**Software Implementation and Testing Document**

**For**

**Group 3**

**WeShed**

Version 2.0

**Authors**:

Steven Knudson

Eliot Shea  
Noal Gelser  
Rogelio Lopez

# Programming Languages

Our group is using JavaScript and MySQL. JavaScript is being used for the front and back end implementations. In the front, JavaScript is being used to serve HTML via React implementations, while in the back, JavaScript plays a role in declaring listening functions and how the listening server is setup to respond and execute. The back-end server communicates with our AWS database via MySQL queries and then communicates those queries back to front end. React serves HTML through the render function.

# Platforms, APIs, Databases, and other technologies used

We are using an AWS RDS MySQL Database to provide persistent cloud data for our schema. The back-end talks to the AWS database and the back-end responds to front-end requests for data. The front end consists of HTML that is dynamically generated using ReactJS.

Packages/Libraries:

Front End:

-ReactJS

-ReactPlayer (component on NPM for convenient embedding of videos via url)

Back End:

-Express for creating the application

-body-parser for json packaging

-morgan for debugging

-cors(Cross Origin Resource Sharing) to understand front-end requests

-mysql to create a connection with the database

-crypto for encrypting passwords in the database

-jsonwebtoken for hashing cookie tokens

# Execution-based Functional Testing

Our execution-based functional testing consisted of both every member of the group testing their individual source code, as well as a team testing of the code when all members source code was integrated together.

These detailed tests included testing on everything interactive on the website.

Header.js: We tested the testuser117 cookie hash and it worked on all private routes transitioning successfully.

Register.js:

We gave input testuser117, 1234, [t117@gmail.com](mailto:t117@gmail.com), test, user in the corresponding fields. It returned success and showed up in the mySQL database.

Login.js:

We gave input testuser117, 1234 and returned success and stored hash in browser.

In addition, we pressed logout and it cleared the browser of the cookie hash.

Songs.js:

Many tests were run while implementing song-session tracking, with recompiling and running the page repeatedly to ensure desired functionality was achieved. Many JavaScript alert() statements were included to make sure the contents of the variables was as desired. Converting the date into a SQL friendly format was also repeatedly tested.

Playlist.js:

It was important that data was joining correctly, so we compared mySQL to the output of the playlist page and it was verified to be the proper results. For more testing, we pressed the delete current song button and it deleted the selected song. Furthermore, we pressed the delete playlist button and it deleted all the songs in the playlist. Another test was making sure play sessions were saved under every page update. Under every circumstance, it stored the play session properly.

Profile.js:

SearchPage.js

Testing included making sure that only the first word of every song was being searched instead of it searching for a .include(). Also making sure that every song, when clicked on, searched the right one.

Stats.js

For the progression donutchart all numbers 0 through 1095 were used to test the rendering. This was done using a full fast forward button that triggered the cycle of 0 through 1095 integer test renders. For the level badge, we looked at the count of the output of the total number of play sessions and made sure that when it was used in the formula, it corresponded to the correct numeric rendering. For the streaks, we tested a wide range of dates including 0000-00-00. This turned out to cause an invalid date error for MySQL. It is expected that the user will give a valid date when they make a play session.

Home.js

For adding friends, we gave input testuser117 and testuser118 and it added it to both the front end and the MySQL database. For adding challenges, we gave input “testuser117”, 100, “Play 100 times” in the corresponding fields and submitted it and it was added to the front end and the MySQL database.

# Execution-based Non-Functional Testing

Our execution-based non-functional testing consisted of both every member of the group testing any non-functional elements in their software, as well as a team testing of the non-functional requirements in a team meeting when all code was integrated together. Security of passwords was tested by comparing the same passwords and ensuring that they never had the same hash in the database. Furthermore, we tested protected routes in the header with invalid cookies and it wouldn’t redirect. For performance, all front-end data rendered in less than a second.

# Non-Execution-based Testing

Non-execution-based testing was done during meetings, or after class when we thought that

walkthroughs were needed to better understand the code. This was done many times during the first increment.