SE 3XA3: Test Plan
DJS

Team 12 , DJS
Amandeep Panesar panesas2
Taha Mian miantm
Victor velech

October 31, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
OCT 28	1.0	Rev0

1 General Information

1.1 Purpose

The test plan document is a helpful tool for many large scale projects since it allows concise information about testing,verification, and validation geared towards the project. The following test cases were created for future references and allows the project to be implementated with testing and maintenance in mind. The test plan document will be updated before the project is fully implementated to allow for revision and any major changes involved.

1.2 Scope

The project, "DJS",is a democratic voting system which allows users to vote for music. Thus testing can cover many areas such as: client methods (ie: update song client, etc), server methods (ie: create Cookie), data structures, and sorting algorithms.

1.3 Acronyms, Abbreviations, and Symbols

N/A

1.4 Overview of Document

This is the test plan document for the project DJS, which is a reconstruction of the application PlayMyWay (href this). The test plan uses the functional and non-functional requirements to detect any errors in the project DJS. The document goes over various techniques for testing such as Manual and automated testing, structural and functional testing, static and dynamic testing, fault testing.

2 Plan

2.1 Software Description

The server running DJS is using nodejs with multiple libraries which includes : express, handlebars, express–handlebars, and socket.io. The implementation of DJS has been modularized into three aspects.The player.js file plays

the music, server.js hosts the website, votes.js helps with voting, and library.js which returns list of songs.

2.2 Test Team

All project members will participate and be responsible for writing and executing tests.

2.3 Automated Testing Approach

The automated testing will be implemented by using javascript libaries and custom unit testing function created in javascript.

2.4 Testing Tools

The following testing libraries will be use: Selenium, Mocha, Karma, and Protractor.

2.5 Testing Schedule

Webpage should be operational by Oct 21/16 Server should be able to handle multiple users Oct 28/16 Voting system works by Oct 23/16 Songs in queue by Oct 25/16

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 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.

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).

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.

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.

3.1.2 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.

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

3.1.3 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.

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.

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.

Check If 5 Random Songs Picked	
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.

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.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirements

Appearance Tests

1. Non-Functional Requirement Test 1

   Type: Structural, Static, Manual

Initial State: Web page is loaded from a device that has Internet

Input/Condition: Users rate the web page on the ascetics of from a rating of one to 10.

Output/Result: The overall average of the results should be over 7.5.

How test will be performed: Users will take a short survey rating the Appearance, the results of multiple users will be tabulated. The overall user average score will be taken, must have 20 plus users.

Style Tests

1. Non-Functional Requirement Test 2

   Type: Structural, Manual, Static etc.

   Initial State: Web page is loaded from a device that has Internet

   Input: Users rate the web page on the ascetics of from a rating of one to 10.

   Output: The overall average of the results should be over 7.5.

   How test will be performed: Users will take a short survey rating the Style, the results of multiple users will be tabulated. The overall user average score will be taken, must have 20 plus users.

3.2.2 Usability and Humanity Requirements

Ease of Use Requirements Test

1. Non-Functional Requirement Test 3

   Type: Structural, Static, Manual

Initial State: Web page is loaded from a device that has Internet

Input/Condition: Users rate the web page on the Ease of Use from a rating of 1 to 10.

Output/Result: The overall average of the results should be over 7.5.

How test will be performed: Users will take a short survey rating the Ease of Use, the results of multiple users will be tabulated. The overall user average score will be taken, must have 20 plus users.

Understandability and Politeness Requirements Test

1. Non-Functional Requirement Test 4

Type: Structural, Manual, Static

Initial State: Web page is loaded from a device that has Internet

Input/Condition: Users rate the tutorial on the web page that shows how it works on the of from a rating of 1 to 10.

Output: The overall average of the results should be over 7.5.

How test will be performed: Users will take a short survey rating the tutorial, the results of multiple users will be tabulated. The overall user average score will be taken, must have 20 plus user survey.

Accessibility Requirements Test

1. Non-Functional Requirement Test 5

Type: Structural, Manual, Static

Initial State: Web page is not loaded from a device that has Internet using local Wifi

Input/Condition: Users should be able to access the web page from local Wifi.

Output: The Web page is loaded on the device from local Wifi.

How test will be performed: The Web page will be loaded from 5 different local Wifi's every time the web page should load

3.2.3 Performance Requirements

Speed and Latency Requirements Test

1. Non-Functional Requirement Test 6

   Type: Structural, Dynamic, Manual

   Initial State: Web page is loaded from a device that has Internet

   Input/Condition: There should be very little latency in loading the web page and making a vote

   Output/Result: Web page should be loaded, Vote should be counted.

   How test will be performed: It should take no longer than 3 seconds for the web page to load and to cast a vote, the time will be approximate so a stop watch will be enough to measure the latency times.

Precision Test

1. Non-Functional Requirement Test 7

   Type: Structural, Manual, Dynamic

Initial State: Server is running

Input/Condition: The song with the most votes should be played next

Output: The song with the most votes is played next

How test will be performed: track the amount of votes and the songs for more than 50 song changes and make sure that they are correct

Reliability and Availability Requirement Test

1. Non-Functional Requirement Test 8

   Type: Structural, Manual, Static

   Initial State: Server is running

   Input/Condition: Server continues to run

   Output: Server should constantly be playing music

   How test will be performed: Let the server continuously run for a length period of time and check put a mic next to it and detect whether or not the speaker playing music. Sound should be coming from the speaker all the time grace period of 50 seconds for in between songs

Robustness Requirements Test

1. Non-Functional Requirement Test 9

   Type: Structural, Manual, Static

   Initial State: Server is running

Input/Condition: Server plays next song(s)

Output: Next song(s) being played is from the sever only

How test will be performed: Users will take a short survey rating the tutorial, the results of multiple users will be tabulated. The overall user average score will be taken, must have 20 plus user survey.

Capacity Requirements Test

1. Non-Functional Requirement Test 10

   Type: Structural, Manual, Static

   Initial State: Server is Running

   Input/Condition: The system storage must be above 32 GB

   Output: The storage space is above 32 GB

   How test will be performed: Inspect the system storage space (different for every OS)

Scalability Requirements Test

1. Non-Functional Requirement Test 11

   Type: Structural, Manual, Static

   Initial State: Server is running

   Input/Condition: Users using the server

   Output: At least 300 users at a time

How test will be performed:Using Post man to generate multiple users 300 is max limit.

Longevity Requirements Test

1. Non-Functional Requirement Test 12

   Type: Structural, Manual, Static

   Initial State: Sever not running

   Input/Condition: run the server

   Output: Server should continue to run unless manual turn off

   How test will be performed: leave server on for a lengthy period of time and then time the amount it on for and come back and see if its still working

3.2.4 Operational and Environmental Requirements

Requirements for Interfacing with Adjacent Systems Test

1. Non-Functional Requirement Test 13

   Type: Structural, Manual, Static

   Initial State: Web page is not loaded from a device that has Internet

   Input/Condition: Web page is loaded from a device that has Internet

   Output: The web browser should fluently communicate with the server and record user interaction.

   How test will be performed: Users will take a short survey rating the tutorial, the results of multiple users will be tabulated. The overall

user average score will be taken, must have 20 plus user survey.

Productization Requirements Test

1. Non-Functional Requirement Test 14

   Type: Structural, Manual, Static
   Initial State: Server not installed

   Input/Condition: Runnable is installed

   Output: Server is installed on system

   How test will be performed: Runnable will be excecuted on multiple systems minimum 5 with different specs, and each time should yeild installation sucessful

3.2.5 Maintainability and Support Requirements

Access Requirements Test

1. Non-Functional Requirement Test 15

   Type: Structural, Manual, Dynamic
   Initial State: Web page is loaded from a device that has Internet

   Input/Condition: User clicks to vote

   Output: Add one song voted for and total number of voters

   How test will be performed: The total number of voters should not increase everytime a voter clicks on another song. The amount of votes per song should change respective of the song last selected

2. Non-Functional Requirement Test 16

   Type: Structural, Manual, Static

   Initial State: Server is running, and user has been registered

   Input/Condition: device is restarted or local wifi is restarted

   Output: server should still hold user information

   How test will be performed: A voter will load web page once vote and then restart there phone and then load the web page again the total votes should not change, then do the same thing but this time restart the wifi connection total votes should be constant

3. Non-Functional Requirement Test 17

   Type: Structural, Manual, Static

   Initial State: Web page is loaded from a device that has Internet

   Input/Condition: Web page is loaded from a device not on local wifi

   Output: Web page should not load web page

   How test will be performed: will try and connect to the web page from another wifi connection

Privacy Requirements Test

1. Non-Functional Requirement Test 18

   Type: Structural, Manual, Static

   Initial State: Server is running

Input/Condition: User loads the web page

Output: There should be no information about the server page to a user

How test will be performed: The sever will be running and a user will try and access the administrators webpage access should be denied

2. Non-Functional Requirement Test 19

Type: Structural, Manual, Static

Initial State: Web page is loaded from a device that has Internet

Input/Condition: User information is stored

Output: Other users should not have access to this information

How test will be performed: User will try and look at other user information on administrator page permission should be denied

4 Tests for Proof of Concept

Run Server	
Type:	Structural, Dynamic, Manual Testing
Initial State:	Nothing Running .
Input:	Javascript Files.
Output:	Running Server.
Test Procedure:	The server should run when the command node file.js is ran.

Play Music	
Type:	Structural, Dynamic, Manual Testing
Initial State:	Server should be running and no music should be played .
Input:	Any song from the generated song list.
Output:	Music playing.
Test Procedure:	Run server and vote for any song. Then after votes have been counted the song with the most votes should be played.

Load Buttons	
Type:	Structural, Dynamic, Manual Testing
Initial State:	Nothing Running .
Input:	Javascript Files.
Output:	Running Server.
Test Procedure:	The server should run when the command node file.js is ran.

Voting System	
Type:	Structural, Dynamic, Manual Testing
Initial State:	The server should be running with the webpage loaded with no votes .
Input:	Vote .
Output:	Vote for song title .
Test Procedure:	The server should out put an array for now which shows the votes in a string array.

5 Comparison to Existing Implementation

The product DJS is a dynamic voting system which allows user to vote for certain songs. The previously implemented open source project called

PlayMyWay lacked in well documented code. Comparing the two products is beneficial to DJS since it allows the product to evolve. The changes that might occur however are dependent on the project's progress and would require the scope to change. The modification to the scope would help develop and implement core features in DJS. The requirements that may add extra features will only be incorporated if time permits. The subsection of requirements such as look and feel, performance, security, and accuracy still need improvements. The section for look and feel for example is only partially implemented in the proof of concept. The requirement of displaying the total number of votes for each song will only be added if time permits. Another example, the performance requirement of loading buttons and quickly is still waiting to be implemented. However, since most core requirements such as playing music and voting have been fulfilled most requirements suggested from before and others will be developed. Although most requirements are important the list may also be narrowed down to help create more time for more ideal requirements such as total votes. Thus, after comparing DJS with the original implementation it demonstrates the similarities and the small differences still left to be negated

6 Unit Testing Plan

6.1 Unit testing of internal functions

The implemented unit tests will help examine the product and will allow for clarity. The automated tests will be used to test a multitude of functional and nonfunctional requirements. Majority of these tests will use the internal functions and variables implemented in the product. All tests will be correlated to core functions utilized in the product to ensure predictable outputs and behaviours for normal, abnormal, and negative scenarios.

6.2 Unit testing of output files

The output for the DJS product will be the music playing and the client-side graphical interface. The unit testing for output will be done with the combination of both manual and automated testing. In addition, the testing of output will also include using an external library called selenium which helps javascript simulate clicks and other actions a user would commit. The

tests will call functions that create the view and check if the view has appeared. An example would be is checking if the buttons have been loaded. The test for checking if buttons have appeared would involved the selenium library and would simulate buttons clicks. The unit tests will ensure that the proper methods are called and the output is the expected result.