SE 3XA3: Test Report
DJS

Team , Team Name
Student 1 name and macid
Student 2 name and macid
Student 3 name and macid

December 7, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Introduction

This document outlines a report on the various tests that were conducted to verify the functionality of DJS. Test cases can be performed by interest clients to determine the validity of our software, and the robustness of our test suite.

2 Functional Requirements Evaluation

Testing was performed with white box unit testing, black box automated system testing, black box manual system testing, and stress testing.

Due to the nature of a server/client system, it is difficult to provide a complete test suite to ensure complete functionality of the system. However, an adequate test suite was provided.

All functional tests passed.

2.0.1 Client-side Graphical Interface

Webpage Title and Buttons Loaded	
Type:	Functional, Dynamic, Manual Testing
Initial State:	Web page is not loaded.
Input:	User's internet browser should navigate to the servers web address.
Output:	The server should serve the users request and load a webpage with a title and five buttons underneath.
Test Procedure:	The web page should be loaded and the title along with 5 buttons should be displayed to the user.
Result:	PASS

Button Includes Song Title	
Type:	Functional, Dynamic, Manual Testing
Initial State:	Web page is opened on users internet browser.
Input:	User's internet browser should navigate to the servers web address.
Output:	The webpage loaded should include five buttons with each button having text. The text inside each button should be of a different unique song title (each button has a song title).
Test Procedure:	Load webpage on user internet browser and check if buttons have song titles (if test failed then output should be giberish on button).
Result:	PASS

Vote Causes Button To Be Highlighted	
Type:	Functional, Dynamic, Manual Testing
Initial State:	Web page is opened on users internet browser and buttons should be present with no prior votes.
Input:	User clicks on one button from the webpage.
Output:	The corresponding button selected will be highlighted in some form to indicate a vote has been cast and recorded .
Test Procedure:	Load webpage on user internet browser and check if buttons have loaded. Once the buttons are present the tester selects one song and should result in the same button being highlighted.
Result:	PASS

Graphic Object Shows Total Number Of Votes	
Type:	Functional, Dynamic, Manual Testing
Initial State:	The web address is not loaded. The server has just started.
Input:	User navigates to web address.
Output:	The web page should load some graphical object which contains the number of votes for each corresponding button. The number of votes should be zero initially .
Test Procedure:	The server should be freshly started. The tester should then navigate to the appropriate web url and load the web page. Once the web page has been loaded the tester can then observe the total number of votes.
Result:	PASS

2.0.2 Client-side Backend Interface

Remeber Voted Song	
Type:	Functional, Dynamic, Manual Testing
Initial State:	One song should have been voted and the internet browser closed.
Input:	The tester will place a vote on one random song and close the browser. After, the web page should be opened again by the tester and the page loaded.
Output:	The song title that was picked before closing the internet browser should be highlighted.
Test Procedure:	The tester will open a internet browser and load the webpage. After the webpage has been loaded the user will cast a vote. The internet browser opened previously will be closed. Then after the tester will reopen the internet browser and the song title that was selected previously should be highlighted.
Result:	PASS

Song List Should Be Valid	
Type:	Functional, Dynamic, Automated Testing
Initial State:	The web address is loaded. The server has just started.
Input:	The song titles that appears on website will be the input for the automated testing.Another input would be the music currently available on the server.
Output:	The unit testing function will return either with true or false. The result of true will indicate that the song list appeared on the web page matches the song titles available on the server.
Test Procedure:	The automated test will record each song title generated and displayed on the client side. Furthermore, the songs available to the server will also be recorded. The result is calculated by matching all the songs recorded from the web page to the songs available to the server
Result:	PASS

2.0.3 Server-side Backend

Create Cookie To Allow One Vote Per User	
Type:	Structural, Dynamic, Automated Testing
Initial State:	The web address is not loaded. The server has just started.
Input:	A simulated user with random voting pattern that is active every couple of seconds.
Output:	The unit testing function will return true or false. The testing function will return true when the sum of total votes for each song equals the number of users connected.Correspondingly the return value of false will suggest that one or more simulated users will have more then one vote.
Test Procedure:	The automated test will create a certain number of random users. The server will create a cookie for each user that indicates a unique id to identify each user. The randomly generated users will all vote for one song that is picked randomly and then change all the votes to another random song (ie. users 1..15 vote for song 1 then vote for song 2). The test function will then check the number of total votes for each song and sum them together which should equal the number of users generated.
Result:	PASS

Reset Votes After Playing Song	
Type:	Structural, Dynamic, Automated Testing
Initial State:	The number of total votes for a certain song is above zero.
Input:	The test function will need the total number of votes right after a certain song has been done playing.
Output:	The unit test function will return true or false. The test function will return true when the total number of votes after playing a song is zero.
Test Procedure:	The test procedure will start by having the webpage start with a song with the total number of votes above zero. The test function will then check after the song has played if the total number of votes is equal to zero.
Result:	PASS

Check If Song List Is Unique	
Type:	Structural, Dynamic, Automated Testing
Initial State:	The server started and web page loaded.
Input:	The test function will need the song list that is being sent to the client.
Output:	The unit test function will return true if the song list sent is unique and has no duplicates.
Test Procedure:	The test function will use the song list being sent to the client and store it into an array. As the song list for the client updates after a song has been played the new song list will be appended to the array. After the last song has played the test function will check the array to see if the server has sent any duplicate song titles and will result in a true or false value.
Result:	PASS

Check If 5 Random Songs Picked	
Result:	PASS
Type:	Structural, Dynamic, Automated Testing
Initial State:	The server started and web page loaded.
Input:	The test function will need to count the number of songs sent to the client after playing the current song.
Output:	The unit testing function will return true if the count is equal to five after playing the current song .
Test Procedure:	The test function will use a counter and check if the counter is equal to five after the current song is done being played.
Result:	PASS

Play Most Voted Song	
Type:	Structural, Dynamic, Automated Testing
Initial State:	The server started and web page loaded.
Input:	The test function will need to record the total number of votes and the corresponding song title picked.
Output:	The unit testing function will return true if the application plays the right song.
Test Procedure:	The testing function will use a counter and rank the songs by votes and check if the playing is song is equal to the song selected with the most votes.
Result:	PASS

2.1 Automated Testing

Automated testing was done with a combination of Mocha.JS, a unit testing framework Node.JS, and Selenium-Webdriver. The test cases are located in the test folder, located here.

3 Nonfunctional Requirements Evaluation

3.1 Usability

3.2 Performance

3.3 etc.

4 Comparison to Existing Implementation

5 Unit Testing

The specific modules used for Unit testing can be found in the test folder which is in the src folder. The results for these tests can also be found in the same folder which is also linked here.

6 Changes Due to Testing

7 Automated Testing

Automated Testing was done through a combination of Mocha.JS (for unit testing) and Selenium-Webdriver (for system-wide testing).

Mocha.JS tested various pure functions throughout the codebase, based on a predefined set of input and output test vectors.

Selenium-Webdriver was used to produce a firefox instance, simulate a connection to the server, simulate user interaction, and analyze the HTML output to ensure the server is producing the correct data, and that the web client is receiving and parsing the data correctly.

7.1 Specific System Tests

Reads songs from music folder	
Initial State:	Library module called to read songs from a folder
Input:	Folder with songs
Output:	List of all the songs in the folder

Reads metadata from a song	
Initial State:	Metadata module called to read the metadata from a music file
Input:	Music file
Output:	Correct metadata information extracted from file

Voting System returns highest voted item	
Initial State:	Multiple users cast their votes, voter module is called
Input:	List of votes
Output:	Returns highest rated item

Voting System handles an empty songs array	
Initial State:	Voter module is called
Input:	Empty array of songs, non-empty array of votes
Output:	Returns empty string

Voting System handles an empty votes array	
Initial State:	Voter module is called
Input:	Empty array of votes, non-empty array of songs
Output:	Returns empty string

Webpage Title is Loaded	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	Correct title of browser window is displayed

Loads the first button	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	First button has the correct name

Loads the second button	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	Second button has the correct name

Loads the third button	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	Third button has the correct name

Initially sets first vote to zero	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	First vote-count element has a value of 0

Initially sets second vote to zero	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	Second vote-count element has a value of 0

Initially sets third vote to zero	
Initial State:	Server is running, browser directed to webpage
Input:	N/A
Output:	Third vote-count element has a value of 0

Votes for an item when a user clicks a button	
Initial State:	Server is running, browser directed to webpage, a vote button is clicked
Input:	N/A
Output:	Vote count for the corresponding button has a value of 1

8 Trace to Requirements

gording			
Wasaddast	sadadup	BOadasdasdai	yeasadsad

9 Trace to Modules

10 Code Coverage Metrics