

C-lyrics - A Word Cloud for Lyrics

Testing Report

Justine Cocchi
jcocchi@usc.edu

Kelsey Fargas
kfargas@usc.edu

Mark Krant
mkrant@usc.edu

Milad Gueramian
gueramia@usc.edu

Jeff Kang
kangjr@usc.edu

Séb Arnold
arnolds@usc.edu

30 March 2015

Contents

Executive Summary	3
1 Introduction	4
1.1 Purpose	4
1.2 Overview	4
1.4 Definitions And Acronyms	4
1.5 Scope	9
1.6 Black-Box and White-Box Methodology	9
2 Unit Testing	9
2.1 Karma Testing Software	10
2.2 PHPUnit Testing Software	10
2.3 Measures and Coverage	14
3 Acceptance Testing	16
3.1 Protractor Testing Environment	16
3.2 Measures and Coverage	18
4 Quality Testing	18
4.1 Reliability Tests	18
4.2 Availability Tests	18
4.3 Other Quality Tests	19
5 Testing Compliances with PMP	19
5.1 Quality Assurance Testing	19
5.2 Risk Monitoring	19
6 Appendices	19
6.1 Declaration of Milestones	19
6.2 Deviations From PMP	20
6.2.1 Deviations from Schedule	20
6.2.2 Deviations From Milestone Planning	21
6.3 Screenshots	21

Executive Summary

C-lyrics is a public website that will generate a word cloud for any given artist based on the most frequently used words that appear across all of the artist's published songs. This product will interface with the EchoNest API which will serve as the database from which we find and analyze the songs. By clicking on a specific word in the word cloud the user can see a list of all of the songs that word appears in and how frequently it occurs in each song. Furthermore, the user can click on any listed song title to see the complete lyrics for that song with the original word that was selected from the word cloud highlighted every time it appears.

C-lyrics is intended for use by the general public. There will be no login required and there is no stored history of previous searches. Because of this we will have very low memory requirements and can run the product off of one server. The user can access C-lyrics using any device running any OS, assuming it has an internet connection. After typing in the artist name and selecting the submit button, the word cloud will be generated and will be able to be shared via Facebook.

1 Introduction

1.1 Purpose

This software testing document is meant to explain, detail, and report on the white-box and black box tests that are run on the implemented software code in order to assure quality. In addition, coverage metrics, justifications and processes will be described. The document also includes project management plans and schedules that were used to accomplish the testing phase.

1.2 Overview

C-Lyrics is implemented using 2 languages, PHP and JavaScript. The backend is implemented using PHP while the front end is built using Java Script with the Angular JS framework. We used the PHPUnit framework for white box testing the back end PHP functions. However, we also needed to perform some whitebox testing on the AngularJS frontend implementation. To test code implemented using AngularJS we used the Karma testing platform which works conveniently well for the purpose. To perform our blackbox end-to-end tests we used the Protractor framework.

The document is therefore organized into two main sections, Unit testing, and Acceptance testing. Details about each testing scenario are explained in subsections of these two main sections. The project management documentation can be found in the appendices which comprise the last section of this document. ##

1.3 References [1] IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

[2] “word cloud”. Oxforddictionaries.com (January 31, 2015)

[3] EchoNest API documentation (January 29, 2015)

[4] A document to remind us the definitions of each UML symbol UML Cheatsheet (February 17, 2015)

[5] Gliffy, a tool to create flowcharts and diagrams. Gliffy.com (February 17, 2015)

[6] Wikipedia definition of black-box testing. [http://en.wikipedia.org/wiki/Black-box_testing] (February 17, 2015)

[7] Wikipedia definition of white-box testing. [http://en.wikipedia.org/wiki/White-box_testing] (February 17, 2015)

1.4 Definitions And Acronyms

Term	Definition
------	------------

AJAX	Asynchronous JavaScript And XML. Technology allowing the transfer of data from between the front- and back-end without reloading the web page.
API (EchoNest)	API will refer to the EchoNest API. EchoNest is a free API that allows developers to retrieve lyrics and artist information in web pages and other programs.
Autocomplete	Autocomplete refers to the functionality addition to the Search Bar, allowing users to enter minimal characters and choose artists that are most similar to the string and display a picture of those artists next to their name.
Autocomplete Delay	A feature designed for the search bar when a user is typing. The delay refers to the suspending action while the user is typing, making the request to the server for autocomplete.
Backend	References the PHP backend page
Back to home button	A button redirecting the user to the homepage.
Back to songs button	A button redirecting the user to the songs list page.
Commonly Used Web Browser	Browsers such as Firefox, Safari, Chrome, Explorer, and Quora which come on mobile phones, tablets and personal computers.
Customer/Client	Dr. William G. Halfond and Sonal Mahajan
GitHub	A web service that provides software version control tools. www.github.com
Stakeholders	The client and the development team
LOC	acronym: for Lines of Code
KSLOC	a metric that stands for: 1,000(K) Source Lines of Code

Desktop Platform	A screen whose width exceeds 560px
Development Team	All of the individuals whose names appear on the cover of this document. These persons have collectively put this document together and will collectively implement the software product described in subsequent sections.
Facebook	Online social network service where the generated word cloud image may be shared amongst users.
FR	Functional Requirement
Google Doc	An online service provided by Google Inc. where an editable document can be accessed and change simultaneously by the members who have been given access to the document. In the case of the development team, google doc is the shared resource which contains the source of this SRS document.
Home Page	The first page of the website visited by the user. It contains the Word Cloud as well as the Search Bar.
Lyrics Page	The third page of the website, it contains the lyrics for one song, which is chosen by the user on the Songs Page. It will have two Navigation Buttons that can take the user to either the Home Page or back to the Songs Page.
Mobile Platform	A screen whose width is less than or equal to 560px
MVC	The Model-View-Controller Software Pattern
Navigation Buttons	Refers to any button that takes the user to previously visited pages of the website.
Design Document	Refers to this document.

Prototype	A small prototype of the software including the barebones of the graphical display. Used during the second meeting with the client, screenshots available in the appendices.
Search Bar	The initial search bar on the first page of the website. Here, users can type in artist or band names to generate a word cloud.
Share Button	The standard, embeddable Facebook share button.
Software or Product	The application software delivered from the supplier to the customer.
Song List	This will be the culmination of all songs found that contain the search word indicated by the user.
Songs Page	The second page of the website. It contains the Song List as well as a Navigation Button back to the Home Page. The user navigates to the Songs page by clicking on a word in the Word Cloud on the Home page.
Submit Button	The button adjacent to the Search Bar. When the user enters an artist name into the Search Bar and is ready to generate the Word Cloud, he or she must click on the Submit Button to begin the process.
add_to_cloud	a boolean variable that represents if the user has pressed the Add to Cloud Button
back_to_cloud	a boolean variable that represents if the user has pressed the Back to Cloud Button
back_to_songs	a boolean variable that represents if the user has pressed the Back to Songs Button
click_word	a boolean variable that represents if the user has clicked a word in the WC

Error Message Visualization State	represents when the user enters an invalid artist name in the Search Bar and presses the Submit Button, causing an error message to appear
Home State	represents when the user first accesses C-Lyrics before a WC is generated on the Home Page
Lyrics State	represents the lyrics of the song that was selected in the Songs Page state and the user being on the Lyrics Page
searchbar_Text	the user's input in the search bar which is limited to alphanumeric characters
select_song	a boolean variable that represents if the user has selected a song from the Songs List Page
share	a boolean variable that represents if the user has pressed the Share button
Song State	represents the user selecting a word from the WC and being on the Songs Page
submit	a boolean variable that represents if the user has pressed the Submit Button
type_artist	a boolean variable that represents if the user typed in a valid artist name to the Search Bar
Word Cloud Visualization State	represents when the user is on the Home Page and a WC is displayed
Supplier	The team developing the product for the customer.
System	The set of machines running the software making it accessible to the user.
User	A person who interacts with C-lyrics software
Word Cloud (WC)	A word cloud (otherwise known as a tag cloud) is, according to the Oxford Dictionary, an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency or importance [2].

1.5 Scope

The intended audiences of this document are future development teams who will maintain or add to this software and the client. All testing code is available through the code base account on version control service Github.com. References to file paths within the repositories contained in the code base are given as needed for further study in each appropriate section.

1.6 Black-Box and White-Box Methodology

The techniques used in black box testing were that the test were administered based on the requirements given by each specification. The developing team examined the functionality of the C-Lyrics system without peering into its internal structures or workings. This was applied to specific levels of software testing, that being unit, system, and acceptance testing [6]. The techniques used in white-box testing were that the test were administered based on the actual code written by the development team. White-box testing tested the internal structures or workings of the C-Lyrics system. These tests needed an internal perspective system, as well as programming skills used to create design test cases. Certain inputs were used to exercise paths through the code and determine the appropriate and expected outputs for a certain function [7]. With these techniques, the overall system and quality of the code was improved in functionality and readability for the user. We decided to use Protractor instead of Cucumber for the black box testing of our system because it came built in to the tools, Yeoman and Angular, that we were using for development. Additionally, Protractor made the testing process easier to test functionality as well as was more easily accessible compared to the later. For white box testing we used both PHP Unit as suggested in class as well as Karma. The PHP Unit was used to test the back end of our system while Karma was used to test the front end. We chose to use Karma in addition to PHP Unit because it, like Protractor, is built in to the tools the development team used for development and was able to provide a more robust framework for testing our code than using PHP Unit alone.

2 Unit Testing

The reason why unit testing was administered to all of the functions below was because they was tested against the requirements specified. The developing team decided to unit test certain functions because the main system was highly dependent upon them to create a robust system that is fully functioning. The majority of these functions were contained in the lyrics javascript code. The

reason being that many aspects of the C-Lyrics system were dependent upon manipulation of the words and lyrics that were given by the API. Therefore, the majority of main functions within the C-Lyrics system were handled in the lyrics javascript document. As a future disclaimer, some of the tests do not pass because we did not meet the requirements met in the first implementation, therefore, there were some errors that could not be caught given the time constraints. Each test below will have a short description of what will be expected from the test and what will be tested based on the function. Additionally, each milestone that will be satisfied will be associated with that test.

2.1 Karma Testing Software

Karma is a testing tool that is used to test web-pages built with AngularJS. It is essentially running several compatibility tests to determine if C-lyrics is running and operating on various web browsers. Each Java Script function has a test suite applied to it. Each suite describes a particular functionality containing several tests. Each test contains several assertion clauses which test the functionalities of the software. The following list describes each test that was run with the Javascript Unit software:

test/spec/services/lyrics.js Test 1: `countFrequency(word, lyrics)` Milestone G.2, H.1 Counts how many times a word will appear in the lyrics array and associates these words to a specific count. Test 2: `selectMostFrequent(words, N)` Milestone G.2, H.1 * Tests if the function returns the top N words from the words array to be selected for the word cloud. Test 3: `getSongsTitle(words)` Milestone H.2 * Tests if the function returns the title of the song that contains the word that is being taken in. Test 4: `formatTop(songs, N)` Milestone G.2, H.1 Tests to see if the function returns the N most frequently occurring words not including stop words. Test 5: `extractWords(songs)` Milestone G.2, H.1 Tests to see if the function will extract all stop words from the list so that they don't appear in the WC. Test 6: `removeDuplicates(words)` Milestone G.2, H.1 Tests to see if the duplicate words that are going to be in the WC are removed. *test/spec/controllers/main.js* Test 7: *main controller functionality* Milestone G, J *Makes sure that the main controller is working properly.

2.2 PHPUnit Testing Software

The PHPUnit software was used to run unit testing on the backend PHP code. These tests were mainly geared towards achieving milestones for the autocomplete functionality and gathering the word cloud data (milestones G.1 and G.2 of the PMP). The following list describes each test that was run with the PHPUnit software:

- Test 1: `testQueryArtist()`

```

INFO [karma]: Karma v0.12.31 server started at http://localhost:8080/_karma_/
INFO [launcher]: Starting browser PhantomJS
INFO [launcher]: Starting browser Firefox
INFO [launcher]: Starting browser Chrome
WARN [watcher]: Pattern "/home/seba-1511/Dropbox/Dev/USC/310/frontend/test/mock/**/*.js" does
not match any file.
INFO [PhantomJS 1.9.8 (Linux)]: Connected on socket 8tL3W6PayxIAHlBEnzy7 with id 43983416
INFO [Chrome 41.0.2272 (Linux)]: Connected on socket H7KclzxRYXB6b4Tmnzy8 with id 82103704
PhantomJS 1.9.8 (Linux) Service: Autocomplete White Box testing should check autocomplete func
tionality FAILED
    TypeError: 'undefined' is not a function (evaluating 'autocomplete.getartist()')
        at /home/seba-1511/Dropbox/Dev/USC/310/frontend/test/spec/services/autocomplete.js
:36
Chrome 41.0.2272 (Linux) Service: Autocomplete White Box testing should check autocomplete fun
ctionality FAILED
    TypeError: undefined is not a function
        at Object.<anonymous> (/home/seba-1511/Dropbox/Dev/USC/310/frontend/test/spec/serv
ices/autocomplete.js:36:38)
Chrome 41.0.2272 (Linux) Service: Lyrics White box testing of Lyrics service should test forma
tTop function and return the N most common words FAILED
    Expected [ Object({ text: 'Ill', weight: 3 }), Object({ text: 'sun', weight: 6 }) ] to
equal [ Object({ text: 'In', weight: 3 }), Object({ text: 'sun', weight: 6 }) ].
        at Object.<anonymous> (/home/seba-1511/Dropbox/Dev/USC/310/frontend/test/spec/serv
ices/lyrics.js:143:39)
PhantomJS 1.9.8 (Linux): Executed 20 of 20 (1 FAILED) (0.125 secs / 0.233 secs)
Firefox 36.0.0 (Ubuntu) Service: Autocomplete White Box testing should check autocomplete func
tionality FAILED
    TypeError: autocomplete.getartist is not a function in /home/seba-1511/Dropbox/Dev/USC
/310/frontend/test/spec/services/autocomplete.js (line 36)
    @/home/seba-1511/Dropbox/Dev/USC/310/frontend/test/spec/services/autocomplete.js:36:25
PhantomJS 1.9.8 (Linux): Executed 20 of 20 (1 FAILED) (0.125 secs / 0.233 secs)
Chrome 41.0.2272 (Linux): Executed 20 of 20 (2 FAILED) (0.3 secs / 0.285 secs)
Firefox 36.0.0 (Ubuntu): Executed 20 of 20 (1 FAILED) (0.223 secs / 0.207 secs)
TOTAL: 4 FAILED, 56 SUCCESS
Warning: Task "karma:unit" failed. Use --force to continue.

Aborted due to warnings.

Execution Time (2015-03-11 11:36:31 UTC)
concurrent:test      3.5s  ██████████ 36%
autoprefixer:dist    350ms  ██████ 4%
karma:unit           5.8s  ████████████████████ 60%
Total 9.7s

seba-1511@ux32vd-linux:~/Dropbox/Dev/USC/310/frontend$

```

Figure 1: Karma Unit Tests Results

- Milestone G.1
 - Ensures that, given a correct artist name, the autocomplete returns results that are not empty
- Test 2: testQueryArtistSize()
 - Milestone G.1
 - Given a correct artist name, ensures that autocomplete returns the correct number of results
- Test 3: testQueryArtistError()
 - Milestone G.1
 - Given an invalid input, ensures that autocomplete returns null instead of incorrect data
- Test 4: testGetApiKey()
 - Milestone G.1
 - Testing to make sure the EchoNest API key is correct and not empty
- Test 5: testGetClient()
 - Milestone G.1
 - Ensures that a client connection to the databases can be established using the API key
- Test 6: testGetArtistApi()
 - Milestone G.1
 - Checks that a client connection to the artist API is valid
- Test 7: testGetArtist()
 - Milestone G.2
 - Ensures that, given an artist name, the webscraper receives the right artist name
- Test 8: testGetUrlList()
 - Milestone G.2
 - Confirms that gathering URL's of songs for a given valid artist name functions properly
- Test 9: testGetUrlListSize()
 - Milestone G.2
 - Given a valid artist name, ensures that the webscraper returns the correct number of URL links for songs
- Test 10: testGetUrlListEmpty()
 - Milestone G.2
 - Given an empty artist name, ensures that the webscraper returns zero URL links

- Test 11: testGetUrlListFaultyInput()
 - Milestone G.2
 - Given faulty artist name input, ensures that the webscraper returns zero URL links (no bad outputs)
- Test 12: testGetLyrics()
 - Milestone G.2
 - Given a valid artist name, asserts that the lyrics list will not be empty and equals the correct number of songs
- Test 13: testGetLyricsFaultyInput()
 - Milestone G.2
 - Given faulty artist name input, ensures that the lyrics list contains no results and throws an error code
- Test 14: testGetLyricsEmpty()
 - Milestone G.2
 - Given no artist name input, ensures that the lyrics list contains no results and returns an error message
- Test 15: testScrapeBetween()
 - Milestone G.2
 - Given a HTML page, ensures that the webscraper functions properly and takes the correct data from the site

```
c:\Users\markk_000\Desktop\Spring15\CS310\C-Lyrics\backend\backend>phpunit --bootstrap EchoNest.php tests\EchoNestConnectionTest
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.

.....

Time: 1.1 seconds, Memory: 4.25Mb

OK (7 tests, 7 assertions)

c:\Users\markk_000\Desktop\Spring15\CS310\C-Lyrics\backend\backend>phpunit --bootstrap WebScraper.php tests\WebScraperTest
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.

.....

Time: 19.87 seconds, Memory: 4.25Mb

OK (9 tests, 10 assertions)

c:\Users\markk_000\Desktop\Spring15\CS310\C-Lyrics\backend\backend>
```

Figure 2: PHPUnit Tests Results

```
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.  
.....  
Time: 19.67 seconds, Memory: 10.00Mb  
OK (9 tests, 10 assertions)  
  
Code Coverage Report:  
2015-03-30 07:41:22  
  
Summary:  
Classes: 100.00% (1/1)  
Methods: 100.00% (6/6)  
Lines: 100.00% (53/53)  
  
WebScraper  
Methods: 100.00% ( 6/ 6)   Lines: 100.00% ( 53/ 53)
```

Figure 3: Code Coverage, part 1

2.3 Measures and Coverage

The tests run using Karma software sufficiently cover all of the functionalities for the JavaScript frontend. Only certain functions were tested since the majority of the code calls functions without specific testable return output. The testing code written actively covers these functions by ensuring that larger, encompassing functionalities are working properly.

The PHPUnit tests that were run have sufficiently explored each necessary function of the PHP backend. By checking how the code processes various types of inputs (valid, faulty, and empty) as well as checking the communication of different components of the code, we can be sure that the code will not provide functional errors in the processes of generating autocomplete results as well as pulling song lists and lyrics from lyric websites. In this test, we have run 15 tests to measure the robustness of the code. All of them have executed without errors, providing sufficient evidence that the PHP code runs properly.

XDebug along with PHPUnit were used to measure code coverage for the backend. The first set of tests, which tests the web scraper functionality, returns 100% coverage in regards to the class itself, lines of code, and methods. The second set of tests, which tests the EchoNest API connection and transfer of autocomplete results, returns 71.43% coverage. The coverage test itself produces other coverage results, but those results result from using a 3rd party class that

```
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.
.....

Time: 19.67 seconds, Memory: 10.00Mb

OK (9 tests, 10 assertions)

PHPUnit 4.5.0 by Sebastian Bergmann and contributors.
.....

Time: 5.45 seconds, Memory: 13.00Mb

OK (6 tests, 6 assertions)

Code Coverage Report:
 2015-03-30 07:41:55

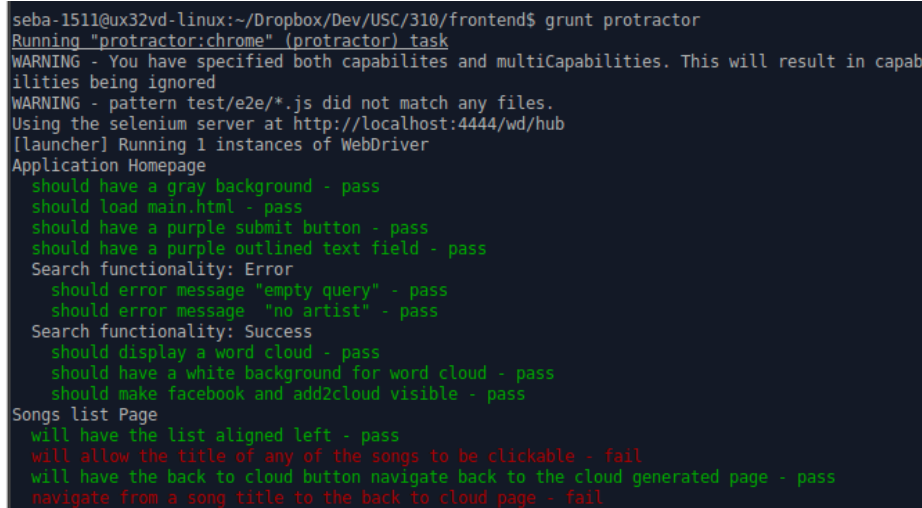
Summary:
  Classes:  0.00% (0/8)
  Methods: 28.79% (19/66)
  Lines:   37.68% (107/284)

@EchoNest.Api::EchoNest_Api_Artist
  Methods:  4.76% ( 1/21)  Lines:  1.94% ( 2/103)
@EchoNest.HttpClient::EchoNest_HttpClient_Curl
  Methods: 25.00% ( 1/ 4)  Lines: 79.63% ( 43/ 54)
@EchoNest::EchoNest_Api
  Methods: 50.00% ( 3/ 6)  Lines: 63.64% ( 7/ 11)
@EchoNest::EchoNest_Autoloader
  Methods:  0.00% ( 0/ 2)  Lines: 55.56% ( 5/ 9)
@EchoNest::EchoNest_Client
  Methods: 33.33% ( 5/15)  Lines: 40.43% ( 19/ 47)
@EchoNest::EchoNest_HttpClient
  Methods: 50.00% ( 4/ 8)  Lines: 64.52% ( 20/ 31)
EchoNestConnection
  Methods: 71.43% ( 5/ 7)  Lines: 68.75% ( 11/ 16)
```

Figure 4: Code Coverage, part 2

connects to the API. It is assumed that the 3rd party class has properly tested the class before public release, so while parts of the API code are not covered in these tests, it's functionality should already be at 100%.

3 Acceptance Testing



```
seba-1511@ux32vd-linux:~/Dropbox/Dev/USC/310/frontend$ grunt protractor
Running "protractor:chrome" (protractor) task
WARNING - You have specified both capabilities and multiCapabilities. This will result in capabilities being ignored
WARNING - pattern test/e2e/*.js did not match any files.
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Application Homepage
  should have a gray background - pass
  should load main.html - pass
  should have a purple submit button - pass
  should have a purple outlined text field - pass
Search functionality: Error
  should error message "empty query" - pass
  should error message "no artist" - pass
Search functionality: Success
  should display a word cloud - pass
  should have a white background for word cloud - pass
  should make facebook and add2cloud visible - pass
Songs list Page
  will have the list aligned left - pass
  will allow the title of any of the songs to be clickable - fail
  will have the back to cloud button navigate back to the cloud generated page - pass
  navigate from a song title to the back to cloud page - fail
```

Figure 5: Protractor Acceptance Tests Results

3.1 Protractor Testing Environment

Protractor is a testing framework used to run end-to-end type tests for AngularJS implementations. These tests will be used as acceptance tests since they reveal if the user defined requirements as described in the SRS are met. The nature of Protractor code makes it difficult to state tests by function name as was done in the previous section. To compensate, the criteria for each unit tests are listed below, the path to each test on the Github repository is also given for further reference.

The procedure to implement acceptance tests was to have a test for each requirement defined in the SRS. Once that mapping was done, tests were created so as to simulate actions a user would perform and ensure that the application always provided a decent rendering. This was accomplished by clicking on buttons, submitting text to input fields. After each one of these actions tests expect a predefined element to exist, to have a certain color or a given shape. Some of those tests currently do not pass for unimplemented functionalities.

- Main Page Layout (frontend/test/spec/controller/main.js)
 - Milestone G
 - Test that the first page when opened has search bar
 - Grey background
 - Submit button purple
 - Text field outlined in purple
- Search (frontend/test/spec/controller/main.js)
 - Milestone G.2
 - If input artist name that doesn't exist
 - * Error "Artist not found"
 - * Empty submit error
 - * Facebook and Add to Cloud not on screen
 - If artist's name correct
 - * Word cloud appears
 - * White background appears
 - * "Add to Cloud" appears
 - * "Facebook Share" appears
- Autocomplete (frontend/test/spec/controller/main.js)
 - Milestone G.1
 - After pause in user's input to search bar autocomplete field becomes visible
 - Test scroll bar
 - Pictures for each artist
- Word Cloud (frontend/test/spec/controller/main.js)
 - Milestone G.2
 - Word size depends on frequency
 - Words multicolored
 - Words link to songs list page
 - Stop words filtered out (ex. "it," "the," and "a")
 - Word cloud generation is 10 sec - 1 min
- Song List (frontend/test/spec/controller/songlist.js)
 - Milestone H
 - Songs sorted from highest frequency to lowest
 - Ability to select a song title to go to lyrics page
 - List will be aligned left
 - Title is clickable
 - Navigation buttons work
 - * "Back to Cloud"

- Lyrics (frontend/test/spec/controller/songlyrics.js)
 - Milestone I
 - Selected word highlighted
 - Lyrics will be aligned left
 - Navigation buttons work
 - * “Back to Cloud”
 - * “Back to Songs”

3.2 Measures and Coverage

There is one black-box test for every requirement. One test is enough since there are a set number of pages and functions (buttons) that the user has access to. The tests will figure out for example, if the add to cloud button works. Adding lyrics to the cloud is a requirement, the test will check if the connection is made. The correctness of what actually shows up in the cloud once new lyrics are added is a white-box testing problem. White-box testing concerns are handled in the previous section of this document. Therefore, our testing policy assumes that as long as each customer requirement is tested at least once, we have achieved satisfactory test coverage.

Furthermore, the development team does not have enough resources to test each requirement more than once. The product will also not be delivered on time if a more comprehensive testing regimen is applied for acceptance testing procedures.

4 Quality Testing

4.1 Reliability Tests

In the SRS we stated that we would test the reliability of our product based on the percent accuracy over 1500 searches, the minimum required percent being 95 percent and the goal being 98 percent. During testing we decided that it was not feasible and unnecessary for us to run the program 1500 times in order to gauge the reliability of our system given time and resource constraints. Instead, we tested our system by searching for distinct 15 artists across different genres running each search 3 times to ensure that it is reliably accurate. Because we are grabbing all of our artist and lyric information from the API, it can be assumed that our system will also work for all remaining artists in the API's database.

4.2 Availability Tests

The SRS states that we will test the availability of our system by the percentage of time that it is available out of 80 hours of monitoring the system, 95 percent of

the time being the minimum required and 97 percent of the time being the goal. This was simulated by having each of the six members of the development team monitor the system for roughly 13 hours adding up to a total of 80 hours that the system was monitored. The system was not only monitored using different OSs but also using different web browsers such as Chrome, Firefox and PhantomJS. We feel that this is sufficient proof that our system will be available as needed and that this test also fulfills all application testability requirements.

4.3 Other Quality Tests

Other forms of quality testing such as maintenance tests and security tests were not administered because they did not fit the needs of this project. Because there is no projected maintenance for this project and there are no foreseeable security risks, we feel that it is appropriate to omit them.

5 Testing Compliances with PMP

5.1 Quality Assurance Testing

For quality assurance purposes, deviations from the PMP were noted in section 3.3.3 and 3.3.4 of the Implementation document. Due to changes in delivery schedule, we did not have the necessary time to achieve full coverage of the test cases.

5.2 Risk Monitoring

Risk monitoring was conducted regularly at all phases of testing. To ensure that the maximum number of EchoNest queries was not met during testing, we made sure to write test cases that were robust and covered multiple features whenever possible. This was the main consideration that was marked out in the PMP. All general risk monitoring practices that were outlined (such as risks not related specifically to implementation) were continually met throughout the testing phase of the project.

6 Appendices

6.1 Declaration of Milestones

The milestones below were conceived for and reported on the PMP document. The below table is an excerpt from the original PMP document. They are reproduced here for clarity and reference.

- Testing and Final Delivery. 3/11/15*
- Milestones G-J
 - G: Testing of the Home Page
 - * G.1: Search bar with autocomplete functionality when typing in an artist's name
 - * G.2: WC generation with words that can be selected to take the user to the Songs Page
 - * G.3: Share button to upload the WC to Facebook
 - * G.4: Add to Cloud button to create a new WC based off of words commonly used by both of the specified artists
 - H: Testing of the Songs Page
 - * H.1: List of songs sorted by how frequently the selected word is used in each song
 - * H.2: Song titles in list able to be selected, taking the user to the Lyrics page
 - * H.3: Back to Home button takes the user back to the Home Page with the WC still displayed and the artist's name still in the Search Bar
 - I: Testing of the Lyrics Page
 - * I.1: Lyrics displayed on page with the selected word highlighted every time it appears in the song
 - * I.2: Back to Songs button takes the user back to the Songs Page with the same list of songs still displayed in the same order
 - * I.3: Back to Home button takes the user back to the Home Page with the WC still displayed and the artist's name still in the Search Bar
 - J: Testing of the entire product to ensure all pages work together as specified in the SRS

6.2 Deviations From PMP

6.2.1 Deviations from Schedule

Originally, milestones were organized according to the table below, where each letter corresponds to a mile stone in each phase of the Waterfall software development cycle. The Milestones of interest are letter G-J. A description of milestones G-J can be found in the previous appendix 5.1. Significant deviation has occurred in project schedule cause by changed deliverable due dates.

The client pushed up the deadline for the testing document delivery to Wednesday March 11 2015. Therefore, milestones G-J should be extended until that date in the image above, representing the change in our PMP.

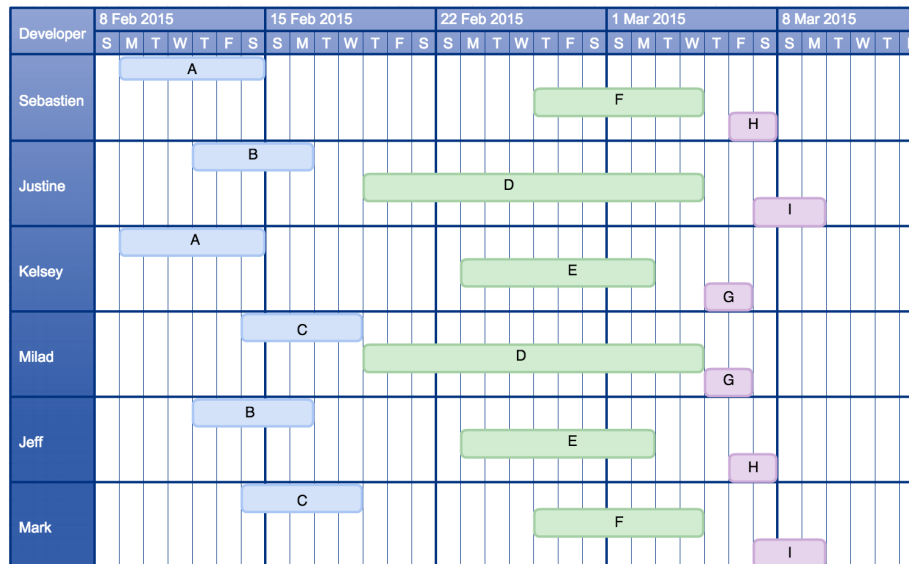


Figure 6: Staff Allocation

6.2.2 Deviations From Milestone Planning

Team member Milad Gueramian was unable to contribute to testing activities due to loss of development equipment (laptop computer). Therefore, he was unassigned from milestone G and was given the task of completing and preparing the testing document. A milestone was created for this task in the Github Issue tracker tool which, as described in the PMP, is use to coordinate and keep track of development team activities.

Furthermore, team member Jeff Kang was also reassigned to create the testing document with Milad Gueramian as the difficulty and resource usage of this task was not considered in the PMP.

6.3 Screenshots

Here are the screenshots of the application as tested in a mobile browser and a desktop one.

