System and Unit Test Report

Product Name: JukeBox
Team Name: Four Eights and a Crown
Date: November 23, 2016

System Test Scenarios

Sprint 1

User Story 1 from sprint 1: As a developer, I should have a UI template so I can quickly generate other pages.

User story 2 from sprint 1: (Decided to change to Firebase halfway through the sprint, so user story needed to change)
Initial Version: As a developer, I should be view database tables and create cloud HTTP calls from Triton.
Updated Version: As a developer, I should be able to view database tables and pull information from them using Firebase

User Story 3 from sprint 1: As a user, I should be able to sign up and log in to Jukebox to access my account data.
User Story 4 from sprint 1: As a user, I should be able to play a song on Jukebox so I can hear my music

1. Start JukeBox app; Select “Log In to Spotify” or “Sign Up for Spotify”
   - Log in with Facebook/Username
     - username/email address = <email@ucsc.edu>
     - Password = <password>
     - Press enter key
   - User should be taken to list of Playlists created in their location (Firebase query)

2. User should be able to click on a playlist, then switch to the Song Search tab and queue songs into that playlist
   - Select tab with magnifying glass icon
   - Enter song name in Search bar
     - Search = <Stairway to Heaven>
   - Click correct song
     - <Stairway to Heaven - Led Zeppelin>
     - If no song is currently playing, this song will start to play
     - Else, song will be added to the playlist’s song queue
3. User should then be able to Currently Playing tab (tab with Play button icon) to see the queue
4. The song will be played when it reaches the front of the queue

Sprint 2

User story 1 from sprint 2: As a user, I want to be able to make a playlist so other users can play songs.

User story 2 from sprint 2: As a user, I want to be to add to a playlist so I can hear my song being played.

1. Start JukeBox app; select 'Add a playlist';type
   - Playlist name=<Test Playlist>
   - Wifi =<Cruznet>
   - Press Submit Key
   - User should be brought back to the list of playlist
2. User should be able to search for songs and add them to their playlist
   - go to search; type
   - click song
   - add to queue
3. Switch to 'Currently Playing' for the playlist 'Test Playlist' to see currently playing song and the next song in the queue
4. Other users on the same wifi should be able to see the playlist 'Test Playlist' and be able to navigate to 'Currently Playing' to see what is currently playing and what is next in the queue
5. If other user wants to hear a song navigate to 'Search';type
   - Click song and it gets added to queue
6. Other User would wait for song to reach front of queue

Sprint 3

User Story 1 from sprint 3: As a user, I want my app to be aesthetically pleasing so I can have a better user experience.

User Story 2 from sprint 3: As a developer, I want the app to be able to make queries through the Firebase database so the app can fetch songs from Spotify.

User Story 3 from sprint 3: As a developer, I want to use an sorting algorithm so the app plays a list of songs in the order that is the most pleasing to the user.

1. Start JukeBox app; be aesthetically pleased; select 'Add a Playlist';type
   - Playlist name=<Test Playlist>
   - Wifi =<Cruznet>
   - Press Submit Key
   - User should be brought back to the list of playlist

2. User can add song to ‘Test Playlist’ by searching for song
    - go to ‘search’; type;
    - click song
3. A User should be able to upvote and downvote a song on a collaborative playlist

Unit Tests:

Tests are located in:

jukebox-cs115/app/src/test/java/com/sideproject/ryanbrounley/jukebox_android/

Module: Playlist - PlaylistTest.java

Tests done on the Playlist object:

1. Create playlist with certain parameters, check to see that all “getters” are working properly (getName(), getID(), getWifi()).
2. Test “setters” (setId())
3. Test that adding Song objects to a Playlist works
    a. Assert that neither songs are in Playlist
    b. Add Song 1 to playlist - test isInPlaylist(Song1) method
    c. Add Song 2 to playlist - test isInPlaylist(Song1) and isInPlaylist(Song2)
4. Add songs in a specific order and test that they are indeed queued in the correct order by checking the Song Uri
5. Test that if a song is removed from the playlist that it will no longer be queued
6. Test Song Veto
    a. Assert that if a song has 2+ downvotes, it is removed from the playlist

Module: Song - SongTest.java

Tests done on the Song object:

1. Created Song object with certain parameters and checked to see that all “getters” for Song object (getName(), getArtists(), getUri(), getAlbum()) returned the correct values.
2. Tested voting functionality by ensuring Song had zero votes upon creation, voting a few times, then making sure getDownvotes() returned the correct value.