LinkList Software Design Document

1. Introduction

- LinkList is a software product, presented as an Android application, that will allow multiple users to create a collaborative playlist on one device and actively edit it through adding or deleting new songs from the playlist.

- Users may host/join a playlist session and edit the playlist while music is being streamed. Host users will have the right to control who is in the session, but everyone will be able to choose songs and vote on the current playlist.

- LinkList will not store music, remember a user's song list (past session life), or let a user play their own downloaded music.

2. Design Considerations

- We will need to address the issue of choosing the intended song when multiple songs of the same title are returned from a query.

- The reliance on Spotify's API, currently still in beta, may present an issue as the software is still new and in the process of debugging.

- Capacity limit of server storage. The number of sessions that can be active is finite based off of the memory limits of the server, there must be a cap set in place once stress testing begins.

2.1 Assumptions

We assume the app will be able to search songs on spotify using a spotify song ID, so that we never need to use the songs until they are streamed directly to the user from spotify, even when it was previously searched and added to a playlist.

2.2 Constraints

Network Connection Speed: The application relies heavily on a consistent and reliable connection to access the internet for both the purpose of streaming music, as well as communicating with the server.

Server Load: In its current presentation the application will only have access to one server. With plans to reduce the load on this server, there will be a limit to the number of user sessions that can be active at any given time. Components not in use will be offset to database on disk and only accessed when needed.

Limitations of Spotify API: With plans to utilize Spotify's databases through their Android SDK, a major component of the application is left up to the Terms of Agreement set forth by Spotify Inc. If those terms were to change, a major issue could arise.

Spotify API Functionality: In order to search the databases of Spotify, we must use the spotify web wrapper. The web wrapper has no playback functionality so the android SDK must be used

also. This means that two different APIs must be implemented for full design functionality.

UI Dependence: The application is heavily dependent on user control. The UI must work quickly and effectively to ease user commands, such as hosting/joining a session.

2.3 System Environment

The software will have two separate entities; the server and the app.

The LinkList application will be functional on all Android devices running 4.0 and above. It will interact with the server infrastructure of LinkList and with the Spotify databases through their Web wrapper API and Android SDK API. The application will need to service individual local requests from users via the user interface, send and receive information via RPC to/from the server, as well as interact with the Spotify API to gain access to its authentication and streaming services.

The LinkList server application will run on a Google Cloud server machine, via a Debian GNU environment. It will process communication multiple LinkList applications as well as service queries to its local database.

3. Architectural Design

3.1 Overview

Server Classes:

Class #	Class Title	Role
1	AppSocket	Used to allow communication to users' Apps
2	Room List	Array of rooms, empty rooms periodically deleted
3	Room	has a user array and a playlist, and a room ID
4	Playlist	Doubly-linked-list of songs
5	Song	song metadata only: name/duration etc.
6	UserList	array of all users who have authenticated
7	User	object with Identification data

UI/App Classes:

Class #	Class Title(.class)	Role
1	Login(Splash/Spotify)	This will be the part of the gui that integrates with spotify's api and lets users log in.
2	Main(Main)	The next step is choosing if you are hosting the room or playing music in a hosted room.
3	HostRoom(HostConnect/HostPage)	This will be the master list of songs to be played, and will receive data from players in the room
4	PlayerRoom(PlayPage,PlayConnect,Player/SearchAdapter/SearchPager/SearchPresenter/Search)	The player room will be based around searching for music and sending music choices to the host.
5	Music(MusicControler[sic]/PlayerService)	uses both sockets to coordinate music internally
6	SpotifySocket(OnLoggedIn/Client)	establishes communication with spotify
7	ServerSocket(Client)	establishes communication with LinkList server
8	Speaker()	media player which outputs streamed content
9	List()	Linked list structure to hold and control playlist
10	Room()	Holds playlist and current users
11	Server()	Executes RPC methods established in other classes based on client calls

3.2 Rationale

This architecture causes minimal server strain, allowing cheaper and wider implementation and expansion. Most strain is forced onto spotify servers.

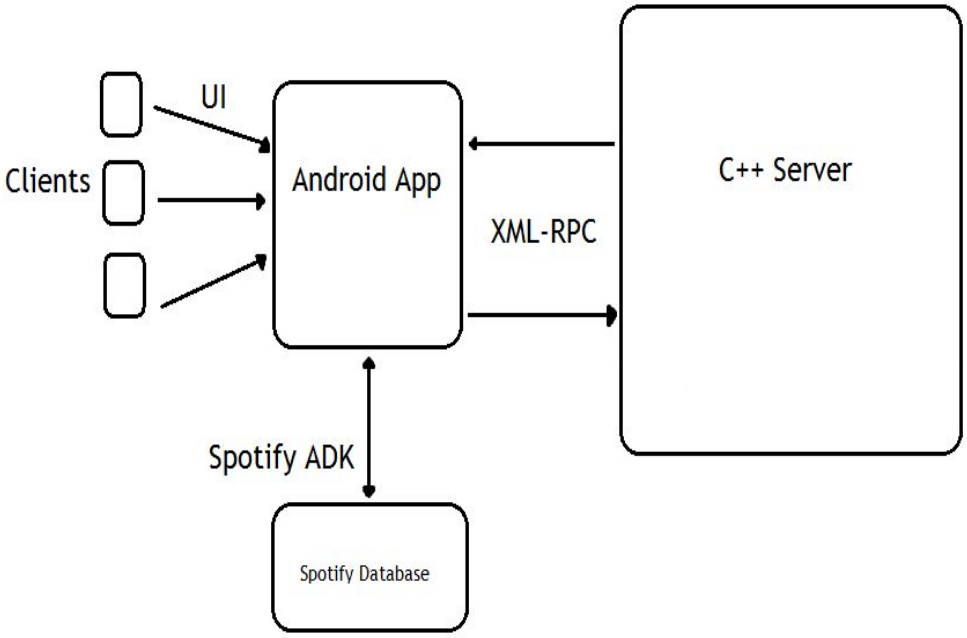
This design also gives users ease of access and full creativity in how they use the app.

3.3 Conceptual (or Logical) View

UI/Device: The first thing users will have to do is log in, once they have logged into a spotify account, they will be given the choice to host or play. When they pick “Play”, the user will be redirected to the PlayerRoom where they will be able to search for and select music to be played through the host. When they select “Host”, users will be redirected to the HostRoom where they will control an updating queue of songs to be played.

Server: The server will handle communications between all parties. Players will send the server Spotify’s ID of the song they wish to add to the database, and the ID will be pulled by the host when it is time to play a song. Spotify interaction will not be handled by the server.

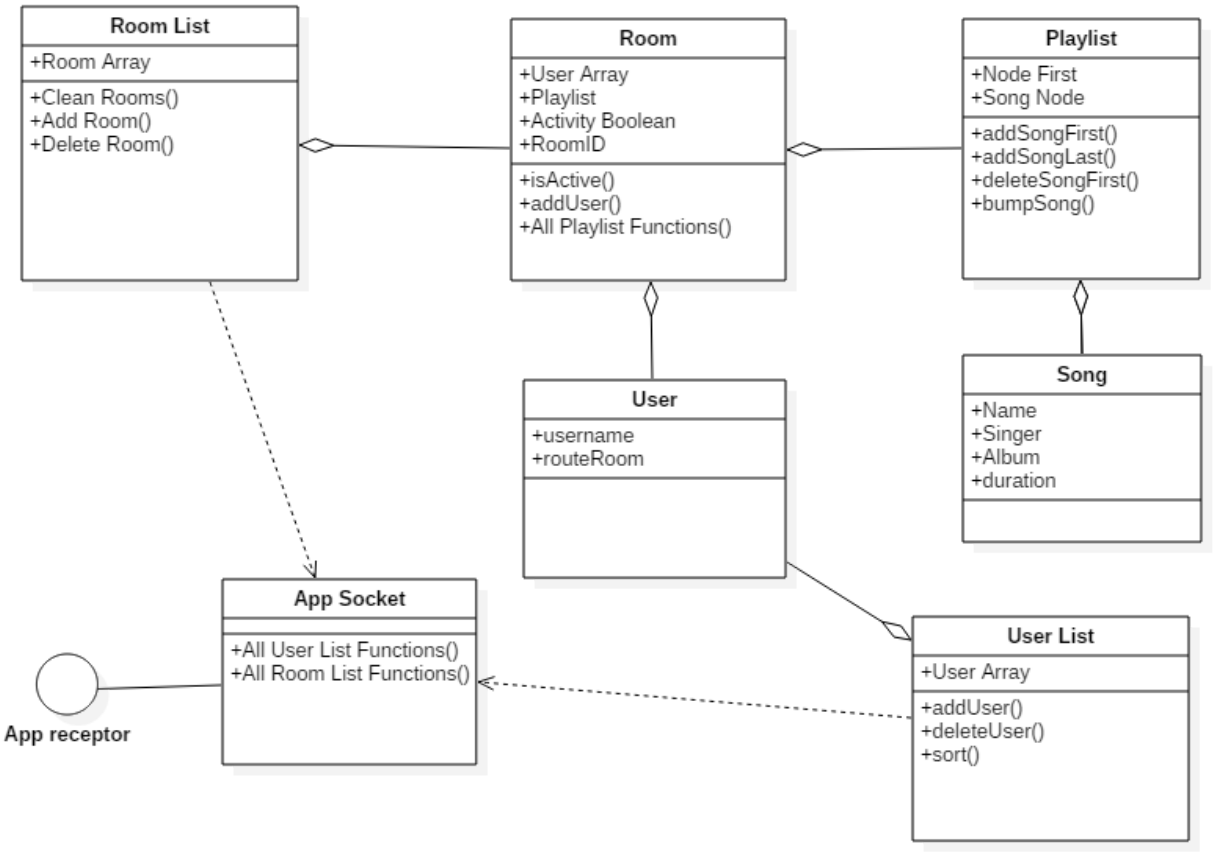
Database: The database will be separated by rooms, and within each room there will be information on who is in the Lobby, and what songs are in the queue.



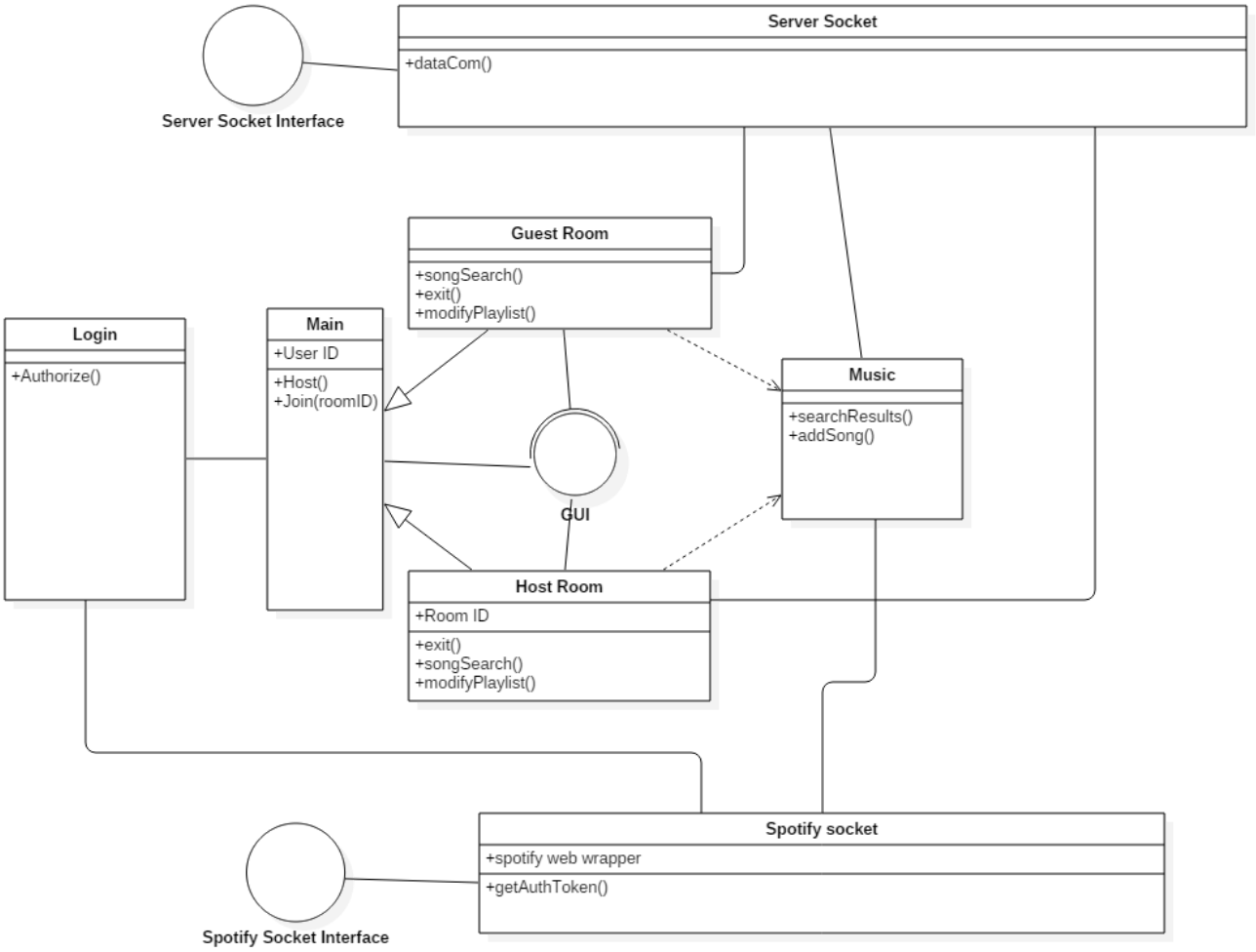
4. Low Level Design

4.1 Class Diagram

Server Class Layout

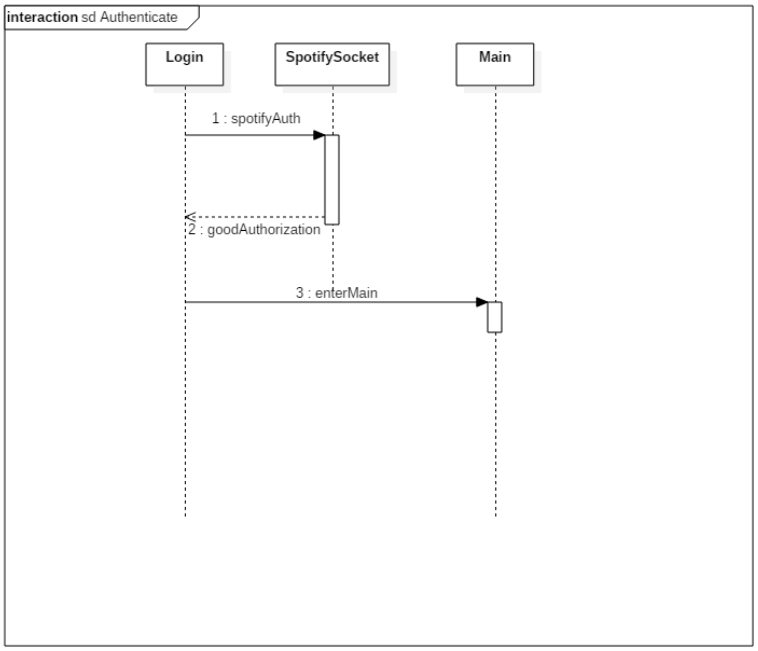


App Class Layout

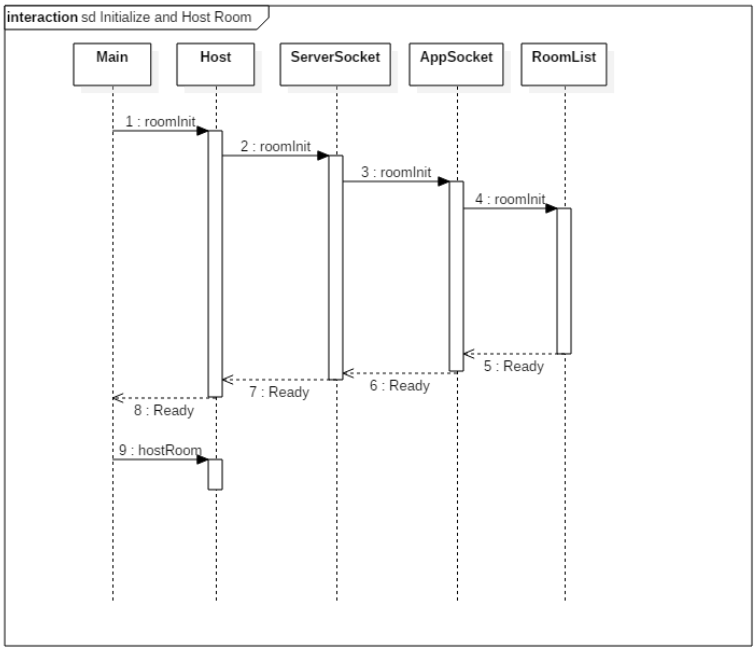


4.2 Sequence Diagrams

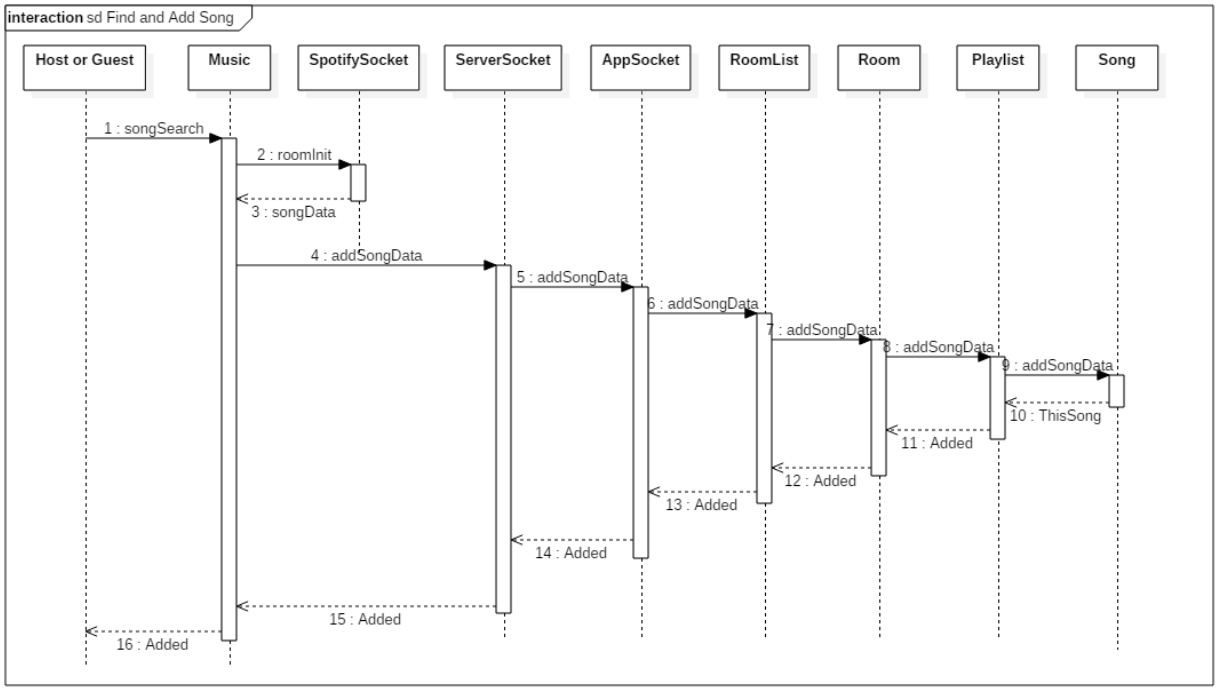
Login:



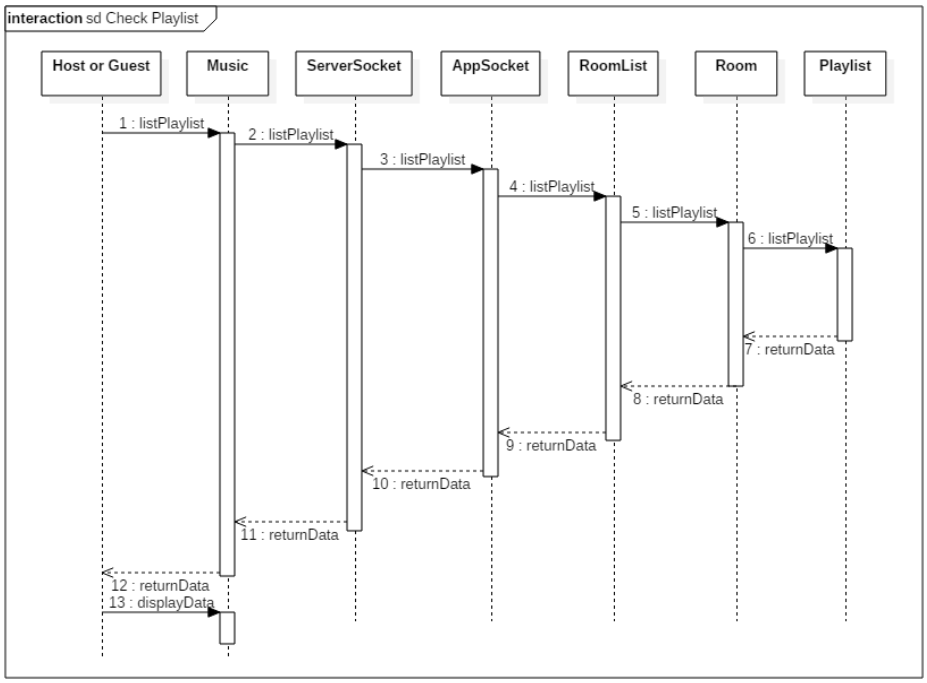
Host room:



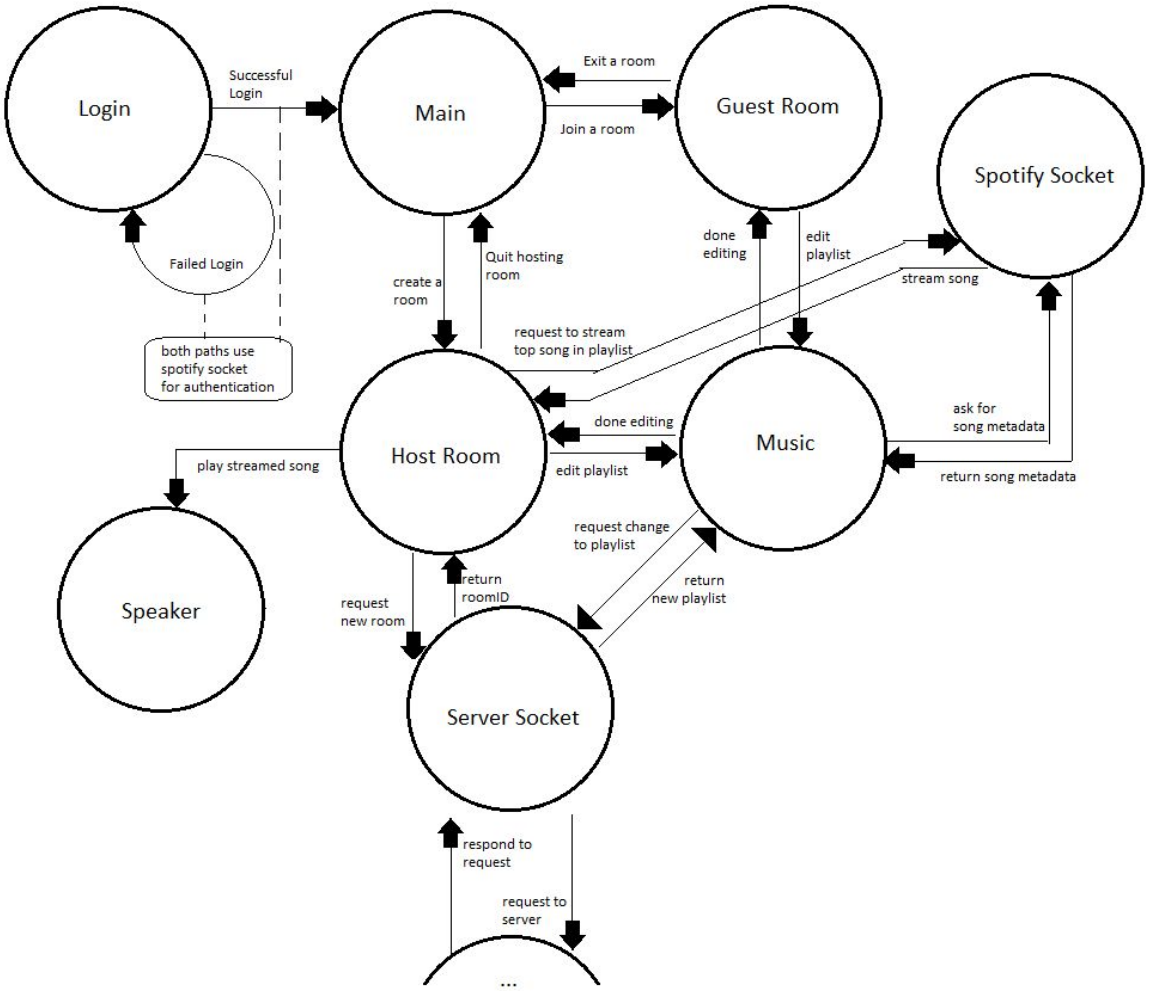
Join as Guest:Add A Song:



Look at Playlist:

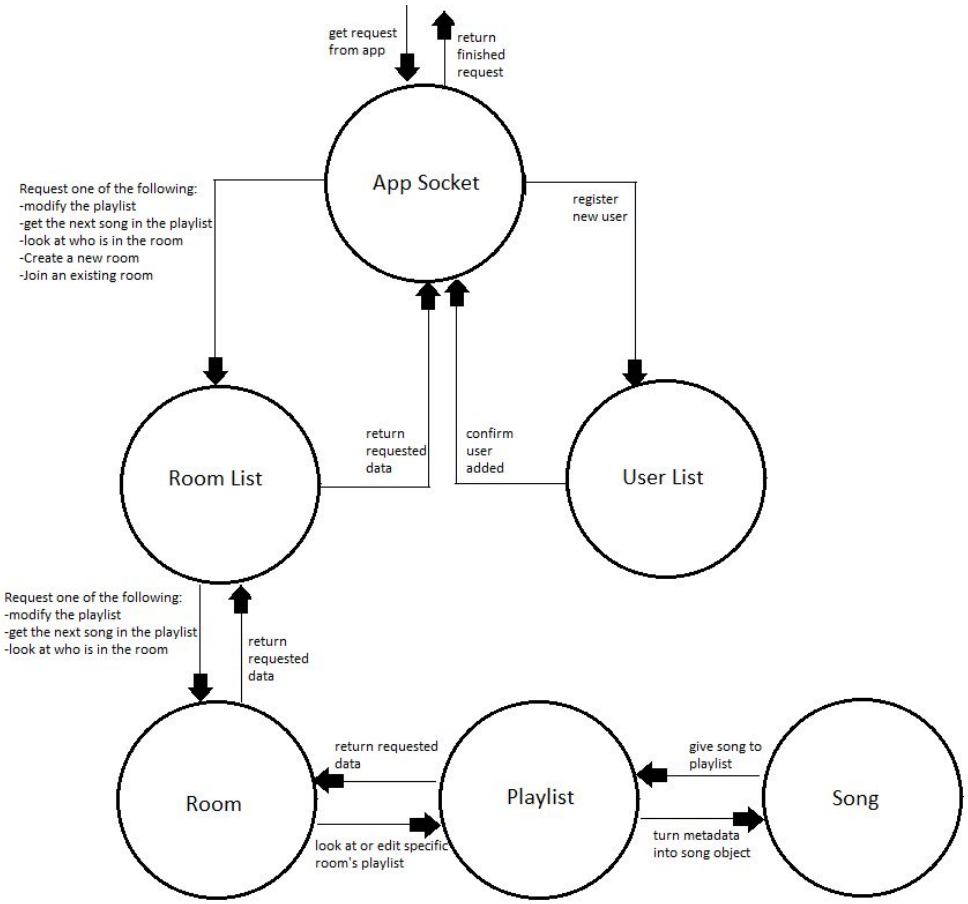


4.3 State Diagram

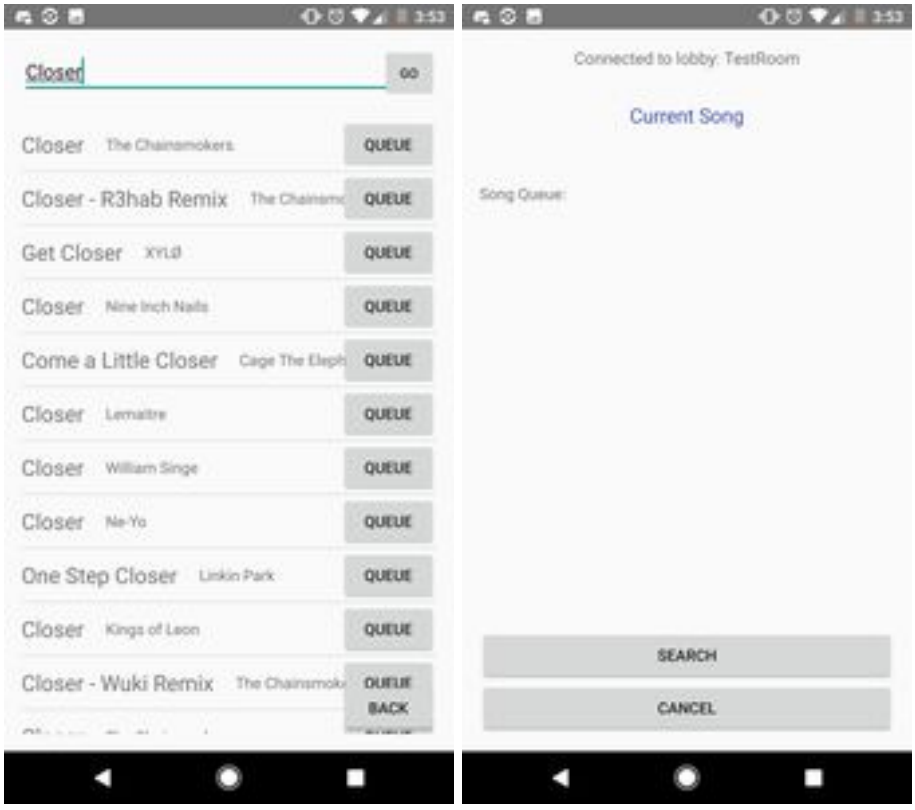


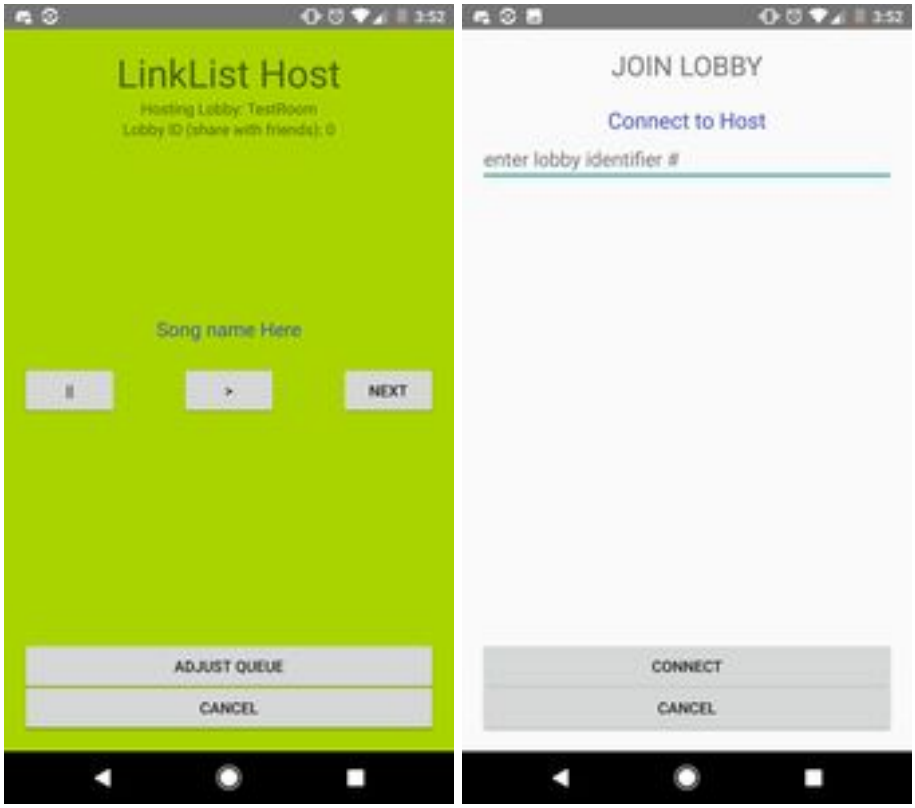
Continued in Server State Diagram below.

Server State diagram:



5. User Interface Design




LinkList Host
Hosting Lobby: TestRoom
Lobby ID (share with friends): 0
Song name Here
||
>
NEXT
ADJUST QUEUE
CANCEL
JOIN LOBBY
Connect to Host
enter lobby identifier #
CONNECT
CANCEL


JOIN
HOST
Connection Success