SE 3XA3: Test Plan
DJS

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

December 7, 2016

Contents

1 General Information **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Acronyms, Abbreviations, and Symbols . . . 1
- 1.4 Overview of Document . . . 1

2 Plan **1**
- 2.1 Software Description . . . 1
- 2.2 Test Team . . . 2
- 2.3 Automated Testing Approach . . . 2
- 2.4 Testing Tools . . . 3
- 2.5 Testing Schedule . . . 3

3 System Test Description **4**
- 3.1 Tests for Functional Requirements . . . 4
  - 3.1.1 Client-side Graphical Interface . . . 4
  - 3.1.2 Client-side Backend Interface . . . 6
  - 3.1.3 Server-side Backend . . . 7
- 3.2 Tests for Nonfunctional Requirements . . . 9
  - 3.2.1 Look and Feel Requirements . . . 10
  - 3.2.2 Usability and Humanity Requirements . . . 10
  - 3.2.3 Performance Requirements . . . 12
  - 3.2.4 Operational and Environmental Requirements . . . 14
  - 3.2.5 Maintainability and Support Requirements . . . 15

4 Tests for Proof of Concept **17**

5 Comparison to Existing Implementation **18**

6 Unit Testing Plan **18**
- 6.1 Unit testing of internal functions . . . 18
- 6.2 Unit testing of output files . . . 19

List of Tables

1 **Revision History** . . . ii

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
OCT 28	0.1	Rev0
DEC 6	1.0	Rev1

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

1.4 Overview of Document

Making an issue to href something MAY be a better idea - CM
This is the test plan document for the project DJS, which is a reconstruction of the application PlayMyWay: `https://github.com/malithsen/playmyway`. 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 server running DJS is using node.js with multiple libraries which consist of the

following: express, handlebars, express–handlebars, and socket.io. The implementation of DJS has been made into eight modules. The module server.js is used for hosting the webpage and uses several modules such as, voter.js, player.js, metadata.js, home.js, library.js, args.js, and error-handler.js. However, server.js is the main module that is being used in every other module. Server.js uses library.js to get an N amount of songs, the number N is set in the module server.js, and in this case the number has been pre-set to 5 . The function of library.js is to get a list of songs that are to be displayed on the webpage to be voted for and the songs are then randomly selected. The same idea can be applied to metadata.js, except metadata.js gets the metadata of the songs selected by library.js and saves the album in a folder called artwork. Another module, voter.js also relies on server.js because voter.js counts the amount of votes and server.js selects the song with the highest amount of votes. Furthermore, home.js is used to display the webpage by server.js this consists of all the information such as artwork, title of the song, and number of votes. Also, server.js uses player.js to just play audio to the speakers of the system. Moreover, error-handler.js is used by server.js for catching errors and determines what message to print to the terminal. Lastly, args.js is used by server.js when the user runs server.js with certain command line arguments.

2.2 Test Team

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

2.3 Automated Testing Approach

Speak to the why, where and how you will run automatic tests - CM

Automated testing will be done through white-box unit tests and system-wide black-box tests. White-box unit testing will be used to test the output of various functions in the server, library, player, metadata, args, and error-handler modules. White-box system-wide testing will be done to ensure that the system fulfills the more broad functional requirements, and that the server and client are able to work together correctly. The testing suite will automatically connect to the server with a web browser instance, vote for songs, and check that the resulting webpage data is correct.

2.4 Testing Tools

Should the reader know these? What are they for, purpose? I know Karma includes many features, which will you use? - CM

Unit testing will be done with Mocha.JS, a unit testing library for Node.JS and Javascript. System testing will be done with Selenium-Webdriver, a library that simulates a web browser, connects to the server, simulates user interaction, and gathers information about the resulting webpage, to determine proper functionality. Specifically, the Webdriver will connect to the server, and determine if the UI elements display songs, with clickable buttons, album art, and a vote count that is initially zero. The webdriver will cast a vote for the first song, and see if the vote-count reflects this by incrementing from 0 to 1.

2.5 Testing Schedule

Poor formatting, poor use of Gantt charts - CM

Functionality Being Tested	To Be Completed by:
Webpage should be operational	Oct 21/16
Webpage loads the right metadata and song title	Oct 22/16
Voting system counts the right amount of votes	Oct 23/16
Voting sytem selects the right song	Oct 23/16
Songs in queue	Oct 25/16
Server should be able to handle multiple users	Oct 28/16

This can also be found in the Gnatt Chart here.

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

I don't like how these are structured. For example, NF Test 1: Initial State: Webpage has not been loaded. Input: User opens the webpage by doing X. Output: Webpage is open. How Test is Preformed: The user will rate on a scale... That is a better test because it is specific and repeatable. Also, make a survey and use symbolic parameters for everything (e.g. stress testing) - CM

3.2.1 Look and Feel Requirements

Appearance Tests

User Interface is aesthetically pleasing	
Type:	Structural, Static, Manual
Initial State:	At least 20 users take a feedback survey
Input:	User rates web page based on the aesthetics (from 1 to 10) on a custom survey
Output:	Average results of survey
Test Procedure:	At least 20 users will take a short survey rating the appearance. The results will be tabulated and an average calculated. The average must be atleast 7.5

Style Tests

3.2.2 Usability and Humanity Requirements

Ease of Use Requirements Test

User Interface is easy to use	
Type:	Structural, Static, Manual
Initial State:	At least 20 users take a feedback survey
Input:	User rates web page based on the ease of use (from 1 to 10) on a custom survey
Output:	Average results of survey
Test Procedure:	At least 20 users will take a short survey rating the appearance The results will be tabulated and an average calculated. The average must be atleast 7.5

Understandability and Politeness Requirements Test

User Help Manual is helpful and understandable	
Type:	Structural, Static, Manual
Initial State:	At least 20 users take a feedback survey
Input:	User rates the effectiveness of the survey
Output:	Average results of survey
Test Procedure:	At least 20 users will take a short survey rating the appearance The results will be tabulated and an average calculated. The average must be atleast 7.5

Accessibility Requirements Test

Web page is loadable on a local WiFi connection	
Type:	Structural, Manual, Static
Initial State:	User attempts to connect to server hosted on the same WiFi network
Input:	N/A
Output:	The web page is loaded on the device from local WiFi
Test Procedure:	The web page will be loaded from five different devices on two local Wireless networks. All devices should connect and display the webpage

Web page functions on all HTML5 web browsers	
Type:	Structural, Manual, Static
Initial State:	User attempts to connect to server hosted on the same WiFi network
Input:	N/A
Output:	The web page is loaded on the device from local WiFi
Test Procedure:	The web page will be loaded on at least 5 different browsers (all of which support HTML5). The voting mechanism should be functional on all devices

3.2.3 Performance Requirements

Speed and Latency Requirements Test

Web page must load within 3 seconds	
Type:	Structural, Dynamic, Manual
Initial State:	Web page is loaded from a device connected to the same WiFi network. The server must be running on an x86-54 based, OS X or Linux system
Input:	N/A
Output:	N/A
Test Procedure:	It should take no longer than three seconds for the web page to load and cast a vote

Precision Test

Song with most votes is always selected to play next	
Type:	Structural, Manual, Automated
Initial State:	Server is running, users are connected
Input:	User Votes are inputted to the voter module
Output:	Vote choice is determined and returned by the voter module
Test Procedure:	The highest voted song (or any of the highest rated, in the case of a tie for first place) should be selected. Automated testing done in test-voter module

Reliability and Availability Requirement Test

Server should constantly be playing music	
Type:	Structural, Manual, Static
Initial State:	Server is not running
Input:	Start the server
Output:	Audio output to speakers
Test Procedure:	Music should begin playing initially, and continue to play without any user interaction until all music in music directory have been played

Robustness Requirements Test

Server handles songs with empty album art metadata	
Type:	Structural, Automated/Manual, Static
Initial State:	Server is running
Input:	Songs with empty metadata are in the music folder
Output:	Data sent to webpage
Test Procedure:	Songs with no album artwork should not crash the system, and should display a 'default' album art picture on the webpage. Automated testing is done in the test-metadata module

Capacity Requirements Test

Scalability Requirements Test

Server should handle at least 300 users at a time	
Type:	Structural, Automated, Static
Initial State:	Server is running
Input:	300 clients, generated by selenium-webdriver, connect to the server
Output:	N/A
Test Procedure:	Server performance should not be significantly affected. Voting and music playback should still function

Longevity Requirements Test

Server should continue to run unless manually turned off	
Type:	Structural, Manual, Static
Initial State:	Server is not running
Input:	Run the server
Output:	N/A
Test Procedure:	Server should run continuously for at least 20 minutes without user interaction, then terminated by the user

3.2.4 Operational and Environmental Requirements

Requirements for Interfacing with Adjacent Systems Test

Web page functions on all HTML5 web browsers	
Type:	Structural, Manual, Static
Initial State:	User attempts to connect to server hosted on the same WiFi network
Input:	N/A
Output:	The web page is loaded on the device from local WiFi
Test Procedure:	The web page will be loaded on at least 5 different browsers (all of which support HTML5). The voting mechanism should be functional on all devices

Productization Requirements Test

Server is installable on Mac OS and Linux through the given terminal commands	
Type:	Static, Manual, Structural
Initial State:	Mac OS machine without the server installed
Input:	Server is installed via the install command given in the README
Output:	Server is functional
Test Procedure:	All other test requirements should be passed on this system

3.2.5 Maintainability and Support Requirements

Access Requirements Test

Privacy Requirements Test

Restarting the device or browser should not allow voting twice	
Type:	Structural, Manual, Dynamic
Initial State:	Web page is loaded from a device on a local network. A vote is issued, then the device is restarted and reconnected
Input:	N/A
Output:	N/A
Test Procedure:	The second vote should replace the initial vote on the server

Webpage is only accessible from a local network	
Type:	Structural, Manual, Static
Initial State:	Attempt to load web page from outside the local network
Input:	N/A
Output:	HTML response
Test Procedure:	The server should not be accessible from an outside network

Users should have no access to other user data	
Type:	Structural, Manual, Static
Initial State:	Connect to server
Input:	N/A
Output:	HTML Response
Test Procedure:	The HTML response should contain no data about other users, other than how many votes have been totalled for each song

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

This is NOT just comparing the projects. It is detailing how to parallel test the two to be sure your project matches the other's (and subsequently your) requirements - CM

It is not possible to parallel test with the currently existing implementation. PlayMyWay, the project which we based ours on, does not have any automated testing suite, and is not currently in working form. The software is buggy and the server does not satisfy any of the main functional requirements in its current form, due to the software bugs, making a direct comparison impossible.

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.