**Software Implementation and Testing Document**

**For**

**Group 10**

Version 1.0

**Authors**:

Krenn Intharaksa

Josiah Walters

Aditi Anandjiwala

Jacob Holleman

Philip Reece Sablan

# Programming Languages (5 points)

*List the programming languages used in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).*

* Javascript
  + React
    - Used to build the interface of our web app
    - Will be used throughout the entire app
    - Works very well with the other languages/frameworks we are using (CSS,HTML,Node)
  + Node
    - Will be used to add features such as chat, authentication, and live user data
    - works well in tandem with frameworks like React
* CSS
  + Used to style the interface that is built by React
  + Works very well together with HTML and React
* HTML
  + Used for adding text onto the pages
  + Works very well together with CSS and React
* SQL
  + Used to communicate with our database

# Platforms, APIs, Databases, and other technologies used (5 points)

*List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).*

* MongoDB
  + Our database of choice to hold all of our data
* Spotify API
  + Will be used to search for songs, albums, or artists through built in methods
  + Will be used in combination with Node
  + Allows us to pull all of this data from Spotify’s database

# Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the* ***functional requirements*** *listed in your RD).*

* Tested our search function with various artist and song entries. It functions properly and gives the most relevant songs for each artist based on Spotify’s API.
* Our Sign in button and Blogs button both route to different urls, but the page still includes the search bar and header text. We need to move the bulk of our code out of App.js to fix this.
* The blog posts that are currently hard coded show up properly after clicking the Blogs button.
* \*Many of the other requirements are yet to be implemented

# Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the* ***non-functional requirements*** *listed in your RD).*

* So far one of the only non-functional testing requirements we have been able to focus on is the intuitive ease-of-use of the website for new users. The sign in option is always listed at the top of the page and is clearly readable and so is the JukeBoxd self-titled link to the homepage at the tab at the top.
* We have also focused on keeping code and documentation neat and organized so it is clear to see what we have accomplished so far.
* The search bar is very easy to use and allows the user to both hit enter for their query as well as click the search button.
* The cards that are displayed after entering a search query include both album/song titles as well as the images linked to each piece of media. The cards are large and easy to distinguish from one another.
* The formatting of the blogs is easy to understand and navigate.
* When searching for a song or artist, the most relevant result will appear even if the search query is misspelled.

# Non-Execution-based Testing (10 points)

*Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).*

* We have reviewed our code pertaining to the Spotify API to make sure that it properly links to Spotify’s database and gives us the results we want based on each artist’s ID.
* We have reviewed our database and packages needed to make sure we can all run the code.
* We have reviewed our routing and made sure all of our code is formatted properly and easily readable.