Thomas Weaver
Jay Doshi
Bryce Check
Can Chen
Dayu Jiang
Professor Miller
CSCI201
28 October 2018

Detailed Design Document

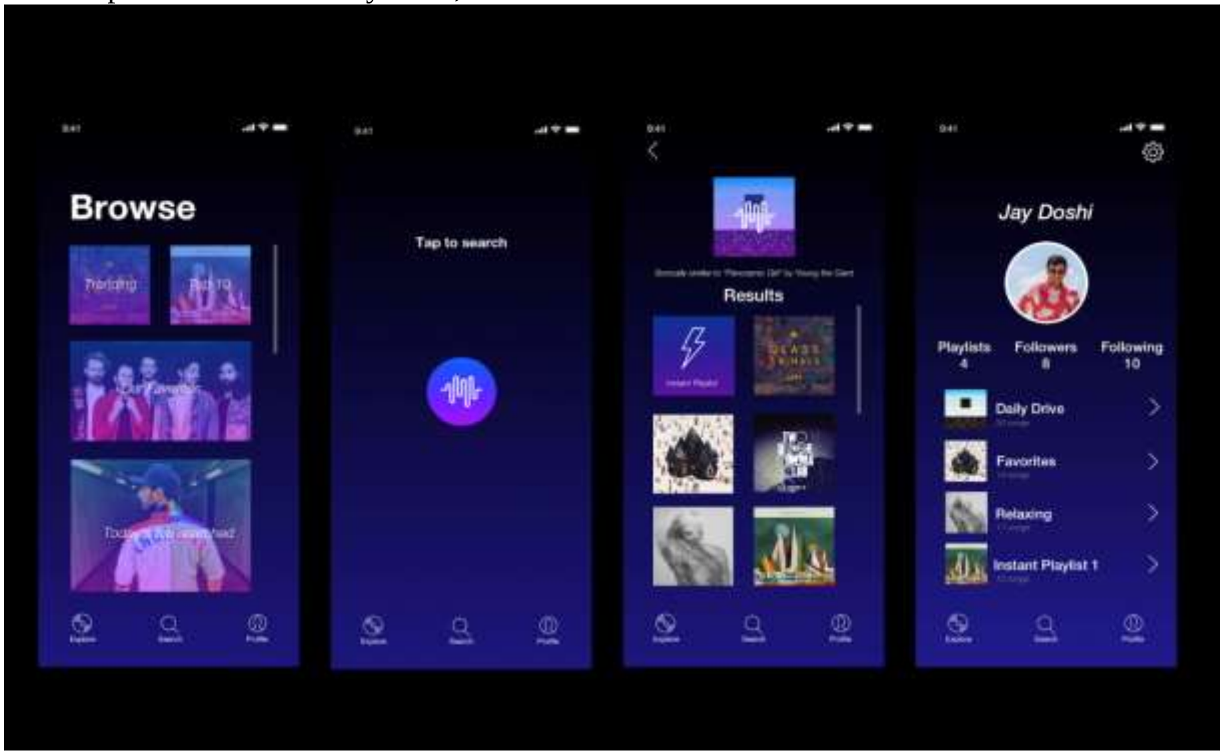
This document is organized starting at the front end of our mobile app and ending at the back end design.

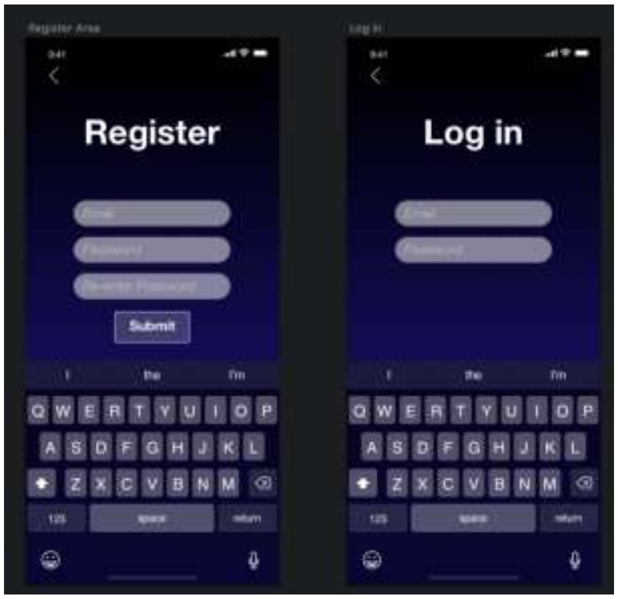
The Front End

General:

An overview of the front end where hardware and testing will be done on an iPhone X and the app development language will be Swift and the design done on Sketch on macOS.

Will be updated with full storyboard, for now these are the main views:





To build the user interface of our app, we are planning to use a tab bar application. There will be three buttons on our tab bar including an Explore, Search, and Profile tab. Users can quickly switch between tabs by accessing them on the bottom row.

Related reference:
https://developer.apple.com/documentation/uikit/uitabbarcontroller
https://developer.apple.com/documentation/quartzcore/cagradientlayer (Background)
https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/ProgrammaticallyCreatingConstraints.html

Currently known way to easily get album art for UI:
Documentation for iTunes is sparse, used Postman to get more info
https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/

The app launches in guest mode, so any user who has not registered will still be able to download the app and access its core features.

Profile:

On the profile page in guest mode, there will be two options, one for users to log in and another for users to register. The registration asks for a user's email and a new password. As a guest, users can only create up to 5 playlists. Logged in users have unlimited playlist creations.

For the profile page, users can follow one another, so they have followers and following. The profile view controller also needs to display a scrollable view of playlists.

https://developer.apple.com/documentation/uikit/uitableview
https://developer.apple.com/documentation/uikit/uiscrollview

UILabels:

1. User name
2. Playlists title
3. Following (and number) title
4. Followers (and number) title

UIButton

1. Add Playlist
2. Delete Playlist

If logged in as Guest:
UIButton

1. Log in / Register

Search:

On the search page, there will be two input fields, one will be the artist name and one will be the song name. We use this as the user's query. The circular button is the search button, when pressed will take the input from those two fields and query our database. The song name and artist name are case sensitive, and they have to be exact for the search to work.

Once the query is made, a new view comes up with the results of the search. This view is only accessed by a search. The search results display the message, "Sonically similar to [song name] by [artist name]:" then an overlay is applied to the album art to show the user that song was the one searched, then the songs are listed below (see diagram)

With these results, the first button is an Instant Playlist button. When pressed, it checks if there is space for a new playlist, then if there is it takes the first 10 songs of the search and creates a playlist. This playlist is added to the user's playlist always as "Instant Playlist #[num]" where num is how many instant playlists that user has made.

Related reference:
https://developer.apple.com/documentation/uikit/uitextfield

UILabel

1. Tap to search title
2. Results title in search results view

UIButton

1. Search button
2. Instant Playlist button

Playlist View:

If you click on a playlist, you are able to see what songs are in it. You can add and remove songs.

UIButton

1. Add song
2. Delete song

Explore:

For now, the explore page will have a list of trending songs, which is based on which songs have been searched the most for within 24 hours (a counter for each song)

UILabel

1. Explore title
2. Trending subtitle

Following MVC (Model-View-Controller):

Sample class structure:

User class

- String firstname, String lastname
- Set of followers
- Set of following
- Int number of playlists made
- Set of Playlists
- Profile picture

Guest is-a User

- String name = “Guest”
- No follower/following
- Set of Playlists (max size 5)
- Default profile picture

Playlist class

- Vector of Songs (songs should be able to be added twice or more)

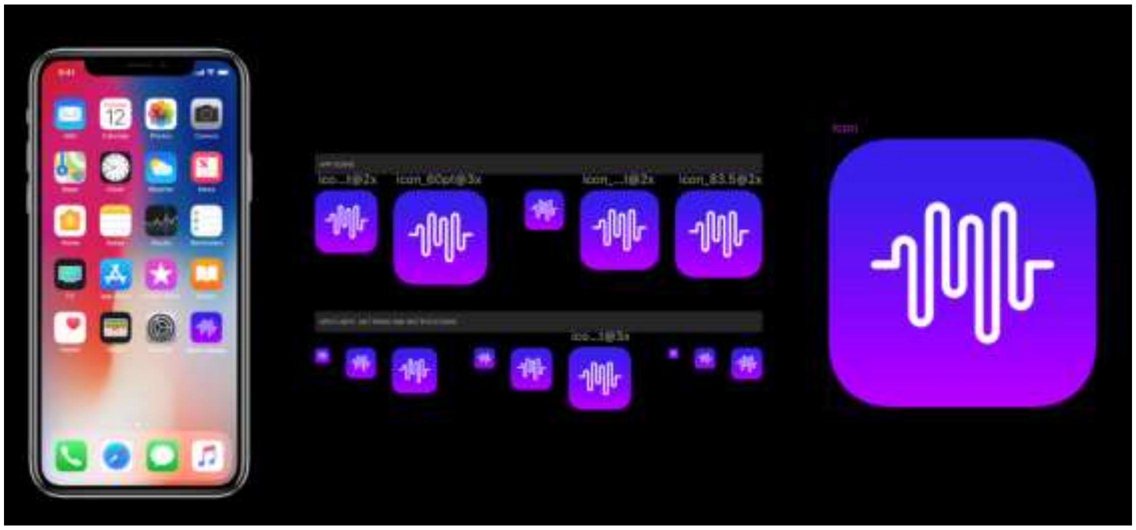
Song class

- String artist name
- String song name
- MP3 file
- Related data stored in this class

View controller needed:

Enter the app on the Search tab. One Swift file to control all trivial view controls including the background layer, gradient settings, color settings, UIStatusBarStyle, then that one method is always called in the viewDidLoad() of every view controller.

SearchViewController.swift
SearchResultsViewController.swift
ProfileViewController.swift
ExploreViewController.swift
PlaylistViewController.swift
LogInViewController.swift
RegisterViewController.swift



App Icon assets

Launch screen asset

<u>Back End</u>

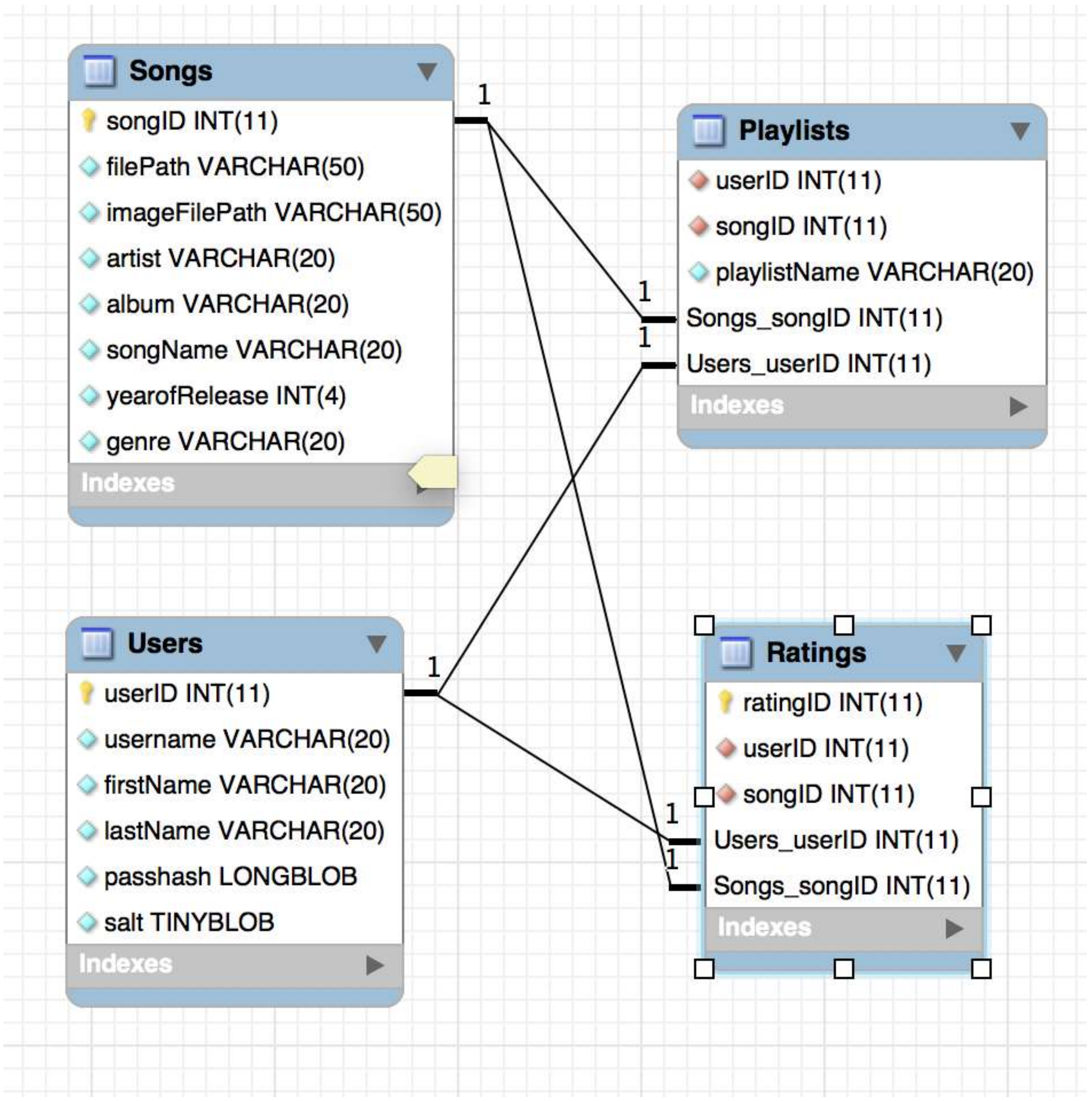
Databases:

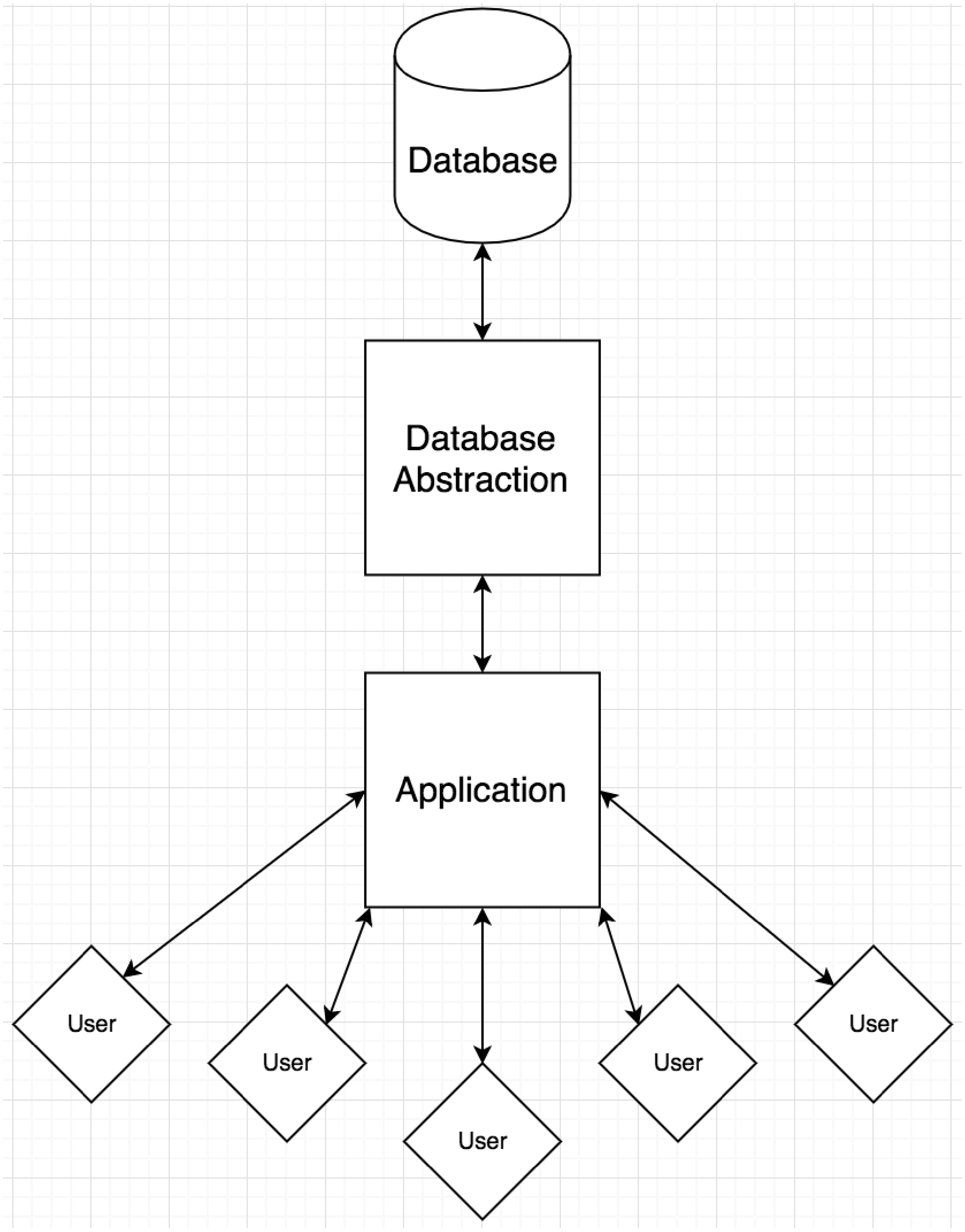
We are going to use MySQL for the database technology, and use the appropriate jdbc connector to connect our java code to MySQL. The App's database will have 4 tables. The first two tables are the Songs and Users table. The songs table will have all the meta information about the song for displaying information and for use in possible clustering or other algorithms for ranking similarity. We are storing filepaths to images as well as the songs themselves. This is because databases are not meant to store such large pieces of data. Instead, songs and images will be saved in a certain folder on the web server, the path to which will be stored in the database.

The users table will store the username, first and last names, the password hash and the salt. We may add in email if we deem it necessary. The salt will be what is combined with the password sent from the user to form the password hash. This is to ensure multilayer security and to take the liability of storing plain-text passwords on our database. Upon successful authentication, our application will generate a JWT (JSON web token) which will be passed sent to the user and passed from the user to authenticate future requests while the JWT is valid.

The playlists and the ratings table will be used to help form suggestions for songs that the user may like. The user will just add a song to a playlist which will be generated if the playlist name is unlike any that the user has already made.

There will be a database abstraction class that the backend team will be able to use in order to get information from the database in the form of plain old Java Objects which will be easy to manipulate and write back to the database. The abstraction class will also be used to manage the connection pool, performing user authentication, and querying information to train and build suggestion models for end users.




Database
Database Abstraction
Application
User
User
User
User
User

Radio Playlist Generation and Music Comparison:

[Change from original][Most of this paragraph was changed]

Next is the Fast Fourier Transform music comparison and playlist building on the back end. To build a playlist of songs that our radio station music app is going to output to the user for them to listen too, the fast Fourier transform will be used to accomplish this as well as the Mel Frequency cepstrum coefficient algorithm and the dynamic time warping algorithm. We will start with a database of about 150 songs and this will be what our app will use. The songs will be stored in a MySQL Database that has a table of songs with different columns that will hold the different aspects of the songs we will use. The songs will be stored in their MP3 format and then be converted into their raw time data forms to be used by the Fast Fourier transform and then a frequency profile will be built for each song. This frequency profile will be then processed in MFC coefficients. Access to the songs in the database will be done through the appropriate JDBC as mentioned above in the database section. To convert the songs we will use the Java Sound API that can accomplish this and then to do the Fourier Transform the JTransform Library will be used. The Java Sound API can be found here:
https://www.oracle.com/technetwork/java/javasounddemo-140014.html
The JTransform is a very fast Fourier Transform and its documentation can be found here:
https://sites.google.com/site/piotrwendykier/software/jtransforms.
This will all be done with java using the class and state diagrams below. Once the same database has a table populated with the song frequency data then we can use that frequency data to compare songs to other songs and find closeness matches.

To build a playlist of songs that sound like the song the user inputs, the Fast Dynamic Time Warp algorithm will be overlayed on the two songs to compare and the input song will be compared against all other songs in the database within its respective genre. Since we do not have access to a commercial music database, genre data will have to entered by hand in the database. The mp3 files used here did not store the genre in metadata as that is only available from commercial versions of the songs. Then the other backend code will take the populated array of song titles which will be the playlist to the front end for the user to play and listen too.

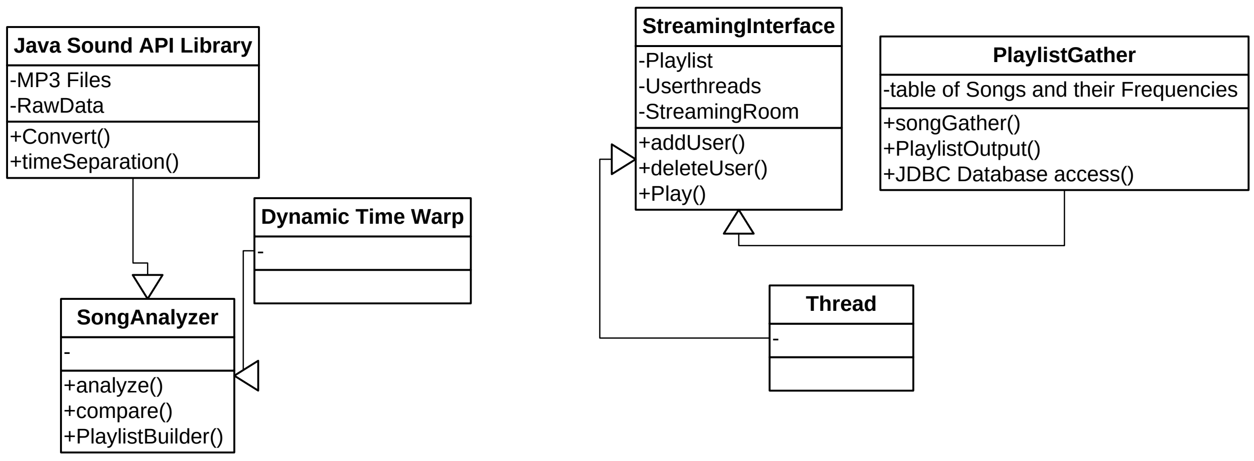
Networking and Backend Communication:

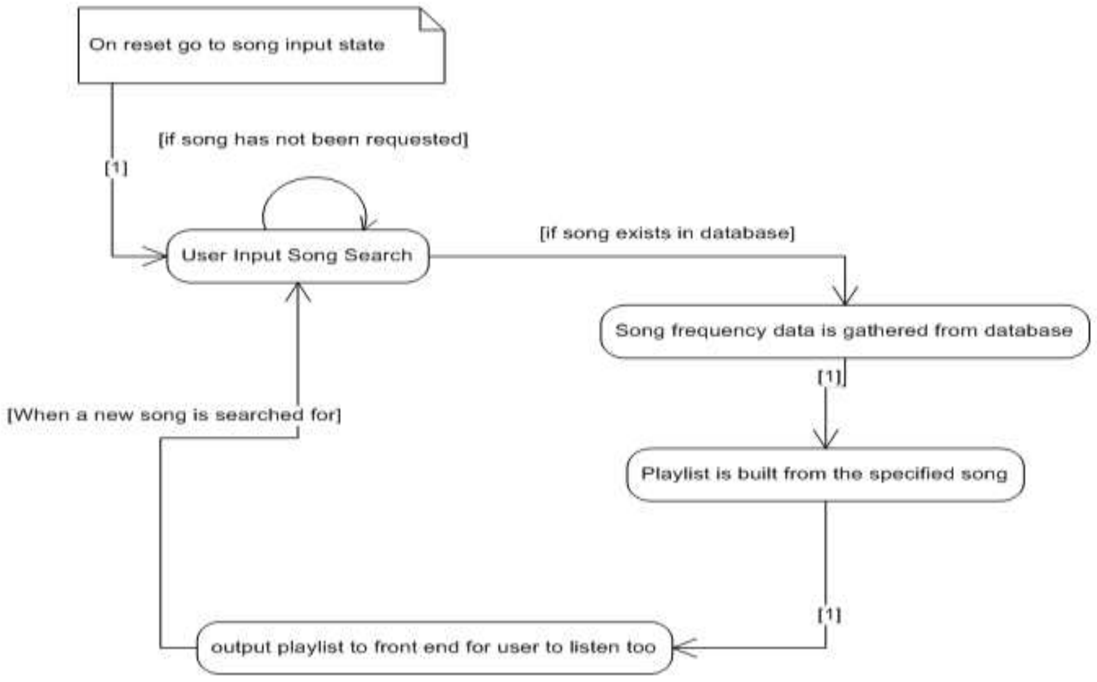
Access to this backend will be done primarily through servlets so that we can utilize the java environment. Also multi-threaded networking capabilities will be built in using network threads and sockets so create the chat environment for users accessing the app as well as allowing multiple users to access the apps at one time. A network will be absolutely needed as well as multi-threading in order to have multiple users streaming music at one time. All of this will also be done in java as well so that the code is compatible.

Below are the high level class diagram for the playlist building through the FFT and MFCC and DTW and a state diagram of the flow of operations in this backend to generate the music lists.

Class Diagram and State Diagram for Playlist Generation and Communication:

[Changed from original]




On reset go to song input state
[1]
[if song has not been requested]
User Input Song Search
[if song exists in database]
Song frequency data is gathered from database
[1]
Playlist is built from the specified song
[When a new song is searched for]
[1]
output playlist to front end for user to listen too