JUKED

Design Presentation

Francisco Rovelo, Laura Phillips, John Mills, Noah Patton, M Rahman

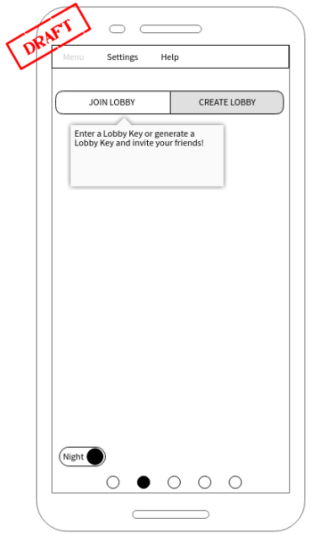
Problem Definition

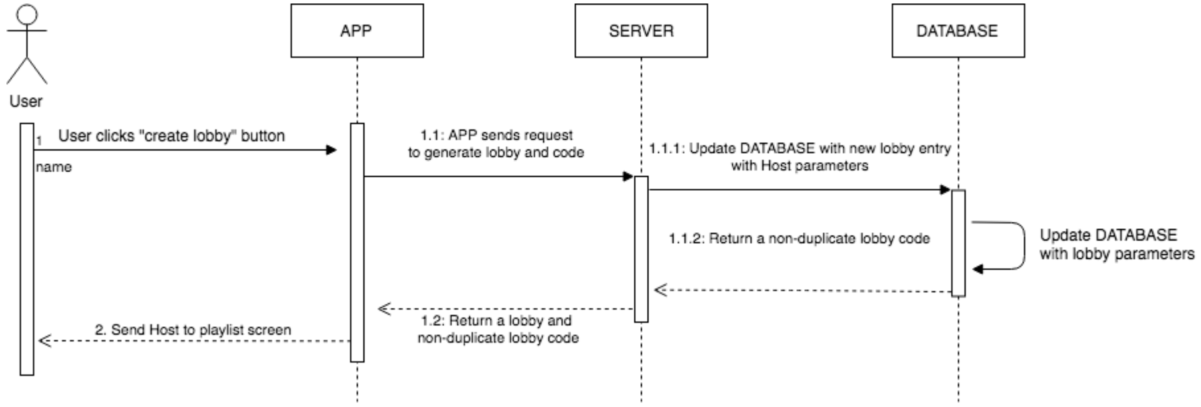
- Most parties have one dj that is the dictator of the music.
- Juked gives power to the people allowing for guests to control the music.
- Let the guests search, pick, and choose songs to be played at the party.
- Allows for upvotes to move a song up in the queue and be played sooner.
- Allows for downvotes to remove a song entirely, rendering that one guy who plays barbie girl over and over powerless.

Updated Scope

- The scope has been degraded to simply a Spotify implementation.
- However, for modularity we have considered using platform objects (i.e., Spotify Object)
  - This object will contain all necessary methods and classes for that object.
    - Spotify Song URL
    - Spotify Song URI
    - Spotify Wrapper Functions
- This was done for the purpose of time. If we have time to implement other services, it will be easier to add this way.

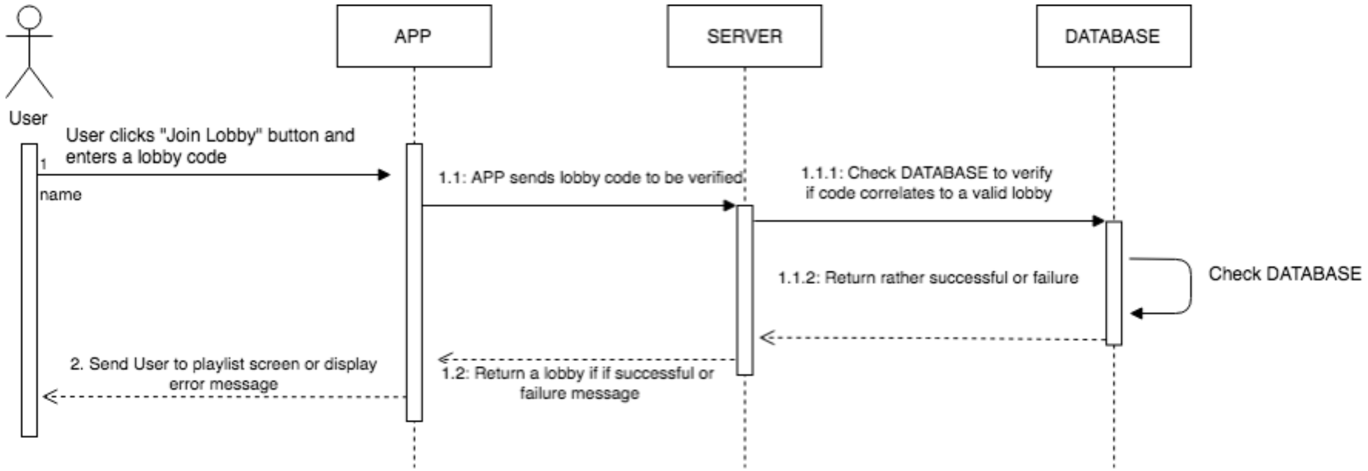
Host Creates A Lobby


DRAFT
Menu
Settings
Help
JOIN LOBBY
CREATE LOBBY
Enter a Lobby Key or generate a Lobby Key and invite your friends!
Night


User
APP
SERVER
DATABASE
1 User clicks "create lobby" button
name
1.1: APP sends request to generate lobby and code
1.1.1: Update DATABASE with new lobby entry with Host parameters
Update DATABASE with lobby parameters
1.1.2: Return a non-duplicate lobby code
1.2: Return a lobby and non-duplicate lobby code
2. Send Host to playlist screen

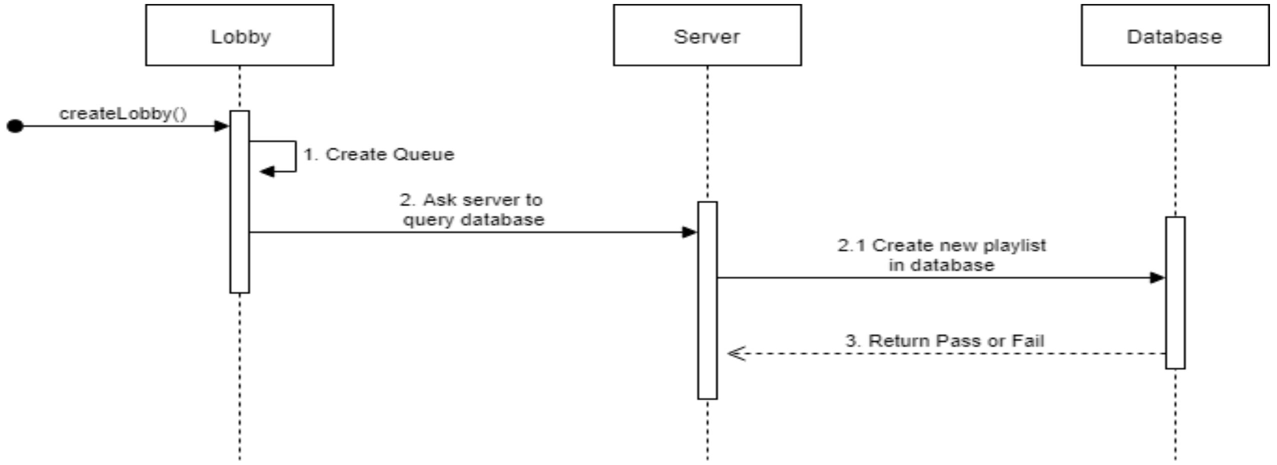
- The host will click “Create Lobby” which in turn sends a request to create a lobby/playlist and return the lobby as well as a unique lobby entry code.

User Joins A Lobby


APP
SERVER
DATABASE
User
User clicks "Join Lobby" button and enters a lobby code
1
name
1.1: APP sends lobby code to be verified
1.1.1: Check DATABASE to verify if code correlates to a valid lobby
Check DATABASE
1.1.2: Return rather successful or failure
1.2: Return a lobby if if successful or failure message
2. Send User to playlist screen or display error message

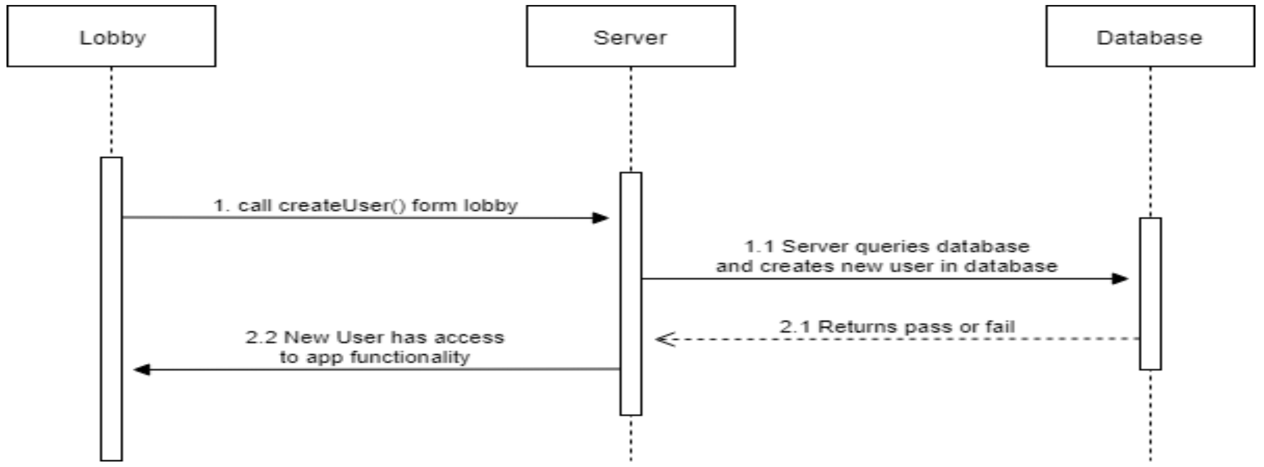
- When a User attempts to join a lobby, they will enter a code which is sent to the DATABASE for validation. If such a lobby exist, it returns success and sends the user to the lobby. Otherwise, error message.

Creation of Playlist



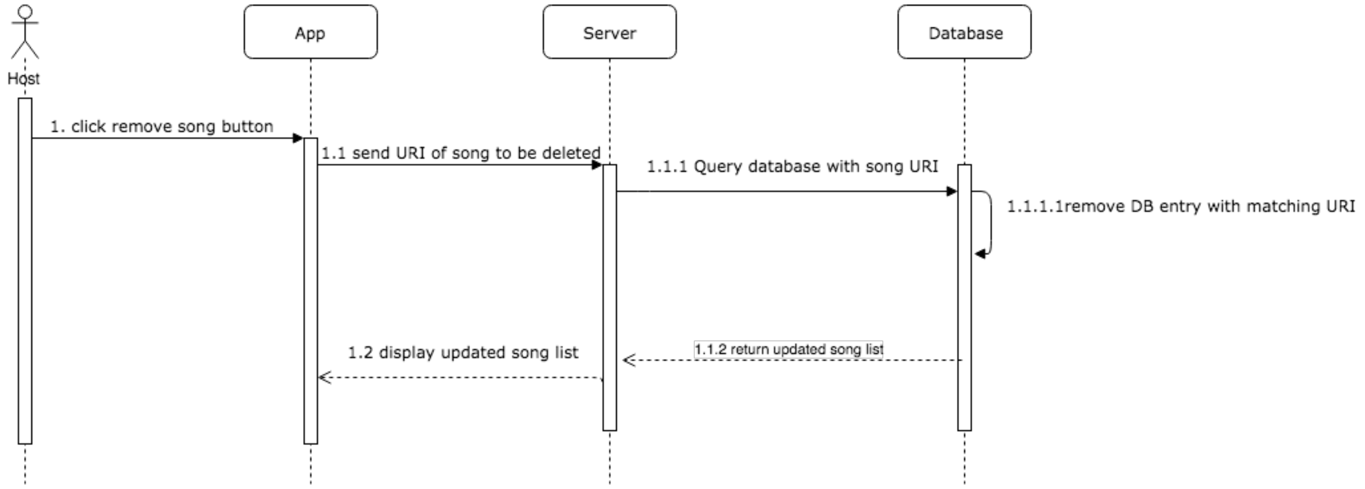
- When a new lobby is created, a playlist is automatically created. An ArrayList is created in the program, then the server queries the database. A new playlist table will then be created.

Creation of User


Lobby
Server
Database
1. call createUser() form lobby
1.1 Server queries database and creates new user in database
2.1 Returns pass or fail
2.2 New User has access to app functionality

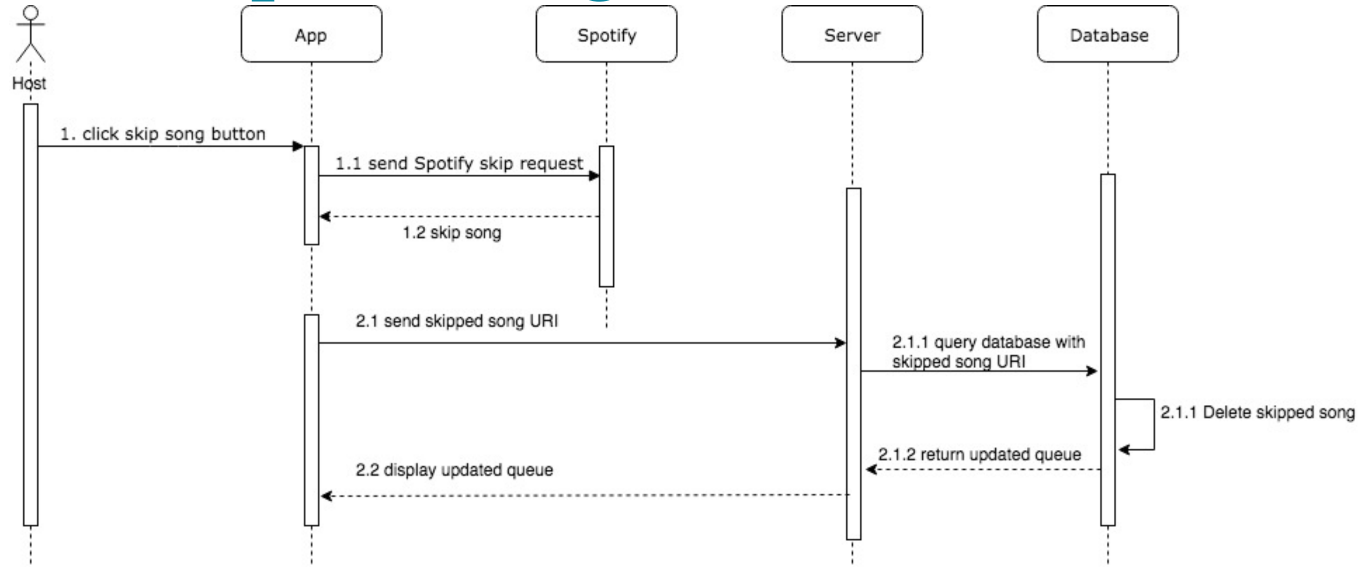
- A user object is created when a user joins a lobby. The method createUser is sent to the server with all of the information that is input from join lobby. The server queries the database and adds the new user.

Host Removes a Song


Host
App
Server
Database
1. click remove song button
1.1 send URI of song to be deleted
1.1.1 Query database with song URI
1.1.1.1remove DB entry with matching URI
1.1.2 return updated song list
1.2 display updated song list

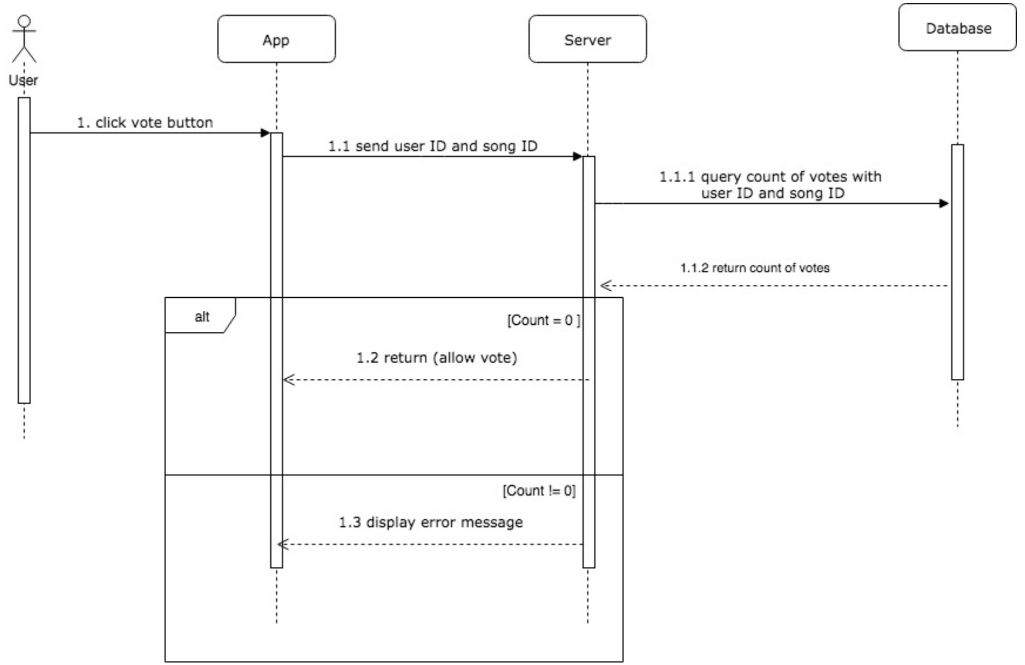
- When a Host removes a song from the queue, the app will contact the server, and the server will contact the database with the song identifier. The database will remove the song from the table of queued songs, and the updated queue will be returned and displayed to the app screen

Host Skips a Song


App
Spotify
Server
Database
Host
1. click skip song button
1.1 send Spotify skip request
1.2 skip song
2.1 send skipped song URI
2.1.1 query database with skipped song URI
2.1.1 Delete skipped song
2.1.2 return updated queue
2.2 display updated queue

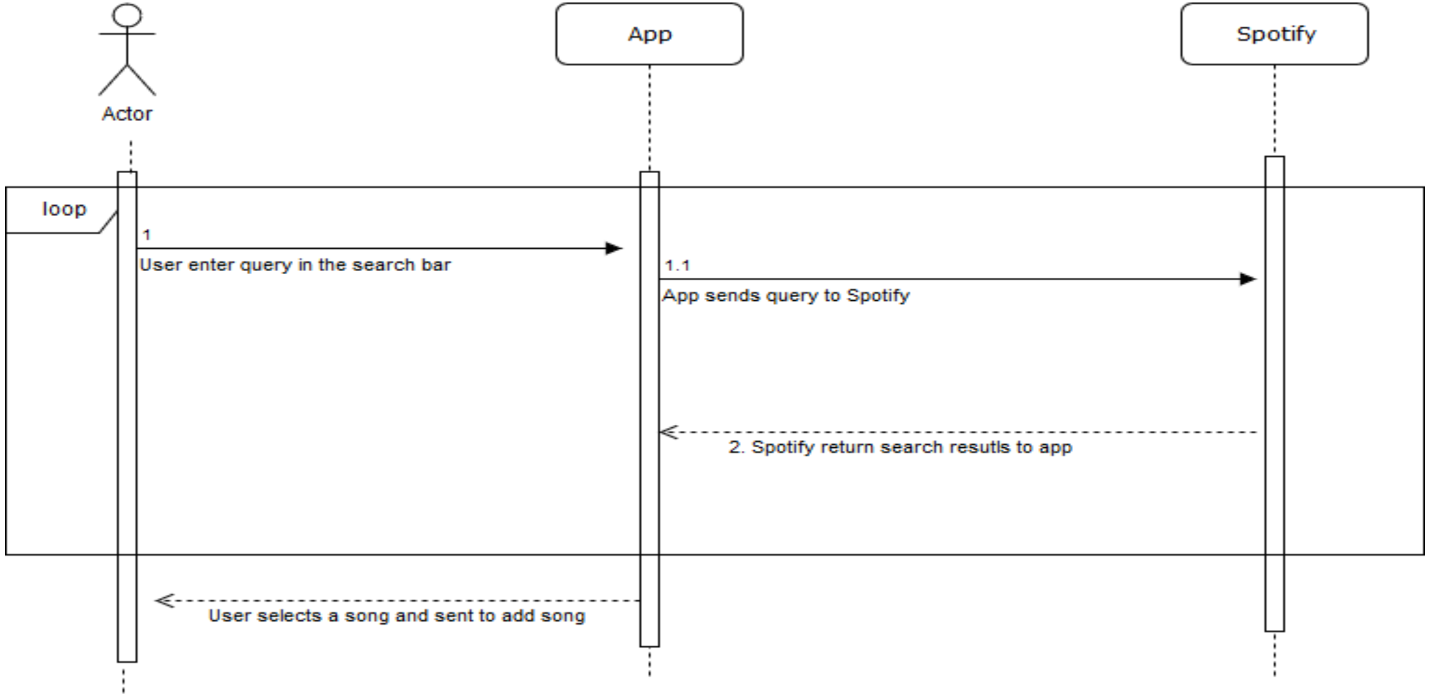
- If the host wants to skip the song that is currently playing, the app will contact the Spotify to skip to the next song. The app will then contact the server with the skipped song's identifier. The database entry matching the skipped song will be deleted, and the updated queue will be displayed

Check If User Has Voted


User
App
Server
Database
1. click vote button
1.1 send user ID and song ID
1.1.1 query count of votes with user ID and song ID
1.1.2 return count of votes
alt
[Count = 0]
1.2 return (allow vote)
[Count != 0]
1.3 display error message

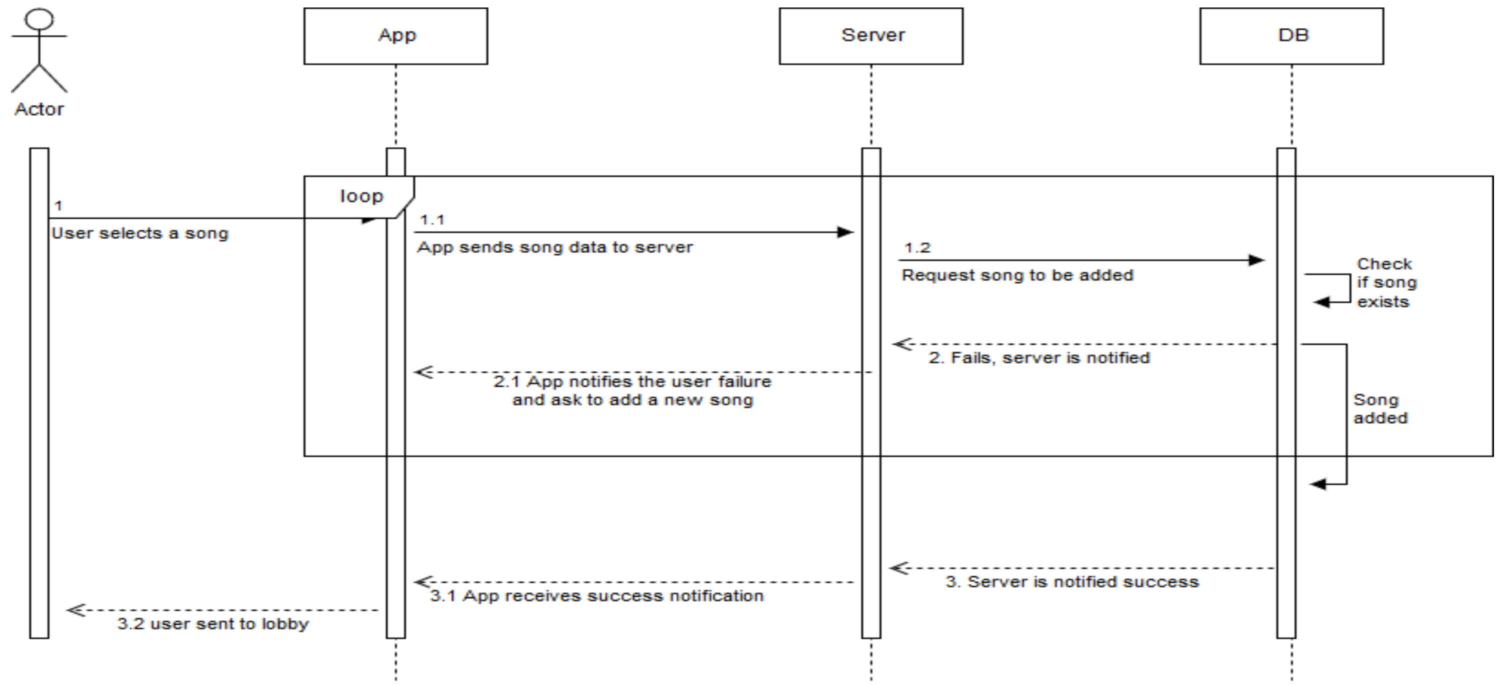
- When a user tries to vote on a song, the app will send the user's ID and the song ID to the server, which will contact the database. If there is a vote matching the user AND song ID, then an error message is displayed. Otherwise, the vote is allowed.

Search for a song


Actor
App
Spotify
loop
1
User enter query in the search bar
1.1
App sends query to Spotify
2. Spotify return search resutls to app
User selects a song and sent to add song

When user search for a song, the passes the query to spotify, returns the result to app. User than have option to choose, discard or redo.

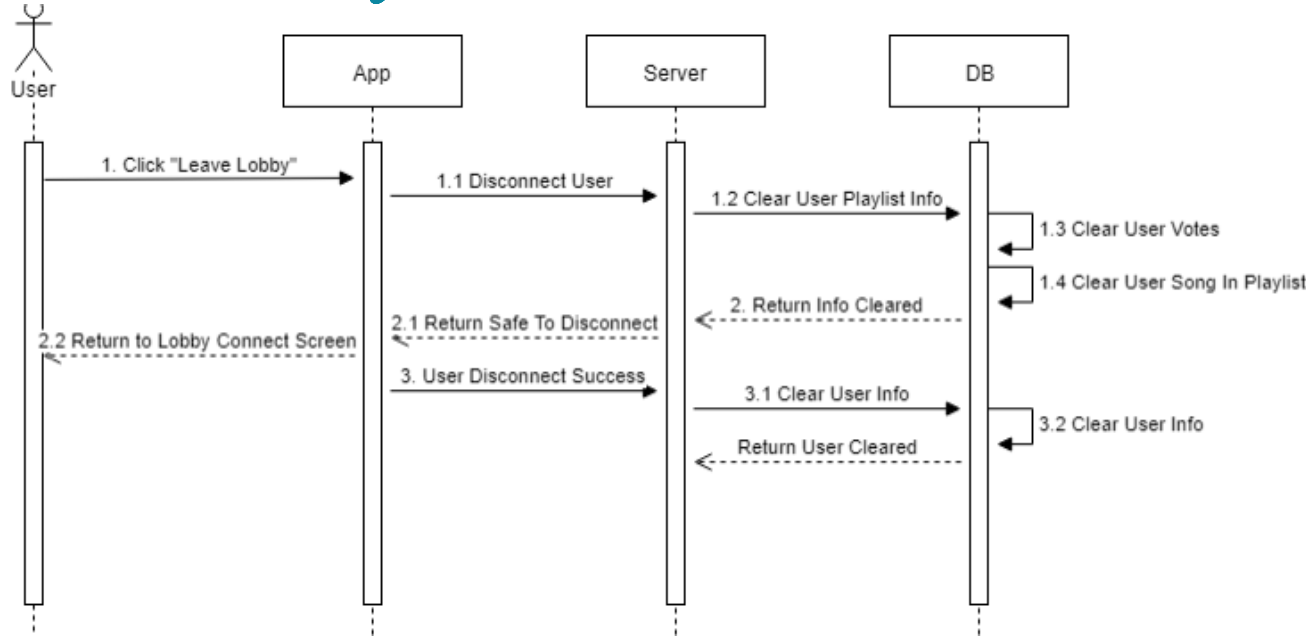
Add a song



Actor
App
Server
DB
loop
1
User selects a song
1.1
App sends song data to server
1.2
Request song to be added
Check if song exists
2. Fails, server is notified
2.1 App notifies the user failure and ask to add a new song
Song added
3. Server is notified success
3.1 App receives success notification
3.2 user sent to lobby

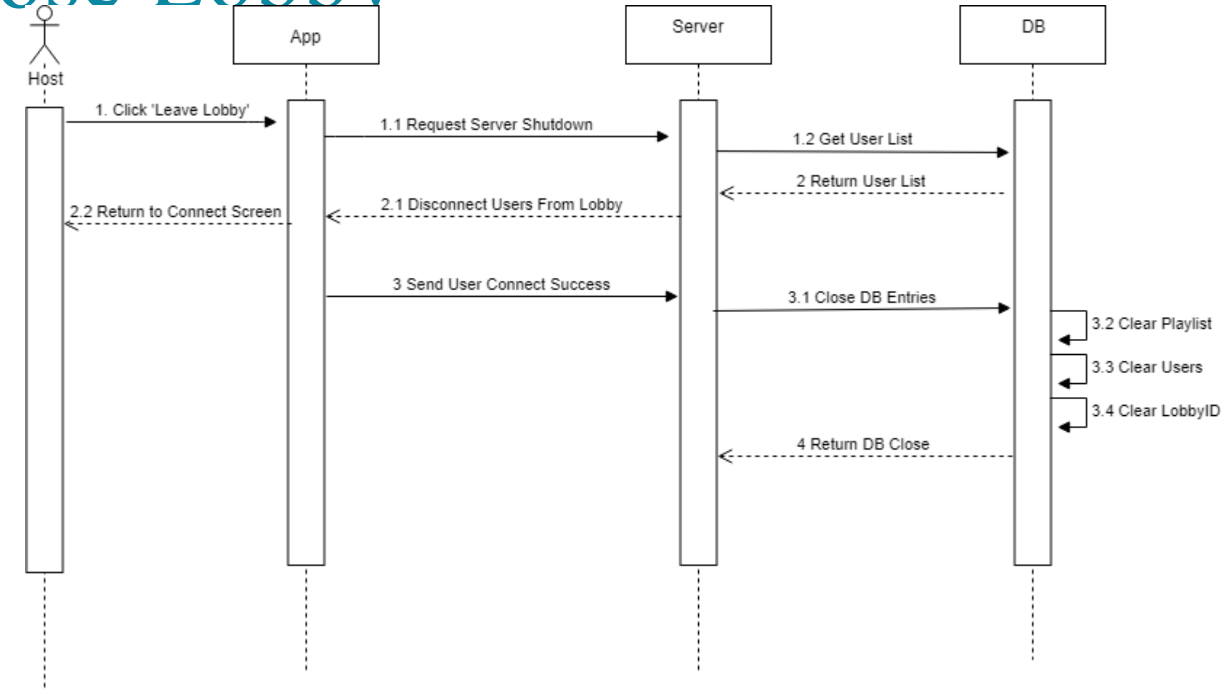
User selects a song, the database gets updated and user is sent to lobby or requested to select another song.

Leave Lobby



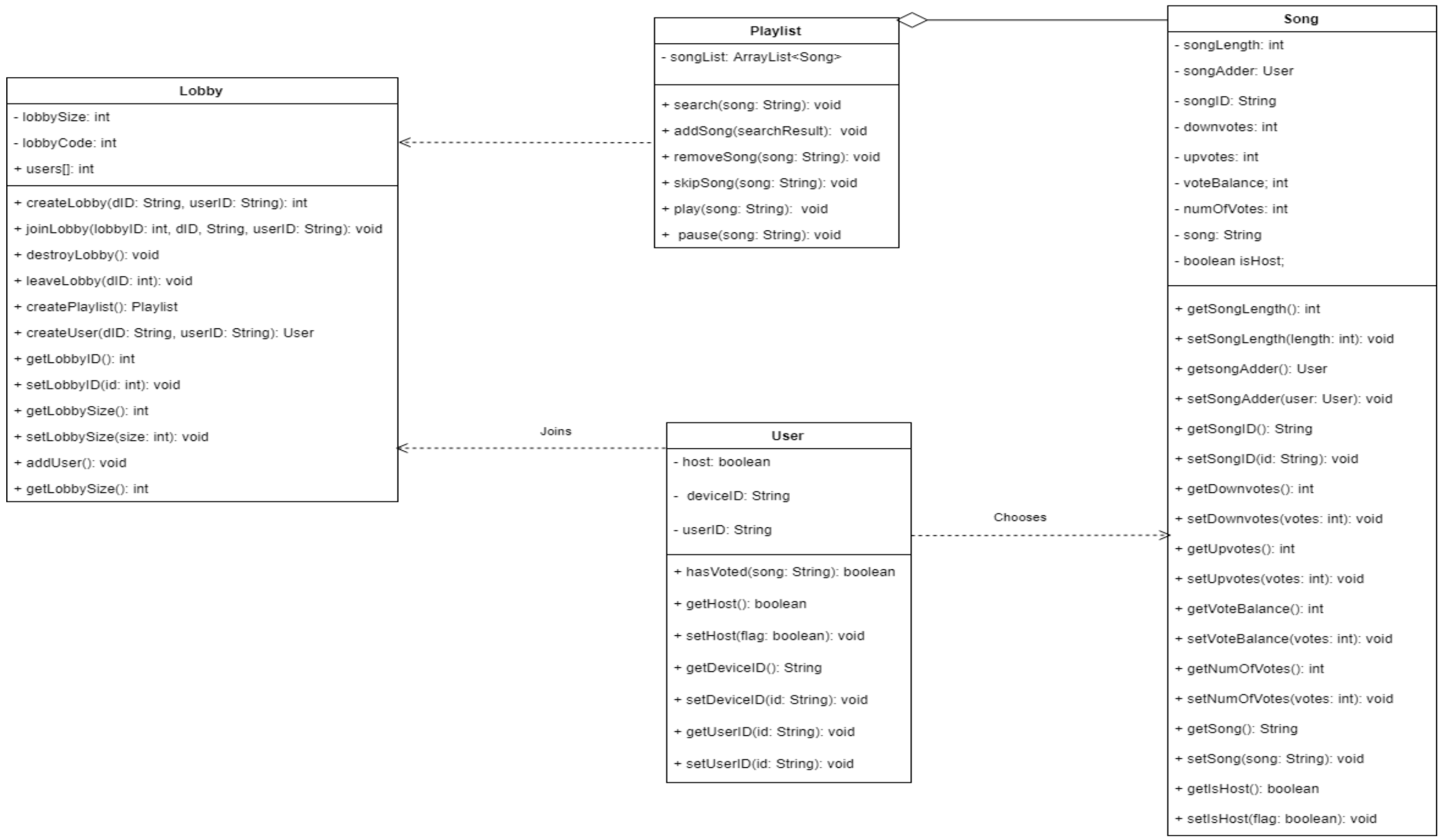
When user clicks leave lobby, the app passes the request to the server, which queries the database, then returns to the server to disconnect the app before fully clearing the database.

Close Lobby



When host clicks leave lobby, the app passes the request to the server, which gathers user list from DB to disconnect users from lobby and then close/clear the DB entries.

Class Diagram


Playlist
- songList: ArrayList<Song>
+ search(song: String): void
+ addSong(searchResult): void
+ removeSong(song: String): void
+ skipSong(song: String): void
+ play(song: String): void
+ pause(song: String): void
Song
- songLength: int
- songAdder: User
- songID: String
- downvotes: int
- upvotes: int
- voteBalance; int
- numOfVotes: int
- song: String
- boolean isHost;
+ getSongLength(): int
+ setSongLength(length: int): void
+ getsongAdder(): User
+ setSongAdder(user: User): void
+ getSongID(): String
+ setSongID(id: String): void
+ getDownvotes(): int
+ setDownvotes(votes: int): void
+ getUpvotes(): int
+ setUpvotes(votes: int): void
+ getVoteBalance(): int
+ setVoteBalance(votes: int): void
+ getNumOfVotes(): int
+ setNumOfVotes(votes: int): void
+ getSong(): String
+ setSong(song: String): void
+ getIsHost(): boolean
+ setIsHost(flag: boolean): void
Lobby
- lobbySize: int
- lobbyCode: int
+ users[]: int
+ createLobby(dID: String, userID: String): int
+ joinLobby(lobbyID: int, dID, String, userID: String): void
+ destroyLobby(): void
+ leaveLobby(dID: int): void
+ createPlaylist(): Playlist
+ createUser(dID: String, userID: String): User
+ getLobbyID(): int
+ setLobbyID(id: int): void
+ getLobbySize(): int
+ setLobbySize(size: int): void
+ addUser(): void
+ getLobbySize(): int
Joins
User
- host: boolean
- deviceID: String
- userID: String
+ hasVoted(song: String): boolean
+ getHost(): boolean
+ setHost(flag: boolean): void
+ getDeviceID(): String
+ setDeviceID(id: String): void
+ getUserID(id: String): void
+ setUserID(id: String): void
Chooses

Present Priority

- Best to have less option but better performance- just focus on Spotify, with the options open for SoundCloud and Youtube in the future.
- Get a working app, then each member picks an area to smooth out the rough edges.
- A little bit everyday gets more done in a week then trying to code the whole day.
- Pick a time everyday for team members to meet and collaborate.
- Consistency.
- Time management.
- We are at a stopping point to make the best app possible (we will never have an app then) and work on the present best scenario.

Summary

- Confident about the design of the application and the backend.
- Confidence in our ability to get a working product soon and add polish later.
- Energized and excited about the product, something we will actually use ourselves