Cumulyrics

Testing Document

Team 19

Kyle Van Landingham, Alec Schule, Alec Fong, Andrew Zolintakis, Noah Bergman

1. Executive Summary

This document thoroughly documents the test case generation process for both white box and black box testing. This document contains sections for the black box testing process, white box testing process, and overall testing process. The white box and black box testing sections describe the test case generation process, and contain the test cases generated by that process. The overall testing process section details the methods used by team 19 to carry out the testing phase. This includes how the team adhered to the Project Plan, especially risk analysis and quality assurance metrics.

2. Overview

2.1 Purpose

The purpose of this document is to verify the implementation of the system against the requirements. This verification comes in the form of black box testing which tests the overall functionality of the system based on the SRS and white box testing which tests the system code on a unit level. This document is meant to describe the motivations for each test case that is applied the the AUT and the process that was followed throughout the testing phase of the project lifecycle.

2.2 Intended Audience

The intended audience for this document include the team members of the project who can refer to this document for future project planning and the other stakeholders to verify how well the system meets the requirements in the SRS.

2.3 References

Reference	Relevant Information
https://github.com/kvanland/csci310Team19	Link to the project repository

2.4 Definitions

Term	Definition
SRS	System Requirements Specification
AUT	Application Under Test

Gherkin	Business Readable, Domain Specific language that lets you describe software's behaviour without detailing how that will be implemented

3. Black Box Testing

3.1 Black Box Test Case Generation Process

Black box test cases were generated based on the SRS. A feature file was written for each of the main features in the SRS, and each feature had numerous scenarios testing all requirements. Due to the design of the implemented system, somewhat atypical Gherkin language was used. Where a more common design might have a line of Gherkin stating "Given I am on /word-cloud.php", this would not work for our system. All features are shown on a single page and shown and hidden through the use of javascript. Due to this, out version of the previous statement would be "Given I am on 'Drake' word cloud", and the implementation of the corresponding function in FeatureContext.php would manually fill in the search bar, and click to get to the desired page.

The generated features and scenarios are listed below, in Gherkin language, for the reader's convenience. In some cases, there are small deviations from the language in the .feature files to enhance the readability of this document. Above each feature and scenario, the requirements that generated that scenario will be listed, along with explanatory text where needed.

3.1.1 Autocomplete

Feature: Auto Complete
In order to choose an artist
As a website user
I need to be able to choose from a list of valid artists

3.2.1.3 REQ-1
Scenario: No input
Given I am on "localhost"
When I do nothing
Then The "searchBar" value should be ""

3.2.1.3 REQ-2
Scenario: Select the searchBar
Given I am on "localhost"

And "searchBar" is visible
When I click on the "searchBar"
Then The "searchBar" should be editable

3.2.1.3 REQ-6
Scenario: Typing an artist that does exist
Given I am on "localhost"
When I fill in "searchBar" with "Drake"
Then I should see an image of "Drake" to the left of "Drake"

Scenario: Typing an artist that does not exist and shares no prefix with an valid artist
Given I am on "localhost"
When I fill in the "searchBar" with "Wamo Blamo Slamo"
Then I should not see any suggestion

3.2.1.3 REQ-5
Scenario: Typing 3 characters of a valid artist prefix
Given I am on “localhost”
When I fill in the “searchBar” with “Dra”
Then I should see at least 3 suggestions for artists containing “Dra”

3.2.1.3 REQ-4
Scenario: Typing less than 3 characters
Given I am on "localhost"
When I fill in the "searchBar" with "Dr"
Then I should not see any suggestion

3.2.1.3 REQ-7
Scenario: Selecting a suggested artist
Given I am on "localhost"
When I fill in the "searchBar" with "Drake"
And I click on the suggestion containing an image of "Drake" and "Drake"
Then The "searchBar" value should be "Drake"

Note: The team was unable to implement the automated testing for this feature due to time constraints. However, the team did design valid and sufficient test cases for the autocomplete feature before attempting to implement the testing frameworks.

3.1.2 Search

Feature: Search
In order to get correct information
As a website user
I need to be able to search by specific artists

3.2.2.3 REQ-1
Scenario: Search with empty search box
Given I am on the home page
When I do nothing
Then I should not be able to click search

3.2.2.3 REQ-2 REQ-3
Scenario: Search valid artist on home page
Given I am on the home page
When I type “drake”
And I select the suggestion “Drake”
Then I should be able to click search
And the wordcloud should be displayed with title “Drake”

3.2.1.3 REQ-2, REQ-4, REQ-7
Scenario: Search valid artist without fully typing
Given I am on the home page
When I type “drak”
And I select the suggestion “Drake”
Then I should be able to click search
And the wordcloud should be displayed with title “Drake”

Scenario: Search for valid artist while displaying wordcloud
Given I am on the wordcloud page
And I type “yung lean”
And I select the suggestion “Yung Lean”
Then I should be able to click search
And the wordcloud should be displayed with title “Yung Lean”

3.2.2.3 REQ-1
Scenario: Valid search term without selecting autocomplete
Given I am on the home page

When I type “CHVRCHES” into the search bar
Then I should not be able to click search

3.2.2.3 REQ-1
Scenario: Incomplete search term without selecting autocomplete
Given I am on the home page
When I type “Man” into the search bar
Then I should not be able to click search

3.2.2.3 REQ-1
Scenario: Invalid Search Term
Given I am on “localhost”
When I type “xysfdg” into the search bar
Then I should not be able to click search

3.1.3 Add to cloud

Feature: Merge
In order to view more than one artist at once
As a website user
I need to be able to merge specific artists

3.2.5.3 REQ-1
Scenario: Merge with empty search box
Given I am on the home page
When I do nothing
Then I should not be able to click add to cloud

3.2.5.3 REQ-3
Scenario: Merge valid artist on home page
Given I am on the home page
When I type “drake”
And I select the suggestion “Drake”
Then I should be able to click add to cloud
And the wordcloud should be displayed with title “Drake”

3.2.5.3 REQ-3
Scenario: Merge valid artist without fully typing
Given I am on the home page
When I type “drak”

And I select the suggestion “Drake”
Then I should be able to click add to cloud
And the wordcloud should be displayed with title “Drake”

3.2.5.3 REQ-2
Scenario: Merge valid artist while displaying wordcloud
Given I am on the wordcloud page
And I type “yung lean”
And I select the suggestion “yung lean”
Then I should be able to click add to cloud
And the wordcloud should be displayed with title “Yung Lean”

3.2.5.3 REQ-1
Scenario: Merge valid artist without selecting autocomplete
Given I am on the home page
When I type “CHVRCHES”
Then I should not be able to click add to cloud

3.2.5.3 REQ-1
Scenario: Merge incomplete search term without selecting autocomplete
Given I am on the home page
When I type “Man”
Then I should not be able to click add to cloud

3.2.5.3 REQ-1
Scenario: Merge Invalid Search Term on Home
Given I am on “localhost”
When I type “xysfdg”
Then I should not be able to click add to cloud

3.2.5.3 REQ-1
Scenario: Merge Invalid Search Term while displaying wordcloud
Given I am on “localhost”
When I type “xhgla”
Then I should not be able to click add to cloud

3.1.4 Song list

Feature: Song List

In order to see more information about an artist
As a website user
I need to see the number of occurrences of a word in all of my chosen artists' songs

3.2.8.3 REQ-1, REQ-2
Scenario: Artist "Bleachers" with word "gave"
Given I am on "localhost"
When I search for "Bleachers"
And I click on the word "gave"
Then I should see the songs Shadow with a frequency of 2 and I Wanna Get Better with frequency of 1

3.2.7.3 REQ-1
Scenario: Artist "Drake" with a random word
Given I am on “localhost”
When I search for "Drake"
And I click on a word in the wordcloud
Then I should see the word I selected on the top of the page

3.2.8.3 REQ-2
Scenario: Artist "Kanye West" with word "go"
Given I am on "localhost"
When I search for "Kanye West"
And I click on the word "go"
Then I should see a list of songs in descending ordered by frequency

3.1.5 Lyric

3.2.9.3 REQ-1 REQ-2
Lyrics Title:

To verify the lyrics title was being displayed with the right format, it was sufficient to check the ‘title’ html tag upon reaching the correct page. This was tested with a representative sample of artist/song pairs. Note this requirement was not implemented correctly, so all such tests fail.

Example test:
Scenario: “Breakaway by Kelly Clarkson” title check
Given the current page is "Kelly Clarkson" word cloud
When I click "fly"
And I click "Breakaway"

Then the page title is "Breakaway" by "Kelly Clarkson"

3.2.8.3 REQ-4, REQ-5
3.2.1.3 REQ-1
Correctness of lyrics:

To verify the correctness of lyrics on the lyrics page, it was necessary to come up with some kind of oracle mechanism that provided “correct lyrics”. Due to limited time, the lyrics to two different songs were obtained from a different source that the application, and hard coded in the testing code for verification purposes.

Scenario: Displayed lyrics for “Breakaway” are correct
Given the current page is “Kelly Clarkson” word cloud
And I click “fly”
And I click “Breakaway”
Then the correct lyrics for “Breakaway” are displayed on the page

Scenario: Displayed lyrics for “Recover” are correct
Given the current page is “CHVRCHES” word cloud
And I click “number”
And I click “Recover”
Then the correct lyrics for “Recover” are displayed on the page

3.2.1.3 REQ-2
Word highlighting:

To test whether words were correctly highlighted, check the html attributes of each instance of a given word in the lyrics to make sure color is set to yellow. Tests were performed on a representative sampling of word/song pairs.

Example test:
Scenario: All instances of “number” in “Recover” highlighted
Given the current page is “CHVRCHES” word cloud
And I click “number”
And I click “Recover”
Then all instances of “number” on the page are highlighted.

3.1.6 Facebook

Due to limitations with the standalone browser instance used by Selenium, tests for “Share to Facebook” requirements were written under the assumption that the user is logged out. This could have been corrected had time permitted.

3.2.6.3 REQ-3
Logging in gracefully:
First check whether a login popup window appeared upon clicking the share button.
Scenario: Test facebook login window appears
Given the current page is "Of Monsters and Men" word cloud
When I click "Share to Facebook"
Then there is a popup titled "Log in With Facebook"

3.2.6.3 REQ-1
Facebook create post page:
Upon logging in, user is redirected to Facebook, on a “create post” page. This was not implemented correctly, so all tests for this requirement fail
Scenario: Check facebook post
Given the current page is "Of Monsters and Men" word cloud
When I click "Share to Facebook"
And I login
Then I am shown a filled-out facebook post

3.2.6.3 REQ-2
Facebook post content:
Check the content of the Facebook post to ensure it includes an image of the word cloud and text listing the names of artists in the word cloud. This was untestable due to Facebook login not working, and so tests were not written due to time constraints.

3.1.7 Home button

Feature:
In order to navigate back from other pages
As a website user
I should be able to navigate to the Word Cloud page or Artist Search page

Scenario: Home from on load page
Given I am on "localhost"
When I click the home button

Then Nothing should occur

Scenario: Home from the Word Cloud page
Given I am on "localhost"
When I search for "Drake"
And I click the home button
Then I should be navigated to the Artist Search Page

3.1.1.3
Scenario: Home from the Song List Page
Given I am on "localhost"
When I search for "Drake"
And I click the word "ever"
And I click the home button
Then I should be navigated to the Word Cloud Page

3.1.1.4
Scenario: Home from the Lyrics Page
Given I am on "localhost"
When I search for "Drake"
And I click the word "ever"
And I click on the song "Forever"
And I click the home button
Then I should be navigated to the Word Cloud Page

3.1.8 Back button

Feature: Back
In order to navigate back from other pages
As a website user
I should be able to navigate to the previous page

Scenario: On home page
Given I am on the home page
Then the back button should be invisible

Scenario: Back on Word Cloud Page
Given I am on the wordcloud page
When I click the back button
Then I should be on the home page

3.1.1.3
Scenario: Back on Song List Page
GIven I am on the song list page
When I click the back button
Then I should be on the wordcloud page
And the artists should be the same as before

3.1.1.4
Scenario: Back on Lyric Page
Given I am on the lyric page
When I click the back button
Then I should be on the song list page
And the songs should be the same as before

3.2 Black Box Validation and report of coverage

The black box tests listed in the previous section are marked with the related requirements that are tested by each scenario. Each functional requirement and certain non-functional requirements are validated by the listed scenarios. Due to the test case generation process, these tests completely cover the SRS requirements. It was intended to test for a wide sampling of cases across each requirements, but this was not fully met due to time constraints. Each requirement was still tested with a representative sampling, and due to the use of Gherkin, many more tests can be somewhat easily generated from these base tests.

4. White Box Testing

4.1 White Box Test Case Generation Process

4.1.1 ArtistSuggestions.php

Test Case: Pass into getSuggestions() a string that begins at least one artist's name(test that main functionality works)
Input: String: "Dra";
Expected Output: {"Drake"=>
"https://lastfm-img2.akamaized.net/i/u/174s/b4310b8ad99f2b5930b36725bb7deb36.png",
"Dragonette" =>
"https://lastfm-img2.akamaized.net/i/u/174s/20bb4ca71fbf4991990cacec6074a90a.png",
"Draper"=>
"https://lastfm-img2.akamaized.net/i/u/174s/14511d935336413883d47e60e55422a4.png",

“DragonForce”=>
“https://lastfm-img2.akamaized.net/i/u/174s/fbbdb94a245043edb56b0f50e413d565.png”,
“Dramarama”=>
“https://lastfm-img2.akamaized.net/i/u/174s/a384389858214653b60464c57b72c229.png”}
Reason: The main functionality of this class is to get a list of artist names that begin with a specified string. This test tests whether or not it will return only artist names that start with the specified name.

Test Case: Pass into getSuggestions() all lower case string that begins at least one artist’s name
Input: String: “dra”;
Expected Output: {“Drake”=>
“https://lastfm-img2.akamaized.net/i/u/174s/b4310b8ad99f2b5930b36725bb7deb36.png”,
“Dragonette” =>
“https://lastfm-img2.akamaized.net/i/u/174s/20bb4ca71fbf4991990cacec6074a90a.png”,
“Draper”=>
“https://lastfm-img2.akamaized.net/i/u/174s/14511d935336413883d47e60e55422a4.png”,
“DragonForce”=>
“https://lastfm-img2.akamaized.net/i/u/174s/fbbdb94a245043edb56b0f50e413d565.png”,
“Dramarama”=>
“https://lastfm-img2.akamaized.net/i/u/174s/a384389858214653b60464c57b72c229.png”}
Reason: This test checks to make sure an all lower case string does not affect the suggestions.

Test Case: Pass into getSuggestions() all upper case string that begins at least one artist’s name
Input: String: “DRA”;
Output: {“Drake”=>
“https://lastfm-img2.akamaized.net/i/u/174s/b4310b8ad99f2b5930b36725bb7deb36.png”,
“Dragonette” =>
“https://lastfm-img2.akamaized.net/i/u/174s/20bb4ca71fbf4991990cacec6074a90a.png”,
“Draper”=>
“https://lastfm-img2.akamaized.net/i/u/174s/14511d935336413883d47e60e55422a4.png”,
“DragonForce”=>
“https://lastfm-img2.akamaized.net/i/u/174s/fbbdb94a245043edb56b0f50e413d565.png”,
“Dramarama”=>
“https://lastfm-img2.akamaized.net/i/u/174s/a384389858214653b60464c57b72c229.png”}
Reason: This test checks to make sure an all uppercase string does not affect the suggestions.

Test Case: Pass into getSuggestions() a string that begins more than 5 artists’ names
Input: String: “bea”

Expected Output: Array: {“Beach House”=>
”https://lastfm-img2.akamaized.net/i/u/174s/7fca68251e01485eb46f2de4acb712da.png”,
"Beastie Boys"=>
"https://lastfm-img2.akamaized.net/i/u/174s/48bebf1d288f47bdb02a3436a90e0d9d.png",
"Beach Fossils"=>
"https://lastfm-img2.akamaized.net/i/u/174s/419d30d6c2aa42cfa46a86ad9e3862f0.png",
"Bear's Den"=>
"https://lastfm-img2.akamaized.net/i/u/174s/ffca5f059f574ce993531ac754f0816f.png"
"Bear Hands"=>
"https://lastfm-img2.akamaized.net/i/u/174s/fd64c055788d4eadcec865dcf4253e5c.png"}
Reason: The file is only supposed to return the top 5 artist suggestions. This test checks to make sure it doesn’t return more than 5 artists if there exists more than 5 artists that begin with the specified string.

Test Case: Pass into getSuggestions a string that begins an artist name, which contains more than one word
Input: String: “Kan”
Expected Outout: Array: {"Kanye West"=>
"https://lastfm-img2.akamaized.net/i/u/174s/2377da54b28a610cb20cbc2b9c5a7517.png",
"Kansas"=>
"https://lastfm-img2.akamaized.net/i/u/174s/ba357970b9ef42adbc2f5f4af2417ee5.png",
"Kano"=>
"https://lastfm-img2.akamaized.net/i/u/174s/89d80e0b29e04cc187925683b3c6f6cc.png",
"Kan Wakan”=>
"https://lastfm-img2.akamaized.net/i/u/174s/d44149116bf94613c6decfcfaeb3a265.png",
"Kangding Ray"=>
"https://lastfm-img2.akamaized.net/i/u/174s/91315f92a29d4b3a9e0b27274d68c014.png"}
Reason: This test checks to make sure an artist name that contains more than one word can be returned, and that both words are returned.

Test Case: Pass into getSuggestions() a string that is the full name of an artist
Input: String: “Kanye West”
Expected Output: Array: {"Kanye West"
=>"https://lastfm-img2.akamaized.net/i/u/174s/2377da54b28a610cb20cbc2b9c5a7517.png"}
Reason: This test checks to make sure that if the specified string is an artist’s full name, that artist name is returned.

Test Case: Pass into getSuggestions a string that contains two words and begins the name of at least one artist

Input: String: “The Bea”
Expected Output: Array: {"The Beatles"
=>"https://lastfm-img2.akamaized.net/i/u/174s/6c9cdd81b0ba4c198d47d13f200bc843.png",
“The Beach boys"
=>"https://lastfm-img2.akamaized.net/i/u/174s/f902c00cdd234924a38c151e7684cfbb.png",
“The Beatnuts”
=>"https://lastfm-img2.akamaized.net/i/u/174s/cefe72c5d5f94f4fab5baacf09119285.png",
“The Beau Brummels”
=>"https://lastfm-img2.akamaized.net/i/u/174s/da2157bfef9d4bff49dd7786bc05c8df.png"}
Reason: This tests checks to make sure that if the specified string contains two words, it will still return the artist names that start with that string.

4.1.2 DatabaseAccessor.php

Test Case: Call the DatabaseAccessor constructor
Input: nothing
Expected Output: Nonnull DatabaseAccessor
Reason: This test checks to makes sure the DatabaseAccessor connects to the database.

Test Case: Pass in an existing artist name to getArtistID()
Input: String: “Rihanna”
Expected Output: int: 7;
Reason: This test tests the main functionality, which is returning the correct artistID from the database, corresponding to the given artist name.

TestCase: Pass in an artist name that does not exist
Input: String: “Brodie”
Expected Output: null
Reason: This test checks to makes sure that if an artist doesn’t exist the function returns null.

4.1.3 getSongs.php

Test Case: English word for the word variable and existing artist name for artist variable (Test main functionality)
Input : String: “ever”, String: “drake”
Expected Output:Valid JSON
Reason: This test tests the main functionality of the getSongs file, which is to output a valid JSON when there is a song that contains the specified word and that is sung by the specified artist.

Test Case: Random word for the word variable and a non existing artist
Input: String: “gg”, String: “ogo”;
Expected Output: null;
Reason: This test checks to make sure that the file outputs null if no songs are found in the database that contain the specified word and that are sung by the specified artist.

Test Case: Empty string for the word and artist variables
Input: String: “”, String: “”;
Expected Output: null;
Reason: This test checks to make sure the file outputs null if the word and artist name are empty strings.

Test Case: Empty string for the word variable and existing artist name for the artist variable
Input: String: “”, String: “Drake;
Expected Output: null;
Reason: This test checks to make sure that the file outputs null if the word is blank.

Test Case: English word for the word variable and an empty string for the artist variable
Input: String: “Ever”, String: “”
Expected Output: null;
Reason: This test checks to make sure that the file outputs null if the artist name is blank.

4.1.4 getWordCloud.php

Test Case: Catches empty string
Input: String: “”
Expected Output: null
Reason: This test checks makes sure the file outputs null if artist name is an empty string

Test Case: Returns valid json
Input: String “Drake”
Expected Output: True (json test)
Reason: This test tests the main functionality, which is to output a valid JSON if the artist exists.

Test Case: Multiple artists only results in 250 words.
Input String: “Drake, Coldplay”
Expected Output Size: 250
Reason: This test checks to make sure that the file only outputs 250 words when multiple existing artists are inputted.

Test Case: Multiple artists provided are in descending order
Input: String “Drake,Coldplay”
Expected Output: True (for descending tester)
Reason: This test checks to make sure that the array of words being outputted is in descending order

Test Case: Accurate word count for multiple artists
Input: String “Drake,Coldplay”
Expected Output: You == 70, Would == 8
Reason: This test checks to make sure that the class is giving the correct word count when multiple artists are inputted

4.1.5 SongsFinder.php

Test Case: Returns null if invalid artist
Input: String: “good”, String: “***”
Expected Output: Null
Reason: This test checks to make sure that the class returns null when the artist does not exist.

Test Case: Returns correct array given regular case
Input: String “Drake”, “good”
Output: {“Hold On, We're Going Home” => {6, “Drake}, “Over” => {1, “Drake}}
Reason: This test tests the main functionality, which is to return the correct songs by an artist that contain the word and the correct frequency the word appears in each song. This only applies to words that are actually said by the specified artist.

Test Case: Returns Null if existing artist but no songs contain word
Input: String: “***”, String: “drake”
Expected Output: Null
Reason: This test checks to make sure that the class returns null if the artist exists, but the artist does not ever say the specified word.

4.1.6 WordCloud.php

Test Case: Empty string returns null
Input: String: “”
Expected Value: Null

Reason: This test checks to make sure that the class returns null if an empty string is inputted.

Test Case: Ensure only 250 words are returned
Input: String: “Drake”
Expected Length: 250
Reason: This class tests to make sure the class returns only the top 250 said words said by an artist when the artist has said more than 250 words.

Test Case: Returns list in descending order
Input: String: “Drake”
Expected Output: TRUE (for descending boolean value)
Reason: This test makes sure that the returned array of words is in descending order.

4.1.7 getSuggestion.php

Test Case: String that begins an existing artist name (Test main functionality)
Input : String: “dra”
Expected Output:Valid JSON
Reason: This test tests the main functionality of the getSuggestions file, which is to output a valid JSON when there exists an artist name that begins with the specified string.
Test Case: String that does not begin the name of any artists
Input: String: “xfv”
Expected Output: null;
Reason: This test checks to make sure that if no artist names begin with the specified string, the file outputs null.

Test Case: Empty string
Input: String: “”
Expected Output: null;
Reason: This test checks to make sure the file outputs null if the inputted string is empty.

4.2 White Box Validation and report of coverage

The white box testing the team has done is in attempt to verify that the system was built correctly. These tests test each file to make sure they return or output the right data and also that the data is in the correct format. Each file has multiple tests, designed to test all the different possible scenarios that could occur when the system is running.

The tests created have about 92% code coverage of the backend php code. That means that the tests run 92% of all the backend code. This enables the team to be very confident that there are no errors if all tests pass.

5. Testing Process Description

5.1 Adherence to Project Plan

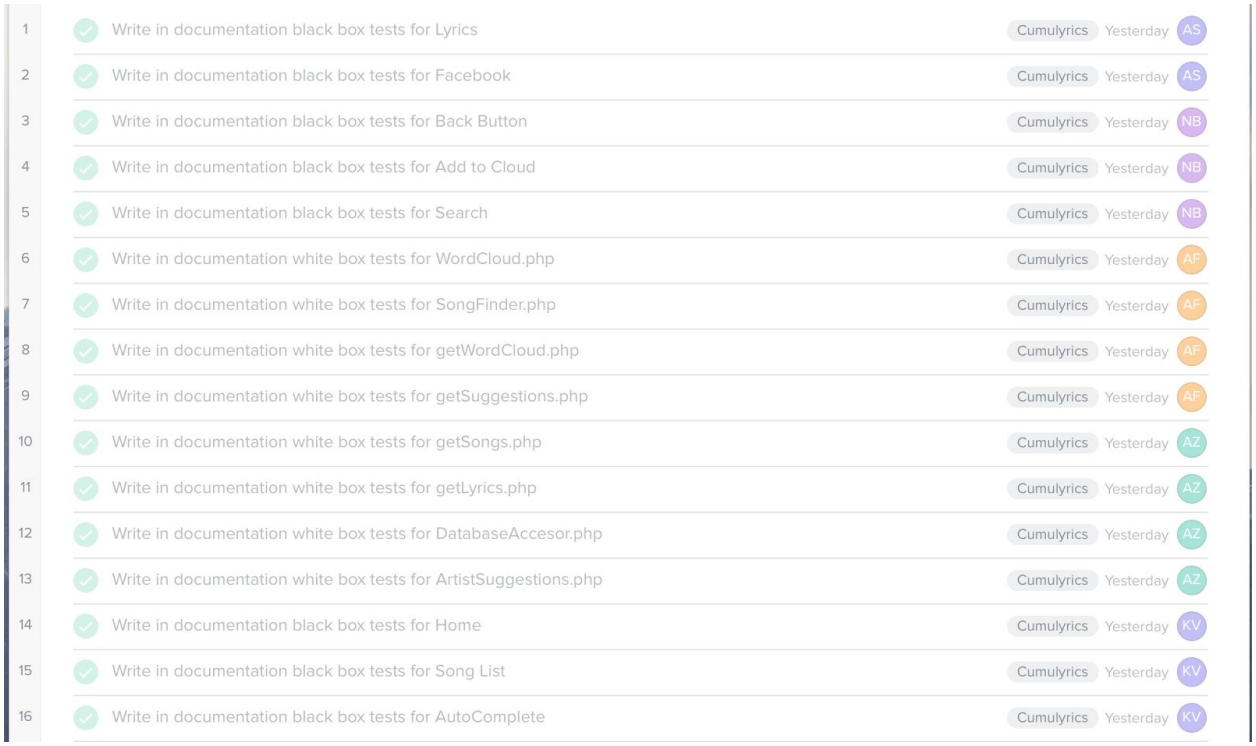
Meeting Log

3/5/17	The group met at VKC 7pm - 9:30pm to discuss the approach for the testing phase. The team met with CPs in SAL for help deciding what testing framework to use.
3/6/17	The group met at VKC 6pm - 7:30pm to assign specific testing assignments and relay information given by the professor with regards to how to test our specific code (eg White box only for php since our javascript had little logic even though it made up 70% of the system code).
3/7/17	The group met at VKC 8:30pm - 9:30pm to work collaboratively on getting Behat, Mink, Selenium2, and phpUnit to work on our virtual machine. At this point all test cases were documented and written in either feature files or php files.
3/8/17	The group met at VKC 7pm- to work collaboratively on getting our tests to run properly on the virtual machine.

Schedule Compliance

According to the schedule established in the Project Management Plan, the team planned to develop the “front-end tests” and “back-end tests” for two to three days and implement the tests for four to five days. This schedule was not followed completely because the 7 day timeline allotted to the testing phase was too constricting to follow with confidence. To explain, it took a lot of deliberation and communication between the team members and staff to fully grasp the steps required to correctly carry out the testing phase. The ordering of the tasks in the schedule remained the same but were just carried out in the much quicker time scale on the order of 4 days.

Asana Compliance



There wasn't much point to assigning implementation tasks because the deadline was the next day when we began to get the implementation of the testing frameworks working and we were all together as a team in person. Asana is mainly used to track tasks when the team is not together in person.

Risk Analysis

During the team meetings risk assessment was conducted before the rest of the work began. Most risks identified in the Project Management Plan related to the implementation phase of the project lifecycle and as a result there were few risks that were assessed at each meeting. Two risks that did occur during the testing phase are a team member having other school work that prevents them from fully contributing to the project and a team member falling ill preventing them from meeting in person. In both of those cases, other team members took on a larger portion of the work demanded by the project and the project manager shifted tasks to accommodate the afflicted team member. The risk avoidance strategy of communicating to the team each team member's personal school schedule was not successful in preventing other work from overwhelming certain team members due to the short amount of time allotted to the testing phase (7 days).

Configuration Management

In accordance with the Project Management Plan, the team used Google Docs to collaboratively develop the documentation for the testing phase. Also in accordance with the

Project Management Plan, the team used Git and Github to organize the code that implemented the testing of the system. As stated in the Project Management Plan, the members of the team created new branches for each task completed and merged branches when the assigned task was complete.

Quality Assurance Plan

The quality assurance plan listed in the Project Plan is ambiguous and therefore difficult to assess. The quality of the tests is determined by how adequately the test cases cover any scenario that may arise from the product use. The rushed nature of the testing phase did not allow for the team to generate a sufficient number of high quality test cases that met the acceptable level of quality described in the Project Management Plan. Another measure of quality for the testing phase included the amount of test cases passed by the implemented system. To reiterate the rushed nature of the testing phase did not allow the team to fix any faults in the code that would have resulted in certain test cases failing.

Fog Index

As stated in the Project Management Plan, the team aims to have documentation with a fog index below 12. The fog index for the current document is 7.435 which is well below the maximum mark of 12 described in the Project Management Plan. The fog index is a measurement of the overall complexity of the language within a document and the lower the number the easier the text is to understand. It is important for this index to be low so stakeholders and developers can both understand and utilize the documentation without difficulty.

5.2 Overall Process Description

The team began the testing phase by identifying the types of testing frameworks that would work best for the team's specific implementation of the system. There was doubt that due to the heavy amount of AJAX and javascript that the traditional testing frameworks behat and phpUnit would not be easily applied to the implemented system. Communication between the team and staff members over the course of two days helped determine that the team would utilize phpUnit for white box testing of the php code and behat, mink, and selenium for black box testing of the overall system. This conclusion was approved by Professor Halfond with the reasoning that even though the implemented system is 70% javascript code and only around 25% php code, most of the logic that would be relevant to white box testing lies in the php server side code.

After solidifying the testing frameworks that were to be used during the testing phase the black box testers were tasked with writing Gherkin features and scenarios in the documentation on Google Docs. Each scenario was required to test some requirement from the SRS or some extra functionality that the specific implemented system had included. The white box testers were tasked with writing a large amount of unit tests for each php file which generally contain

one function. The unit tests were designed to cover as many paths and as many SLOCS as possible given the phpUnit testing framework.

Once the tests were documented, the team began to implement the testing frameworks agreed upon earlier in the phase. There were significant struggles with regards to the black box testing frameworks that were remedied by staff guidance and a large amount of stack overflow searching. Once the black box testing frameworks worked on the virtual machine, writing the implementation for the featureContext.php file was relatively simple and straightforward. White box testing overcame a version incompatibility issue which when resolved allowed the white box testers to finish their implementation significantly more quickly than the black box counterparts.

The testing process was terminated due to time constraints rather than another metric such as rate of finding faults decreasing to acceptable levels. A large contributor to this time constraint was the amount of time it took for human oracles to verify the validity of the system functionality. It took a large amount of time to identify songs for an artist that contained certain words and the frequency of that word in each song. It also took a large amount of time to identify the urls for the images of artists that were to appear in the autocomplete feature and tested in the white box testing.

The team failed to implement the autocomplete feature tests due to time constraint but due to the process followed, the team was able to document the test that were suppose to be implemented during the testing phase. The team also failed to gather appropriate cover metrics and develop cover reports. The team understands that this failure is significant, but the team assessed the failure and chose to get a working system and thorough documentation finished instead of pursuing the coverage report for the black box testing and a more detailed coverage report for the white box testing. Another minor failure on the part of the team was the low amount of faults fixed which results in a good amount of failed black box tests. The reasoning behind not fixing most faults discovered during the testing was the time constraint. If there were a maintenance phase the team would use that phase to address the faults found in the testing phase.

As an aside it became evident that well documented system are important while the team was attempting to become familiar with four different testing tools. Even though the documentation for the tools used during the testing phase are documented, they were not well documented enough that users who have no prior knowledge of the tools or related software could understand how to use those tools without help from outside sources.