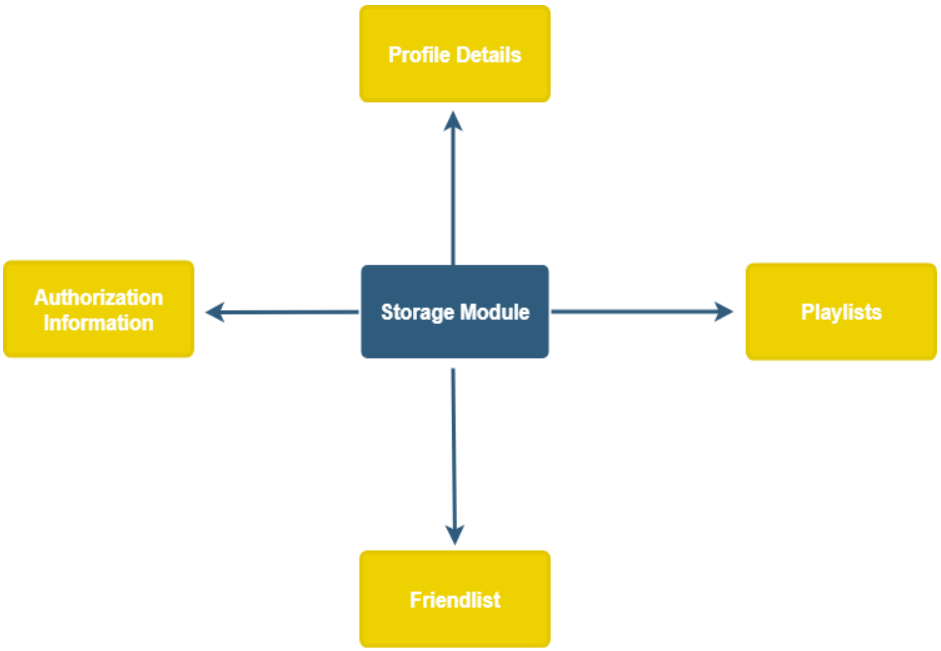
1. Main Request Handler Module

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.

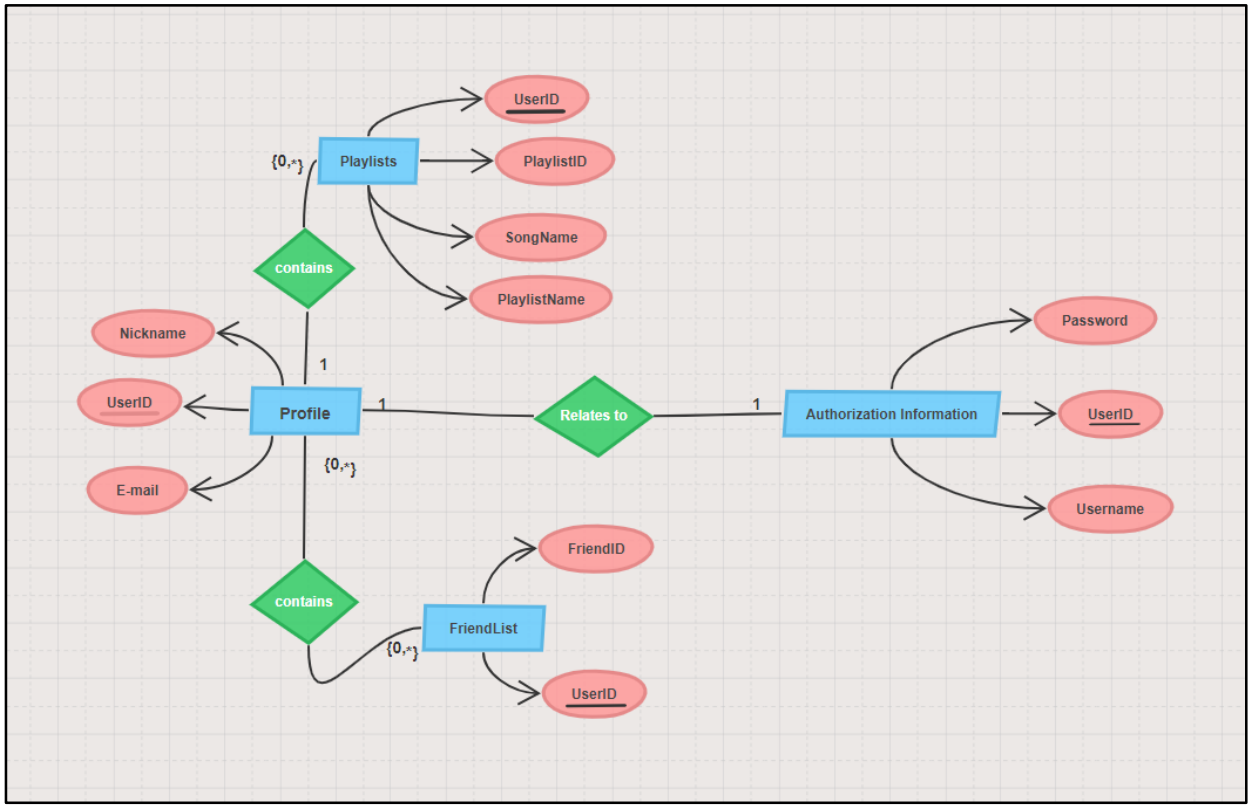
- **Storage Module: Context Diagram**



Details on stored data:

1. Profile Details: UserID, Nickname, E-mail
2. Authorization Information: UserID, Username, Password
3. Playlists: UserID, PlaylistID, PlaylistName, SongName
4. Friendlist: UserID, FriendID

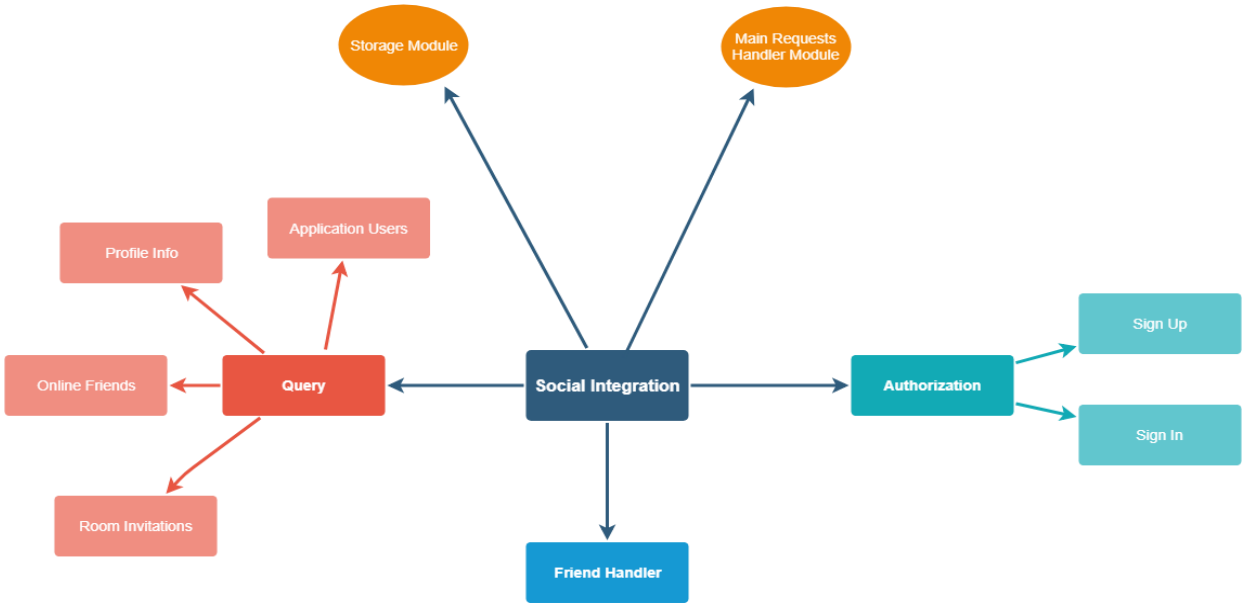
- **Storage Module: ER Diagram**



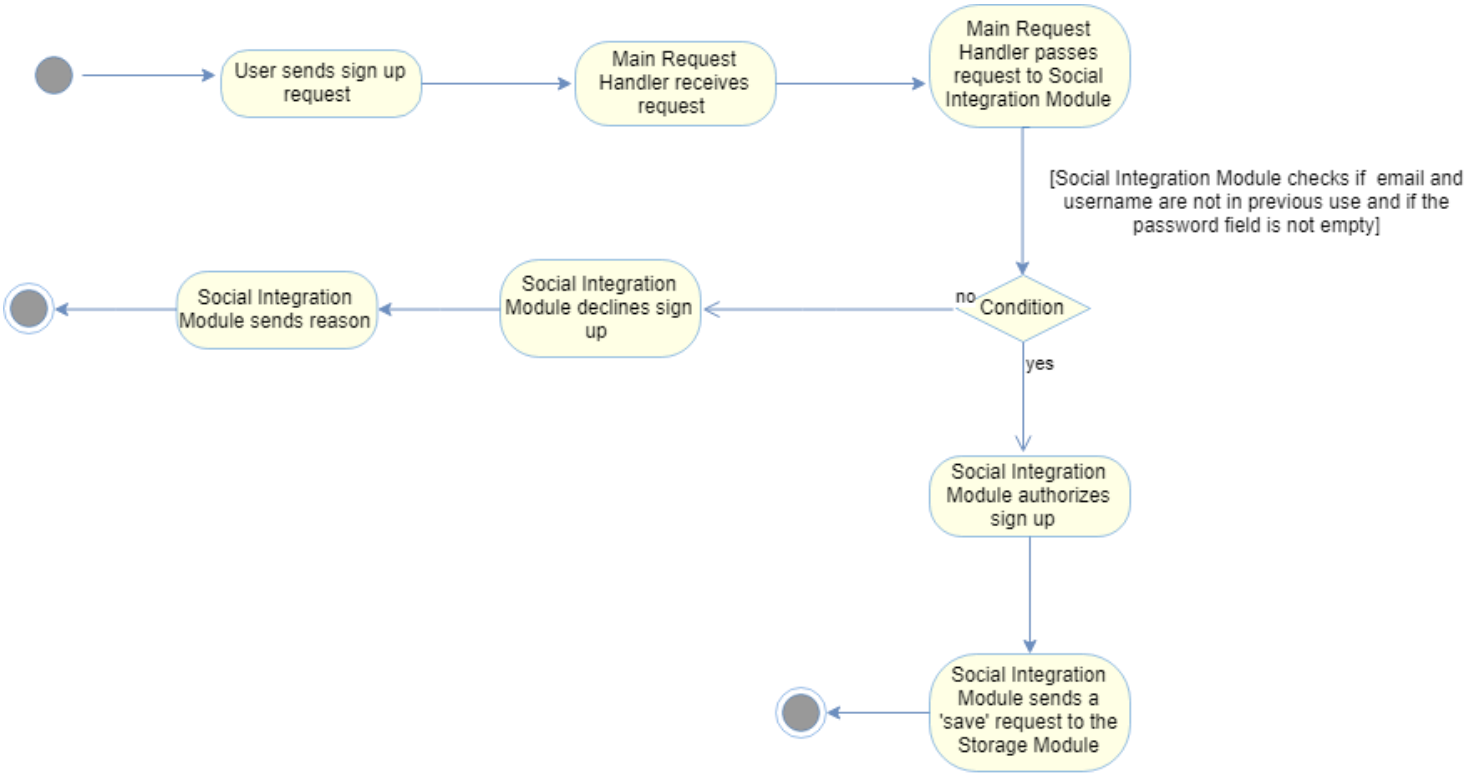
3. Social Integration Module

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.

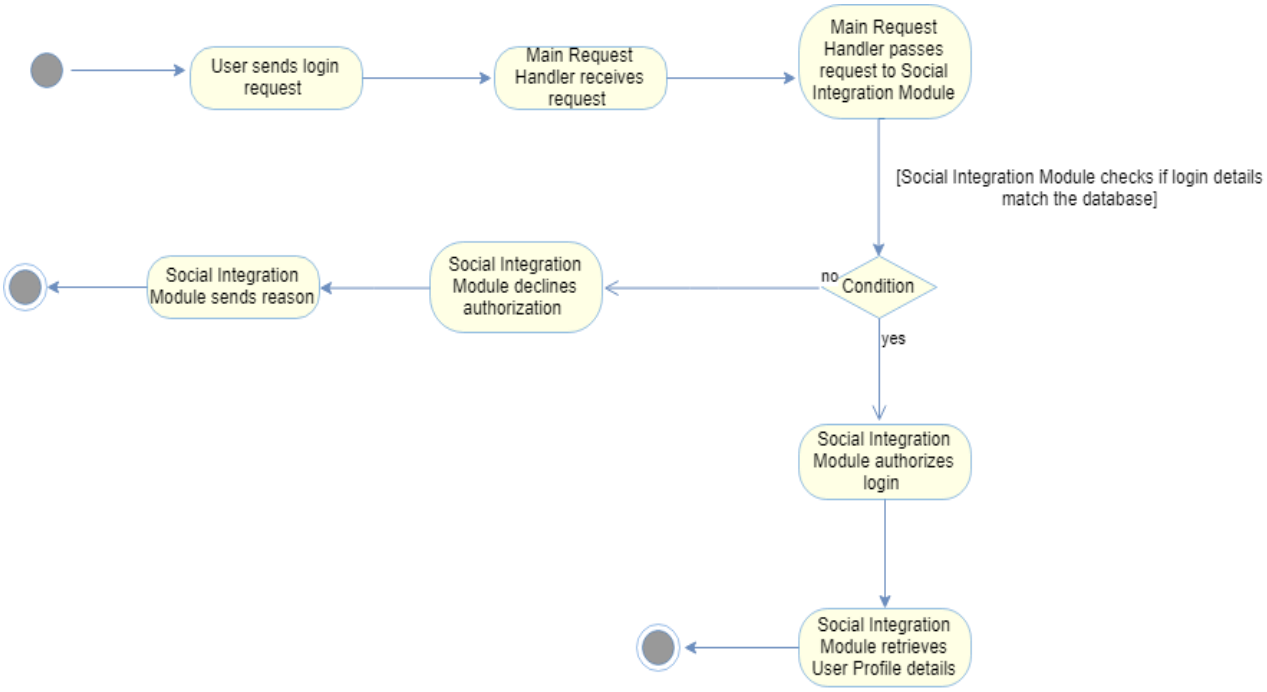
- **Social Integration Module: Context Diagram**



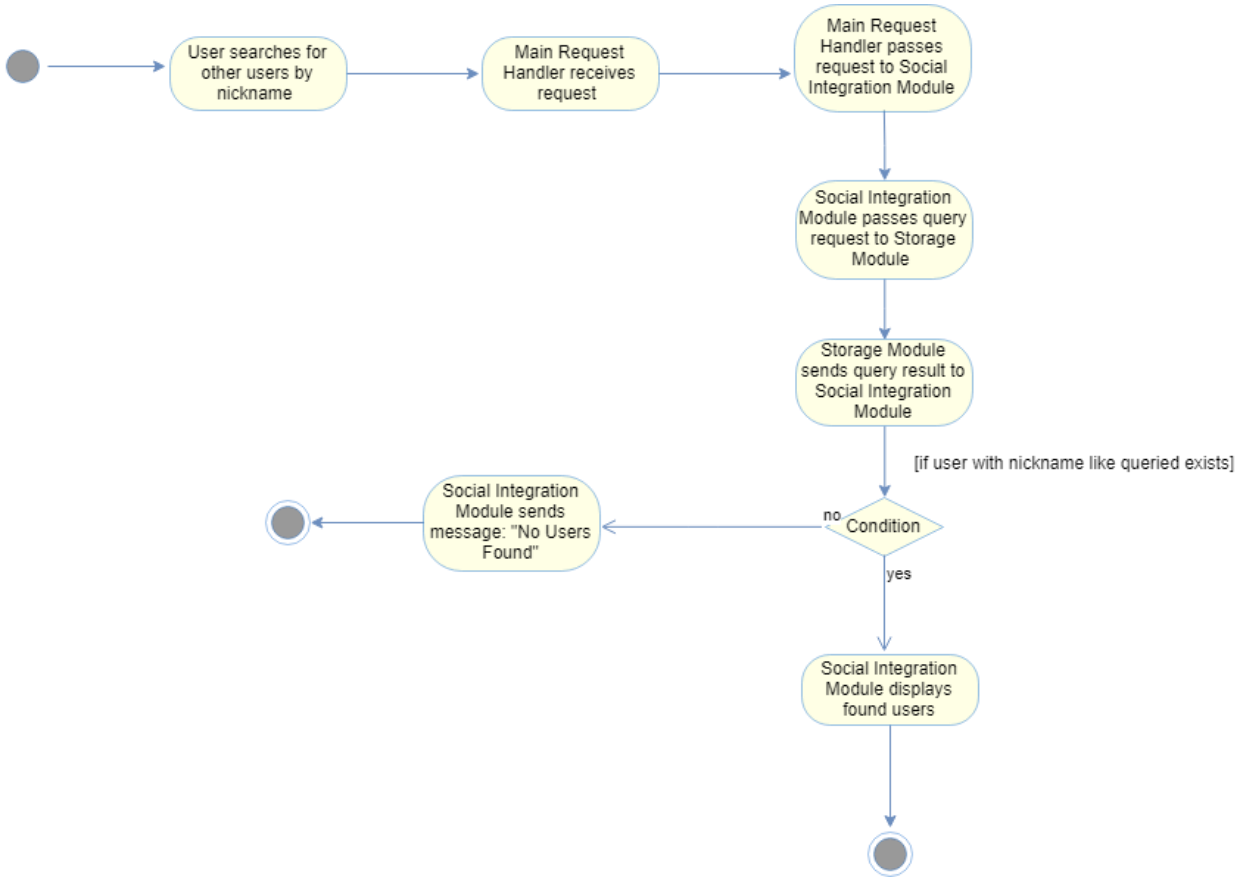
- **Social Integration Module: Login Activity Diagram**



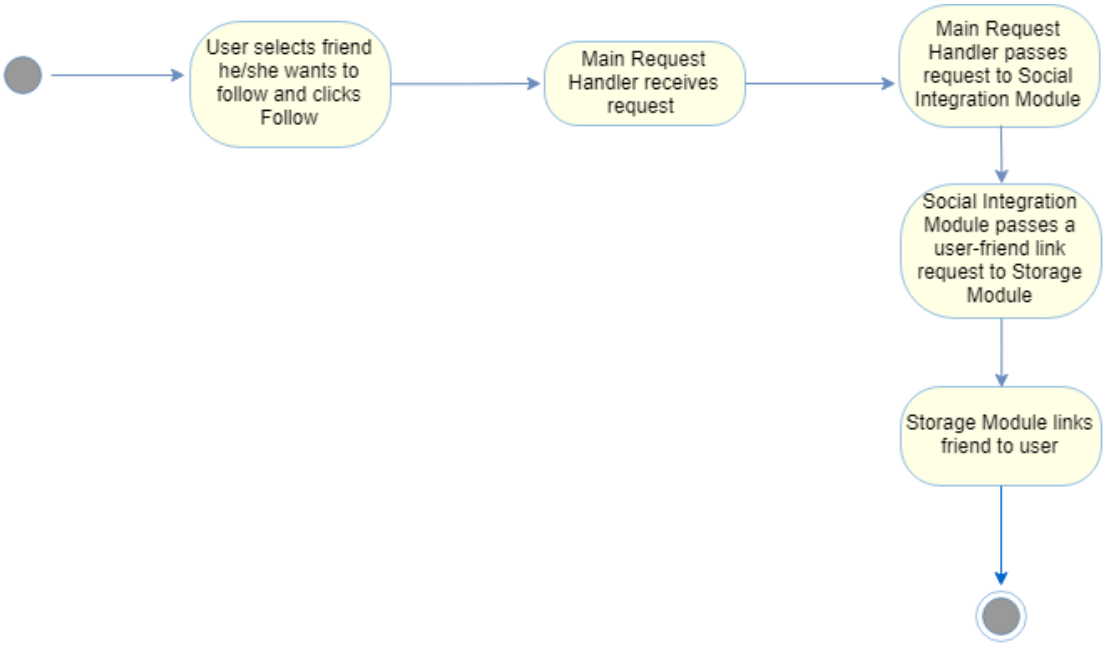
- **Social Integration Module: Sign Up Activity Diagram**



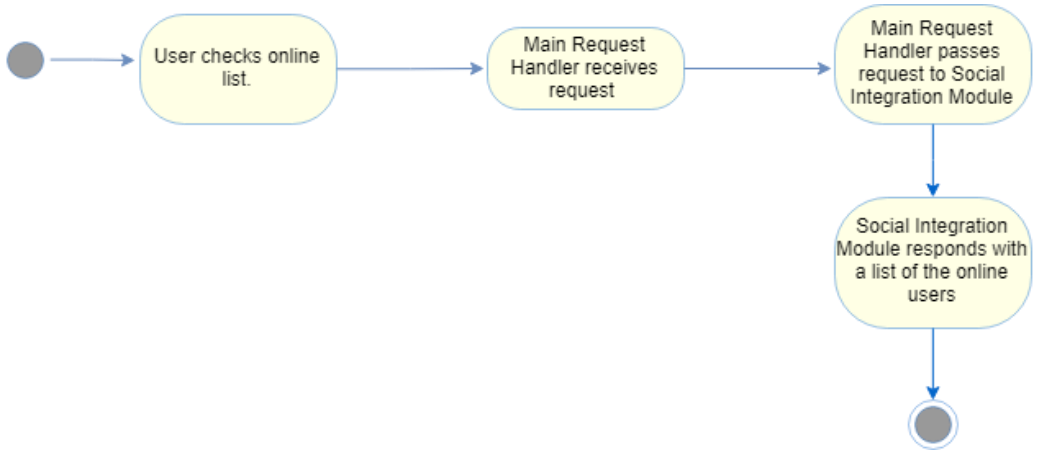
- **Social Integration Module: Search User Activity Diagram**



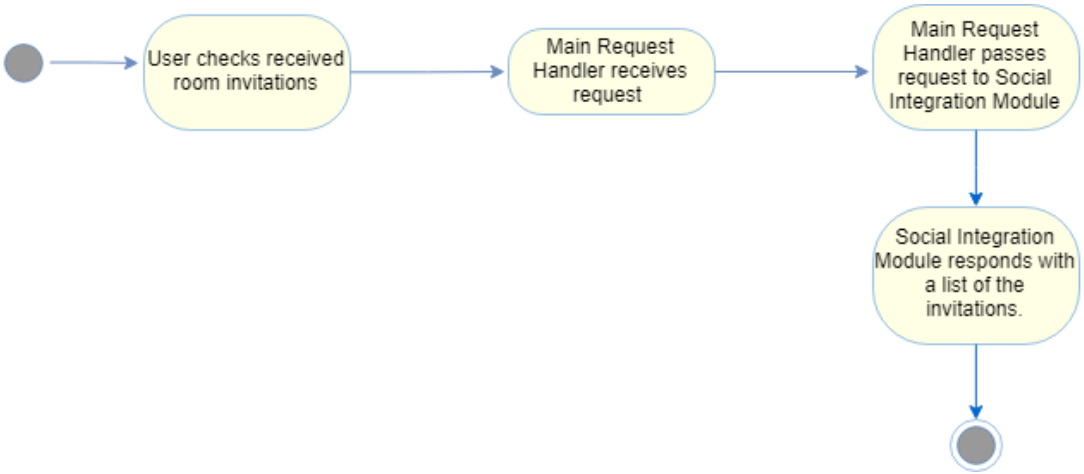
- **Social Integration Module: Follow Friend Activity Diagram**



- **Social Integration Module: Online Users Activity Diagram**



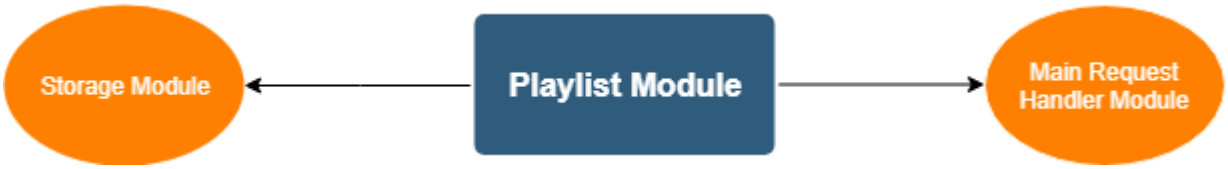
- **Social Integration Module: Online Users Activity Diagram**



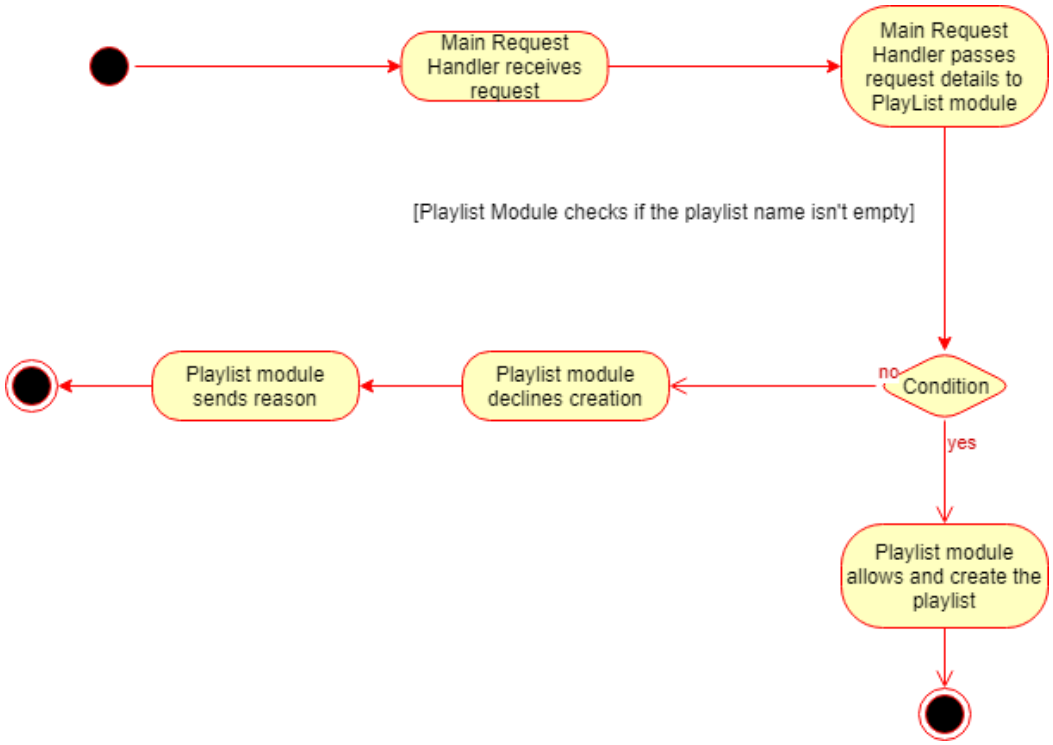
4. Playlist Module

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

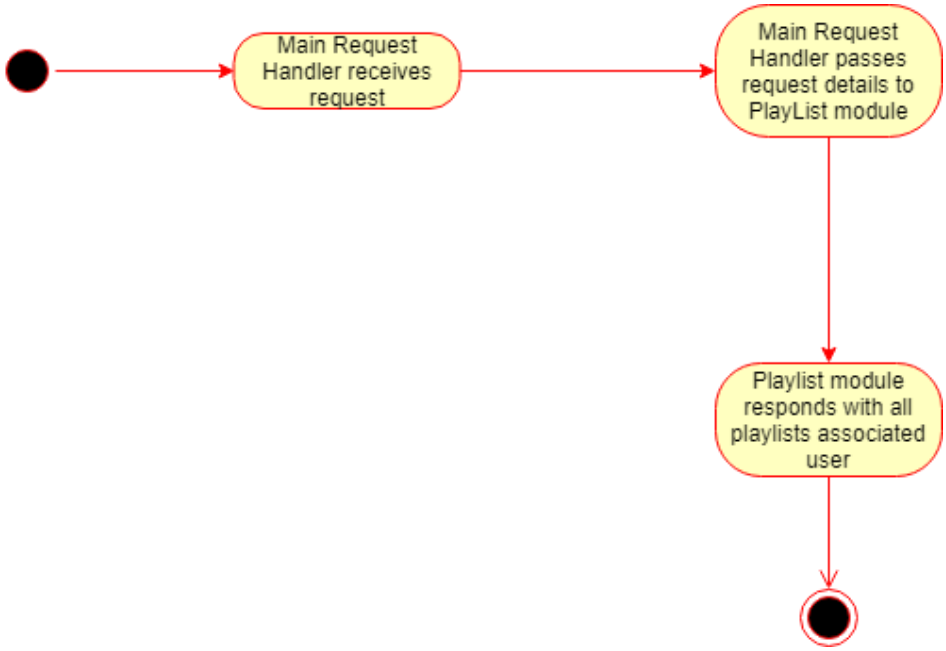
- **Playlist Module: Context Diagram**



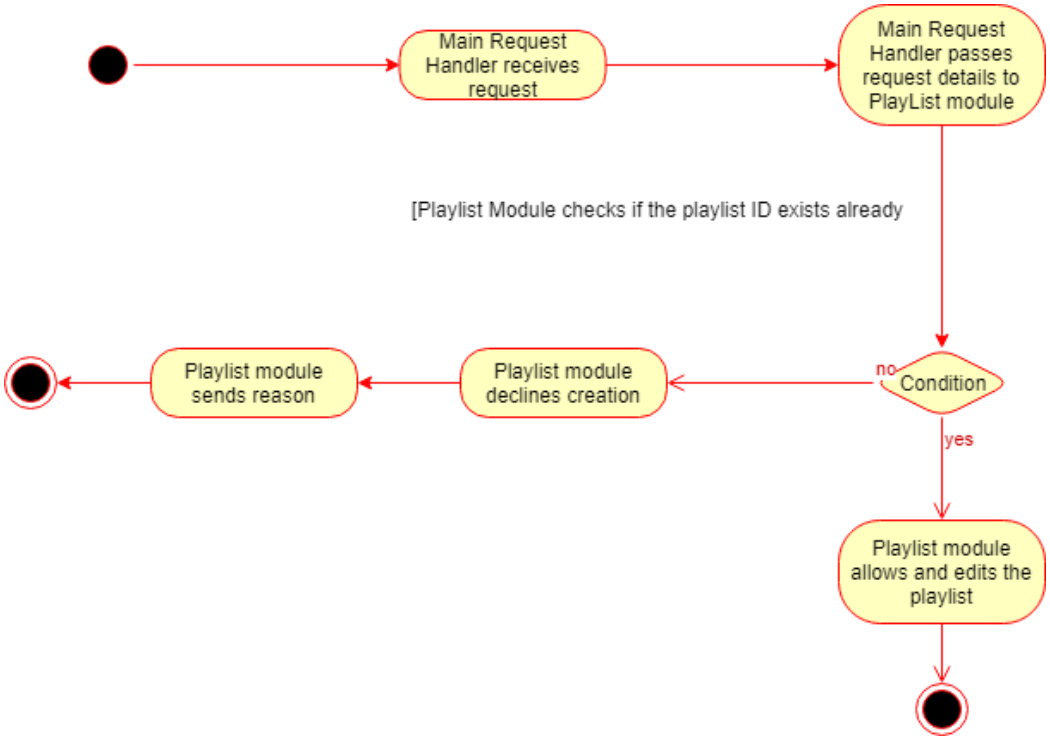
- **Playlist Module: Playlist Creation Activity Diagram**



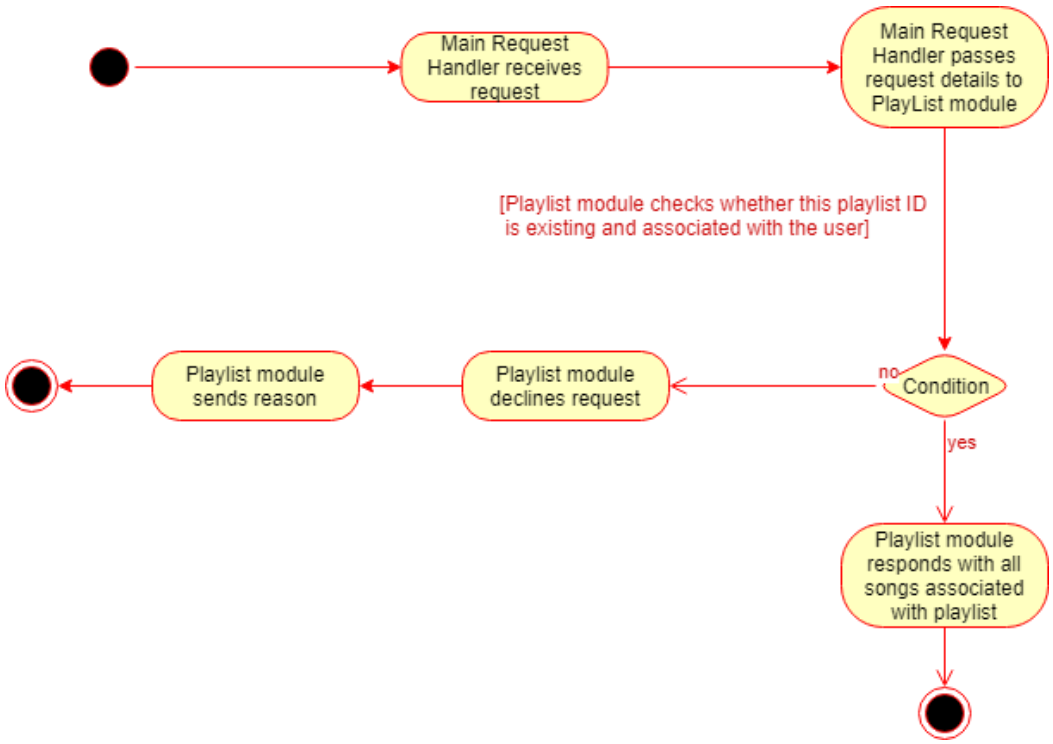
- **Playlist Module: Playlist Editing Activity Diagram**



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



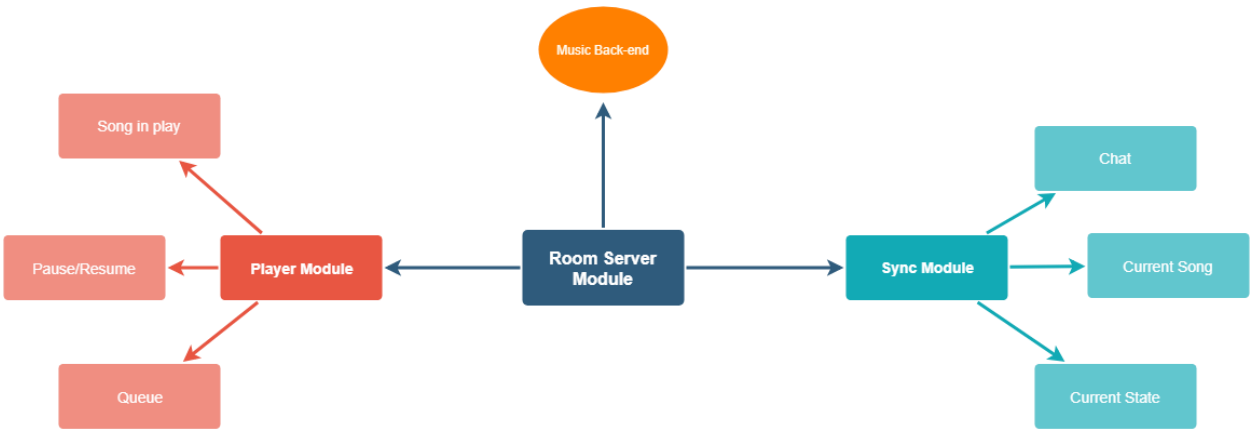
- **Playlist Module: Playlist Retrieve Songs Activity Diagram**



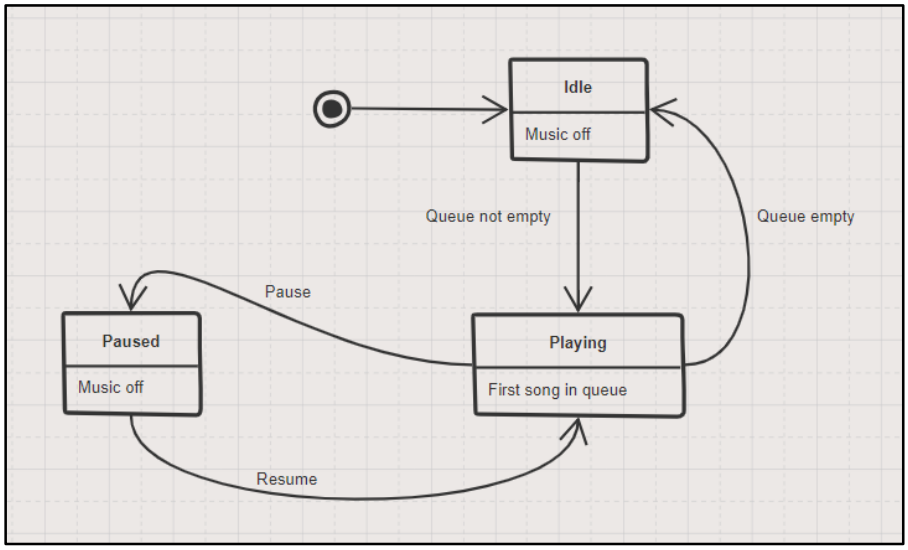
5. Room Server Module

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.

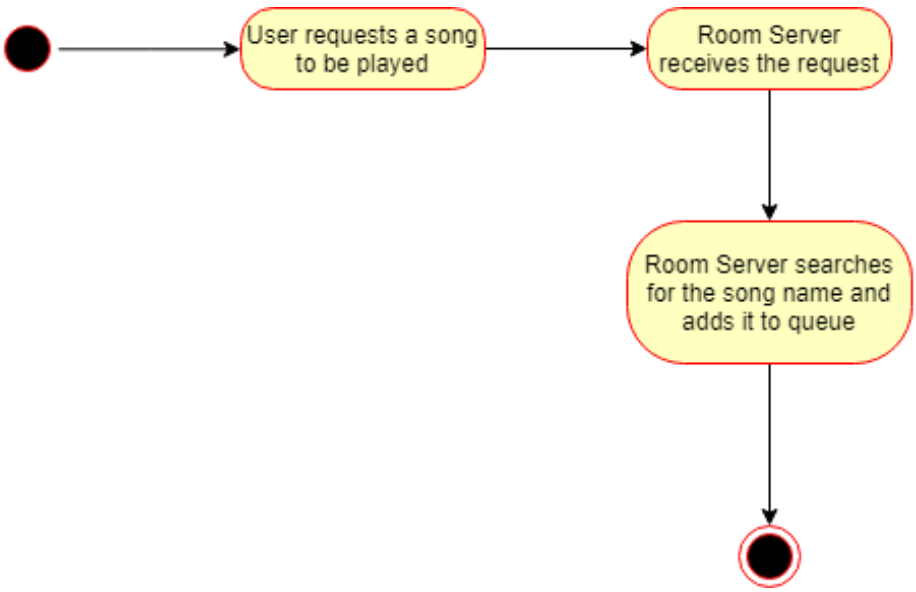
- **Room Server Module: Context Diagram**



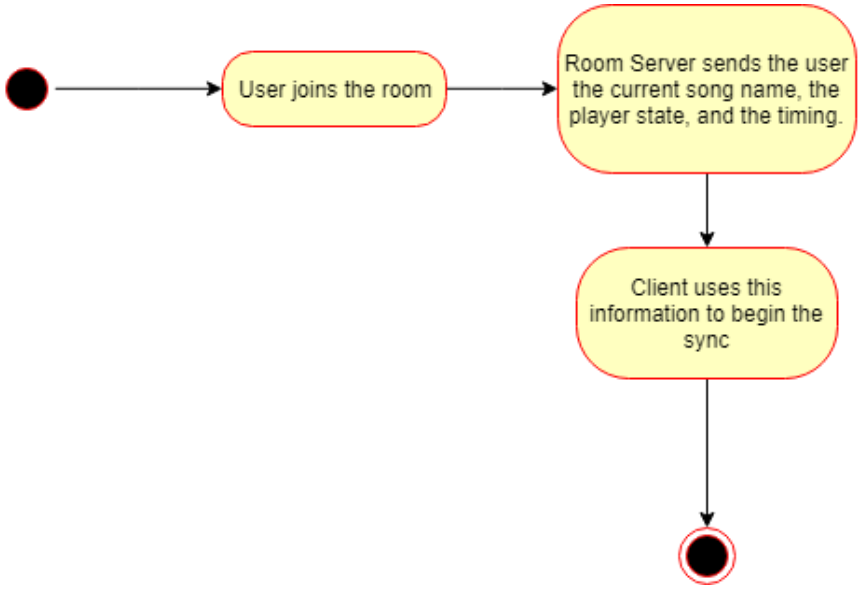
- **Playlist Module: Playing State Machine Diagram**



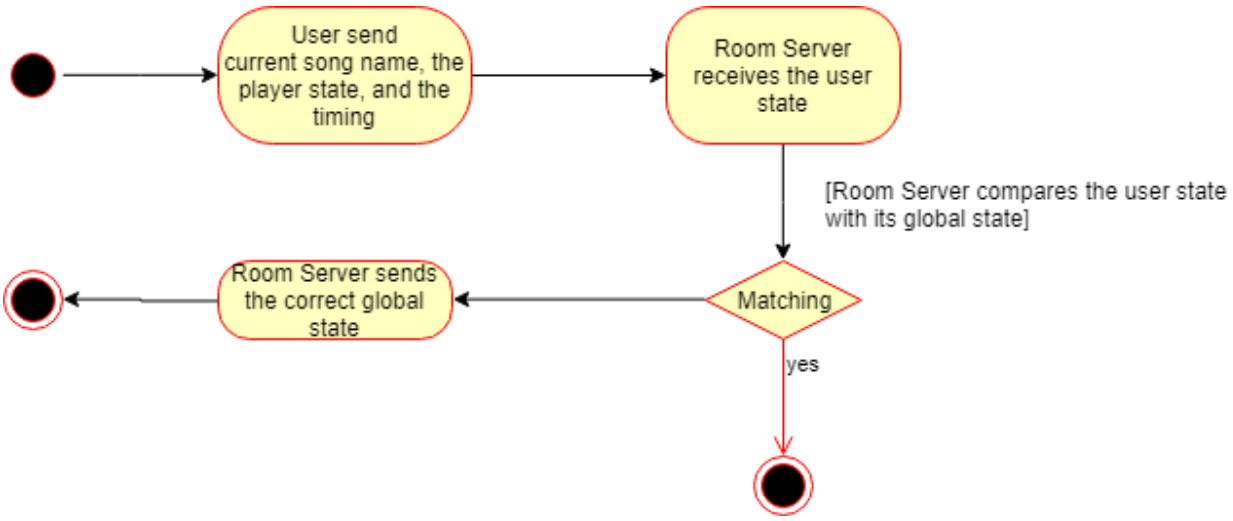
- **Playlist Module: Song Play Request Activity Diagram**



- **Playlist Module: On-Join Activity Diagram**



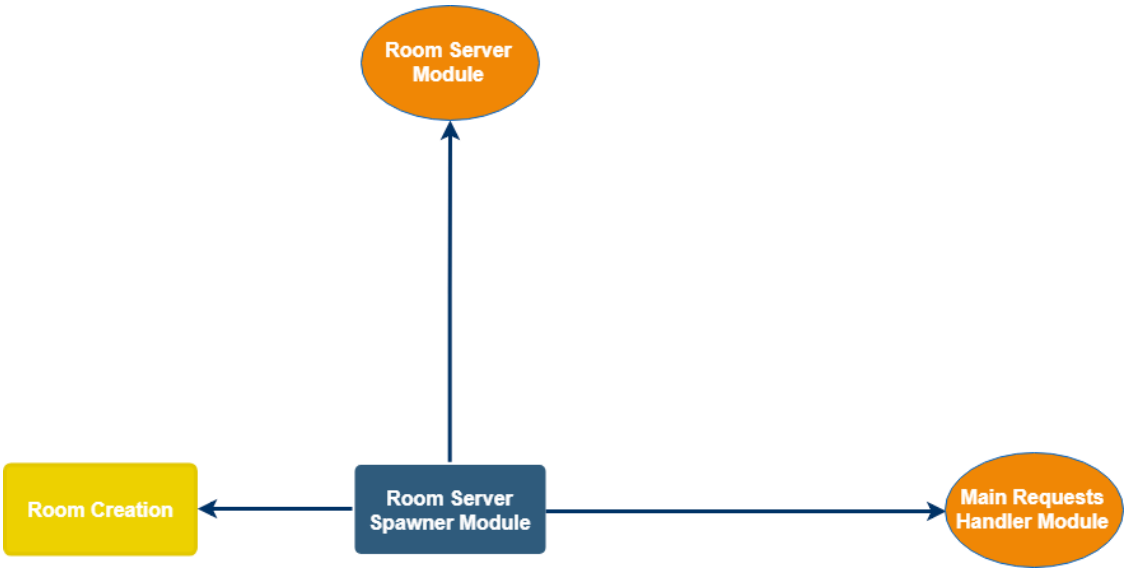
- **Playlist Module: Continuous Sync Activity Diagram**



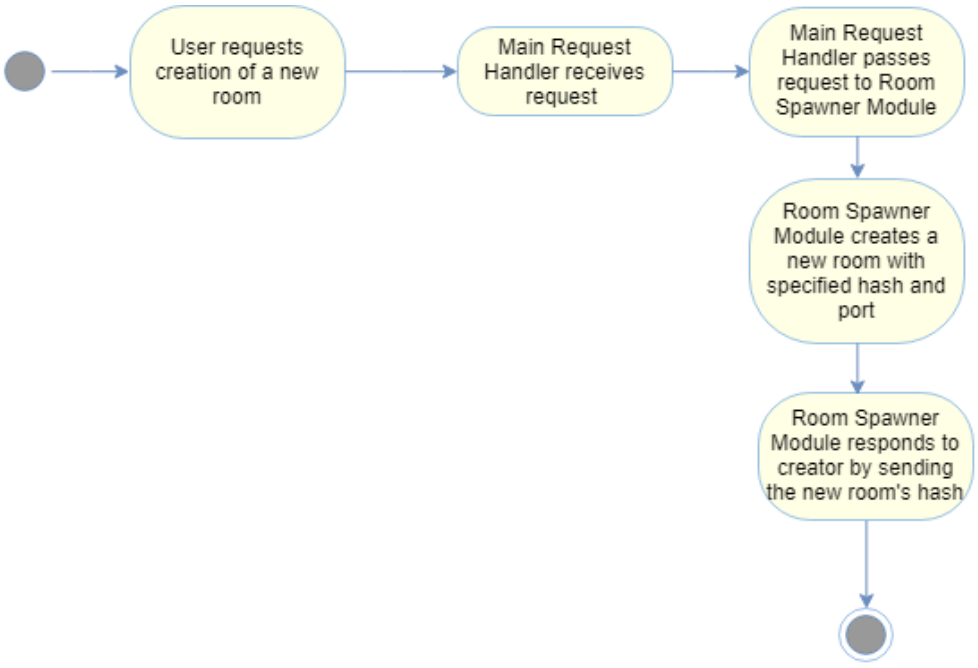
6. Room Server Spawner Module

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.

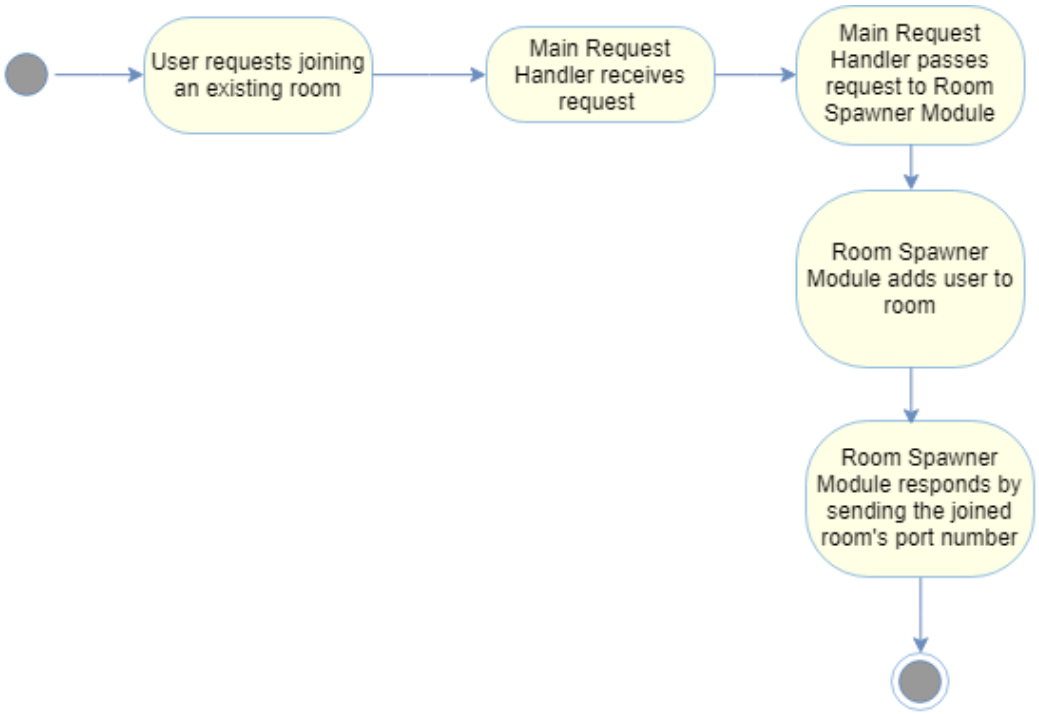
- **Room Server Spawner Module: Context Diagram**



- **Room Server Spawner Module: Room Creation Activity Diagram**

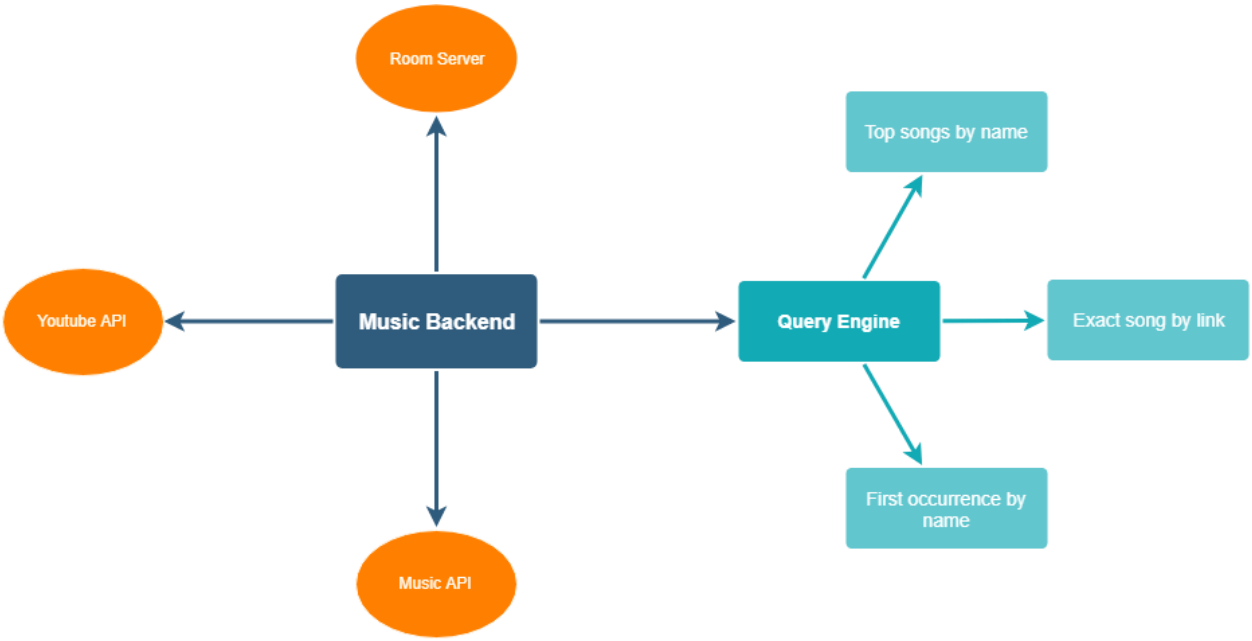


- **Room Server Spawner Module: Room Joining Activity Diagram**

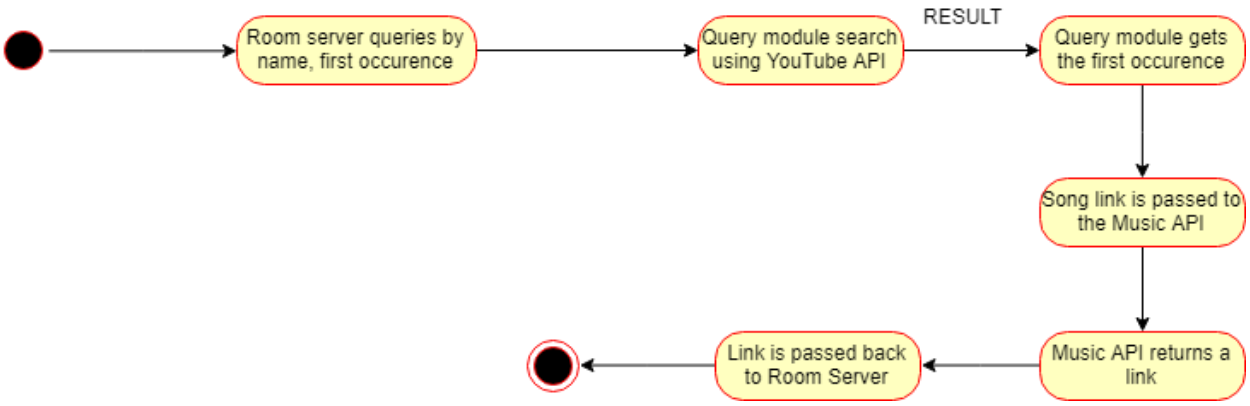


7. Music Backend Module

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 occurence, or even querying by link.



- **Music Backend Module: Basic Song Search Activity Diagram**



8. Mobile App Module

Provide module description.
Use block or context diagram to illustrate external and sub-modules.
Use activity diagram, state machine diagram, data flow diagrams to illustrate module operations.

System Functions

[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

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 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 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. 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_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.

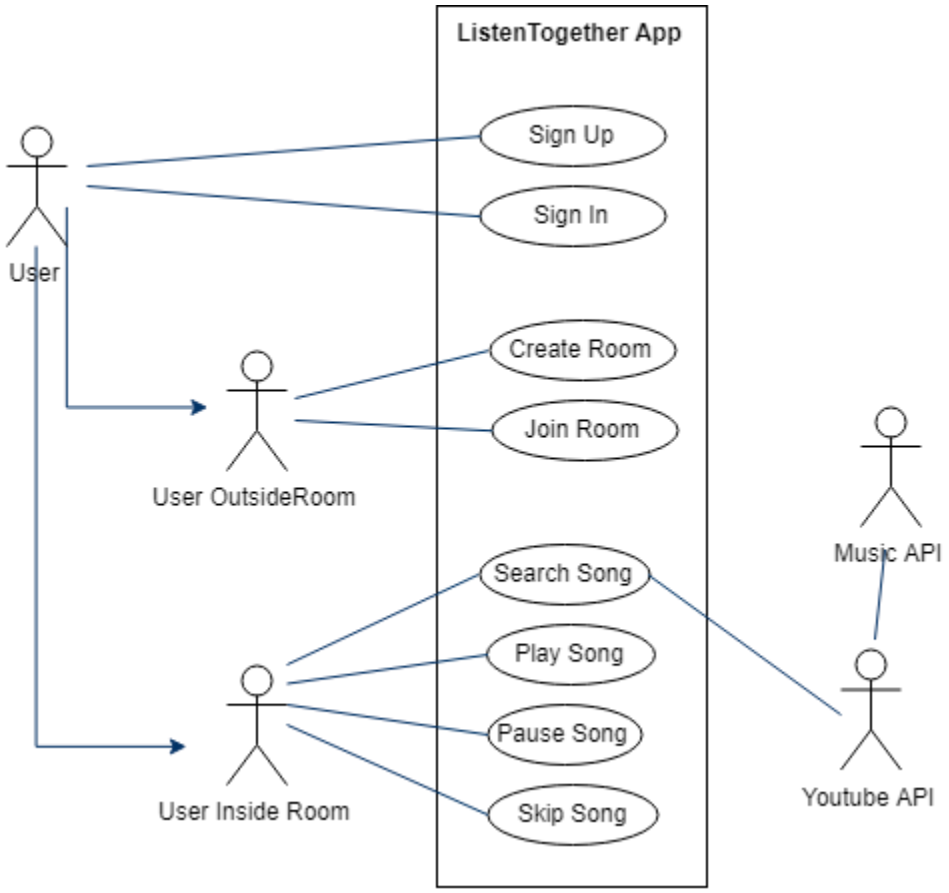
Post-conditions: none.

System Models

<Make only mandatory diagram to illustrate overall system interaction or to explain complex scenarios>

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.



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] 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] 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.

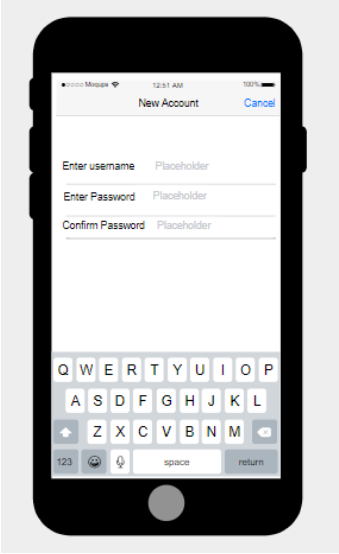
[NFR_P] Signal Strength Requirements

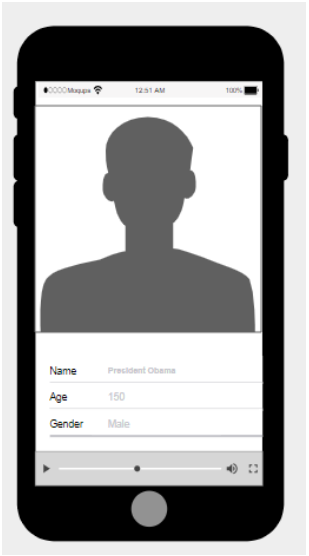
A mobile application is more likely to encounter a network drop or signal strength reduction situation than the desktop version of the application due to the inherently mobile nature of the platform. Some features may not be either network-fault tolerant and might not degrade

System Interfaces

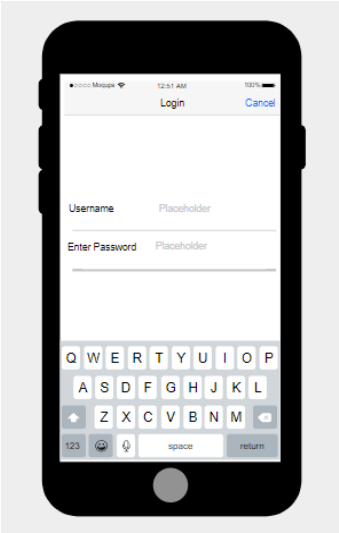
User Interfaces

- **Sign Up**





- **Sign In**


12:51 AM
100%
Login
Cancel
Username
Placeholder
Enter Password
Placeholder
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return