Project #1
Software Design

Group #6
Yawen Cao
Xiaoyang (Leo) Liao
Juntao Shen
Jiachang (Ernest) Xu
Mengqian Yu
Lifan Zhao

Executive Summary

This document serves as the software design document for Lyrics Cloud, based on the Software Requirements Specification document provided by Group #17 of CSCI 310 (Software Engineering). This document is composed by two major components: Design Diagrams, and Design Evaluation.

To get started, the Design Diagrams chapter goes through every diagrams necessary for the design of this system. All the diagrams in this document are generated in Unified Modeling Language (UML). This group uses a top-down approach to visualize the design. All the design diagrams are intentionally constructed and arranged in the following order:

- Use Case Diagram
- Data Flow Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram
- Component Diagram
- Class Diagram

In addition, the Design Evaluation chapter evaluates the design from an integrated level for this system, emphasizing design qualities, such as abstraction, modularity, coupling, cohesion, information hiding, simplicity, and hierarchy. This design uses a lot of abstraction. Regarding to modularity, this design possesses much more cohesion than coupling: types of cohesion exhibited in this design include procedure cohesion, function cohesion, temporal cohesion, logical cohesion, and sequential cohesion; types of coupling exhibited in this design only include data coupling, and control coupling. Thus, this design demonstrate a fairly high level of modularity. Due to use of API and construction of different classes, this design shows sufficient amount of information hiding. The Detailed Design section examines the time complexity across this project; it also explains the use relationship among modules.

In the end of this document is Appendix I, which describes this group's design process. Appendix I makes sure that the design process complies with the Project Management Plan (PMP) document, which this group submitted on February 7th, 2017. It has been verified that this design follows the monitoring plans, scheduling plans, etc. Appendix II shows the original hand-drawn Data Flow Diagram during the first design meeting. Appendix II shows the original Class Diagram during the second design meeting.

Table of Contents

1. Introduction

1.1 Project Overview

Upon the STAKEHOLDERS' request, this project is designed to generate word clouds for lyrics of specified artists. This system is call Lyrics This document specifically walks through every detail about the design of this project. All the design features are based on the Software Requirements Specification document provided by Group #17 of CSCI 310 (Software Engineering). This document includes two major components: Design Diagram chapter, and Design Evaluation Chapter.

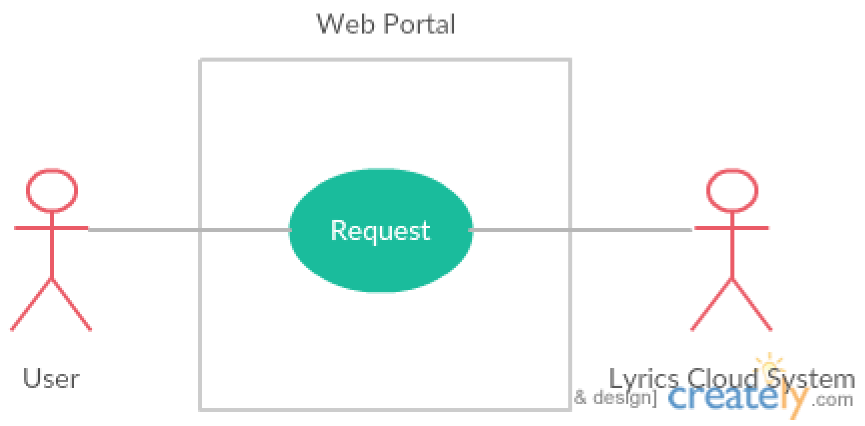
1.2 Definitions and Acronyms

SRS: The specific Software Requirement Specification produced by Team #17 on Blackboard.
PMP: Project Management Plan, specifically the documentation Team #6 handled in on Feb 8th.
Lyrics Cloud System: the system which will generate a clickable word cloud as mentioned in Team#17's SRS
Client: Users who are using Lyrics Cloud System.
Interaction: the combination of a sequence of input to the system and the corresponding result displayed by the system to clients.
Subteam: a 2-person organization under this group.
Subteam 1: the first of the 3 subteams from this group, composed by Ernest and Leo.
Subteam 2: the second of the 3 subteams from this group, composed by Lifan and Mengqian.
Subteam 3: the third of the 3 subteams from this group, composed by Juntao and Yawen.

2. Design Diagrams

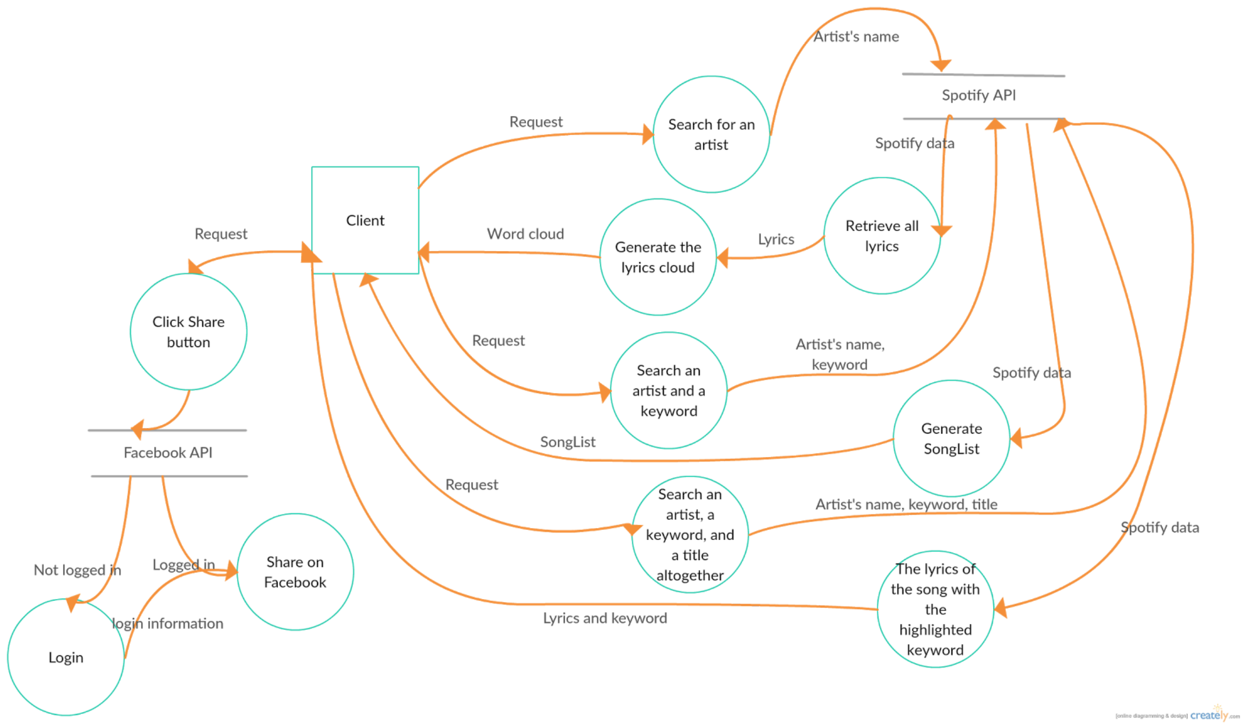
This chapter includes necessary diagrams to illustrate all the design features of this system. Each diagram is followed by a few paragraphs to explain how this diagram complies with the Software Requirements Specification document provided by Group #17.

2.1 Use Case Diagram



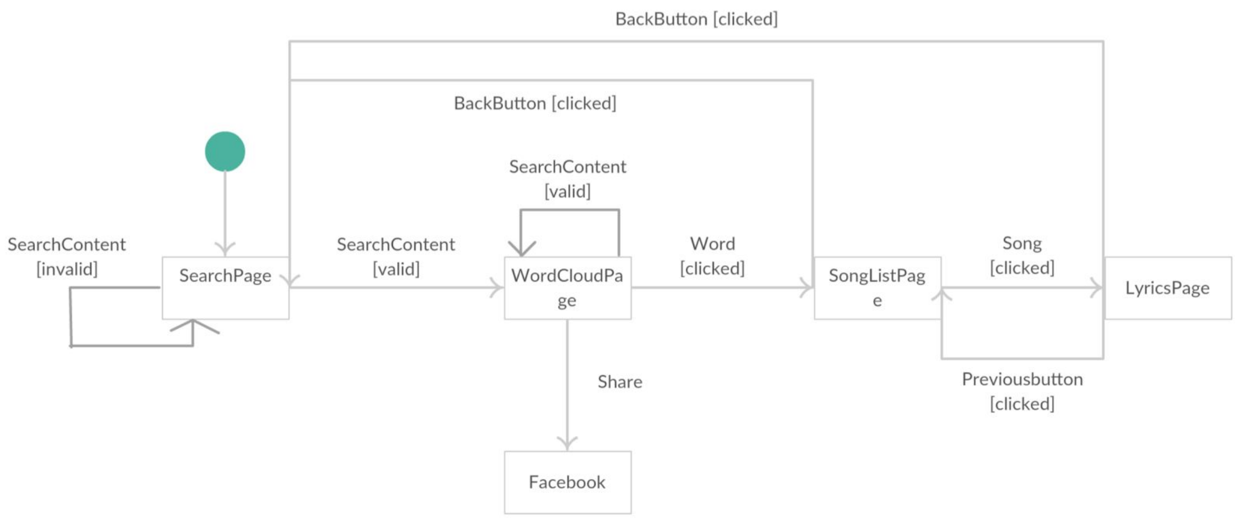
Lyrics Cloud: Request	
Actors	Clients, Lyrics Cloud System
Descriptions	A client may send request to the system. The requests may either be 1. Searching one or several artist names for word cloud, 2. Searching(by clicking) a specific word which is shown in word cloud or 3. Searching(by clicking) a specific song which is shown in song list
Data	Formatted request, Corresponding webpage url
Stimulus	Search Button clicked by clients
Response	Result corresponding to requests: 1. Navigate to Word Cloud Page 2. Navigate to Song List Page 3. Navigate to Lyrics Page
Comments	N/A

2.2 Data Flow Diagram



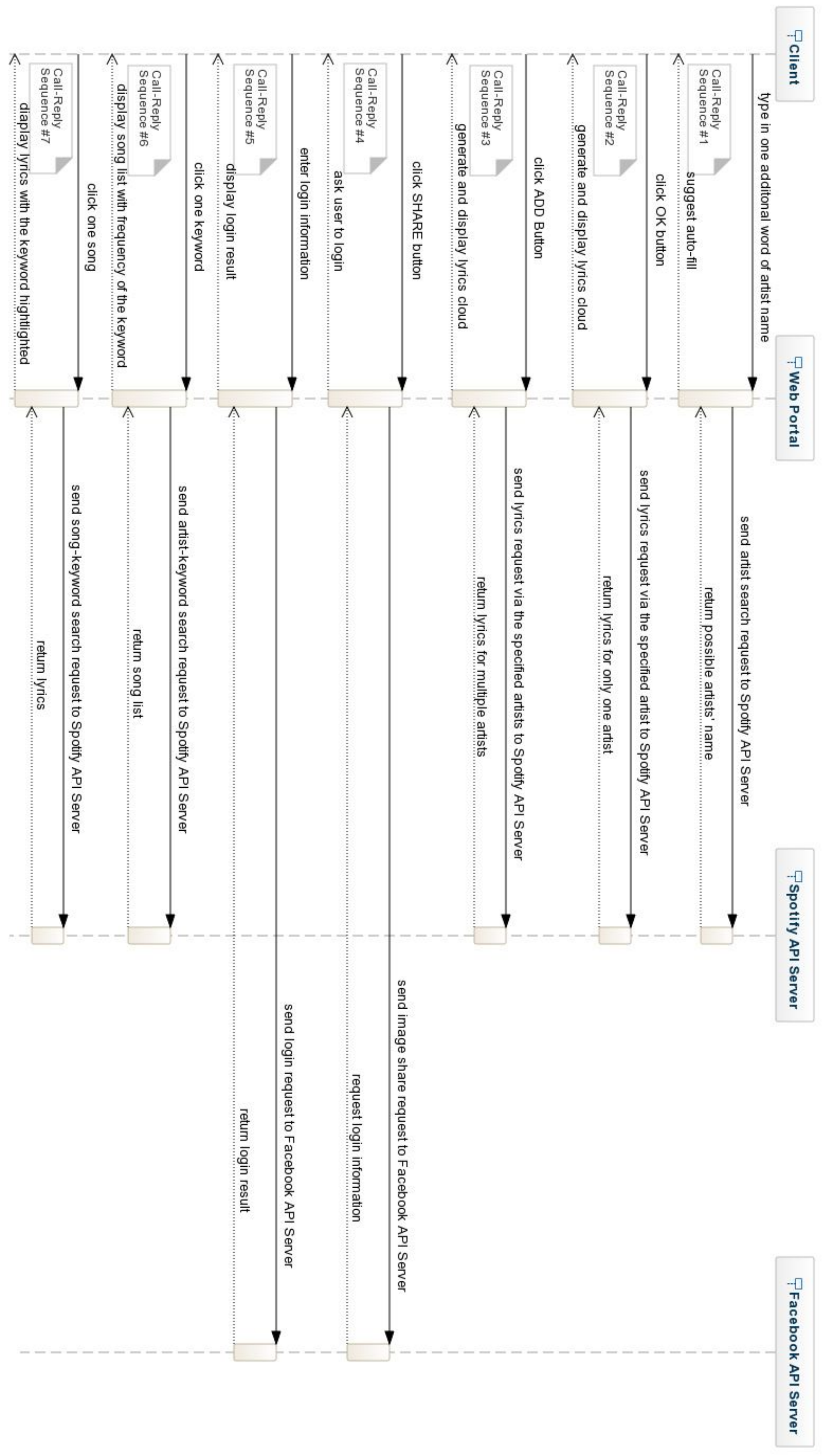
This data flow diagram demonstrates that how data flow through the Lyrics Cloud System and how the system processes data to implement the required features. Spotify API plays a very critical role at processing data because the majority information comes from the API server. There is also a variety of types of searches, so that the system can use more detailed search inputs to narrow down the search results and eventually get more accurate results. For example, when the system requests Spotify API with a song's title, a keyword and an artist's name, the goal is to get the correct lyrics of that song in case where a song has many cover versions.

2.3 State Machine Diagram



This state machine diagram demonstrates that different states in the Lyrics Cloud System and transitions among stages. A web page would be properly considered as a stage because a web page can respond differently with respect to all different clients' operations. Transitions from one web page to another are also triggered by clients' operations along with the result sent by API.

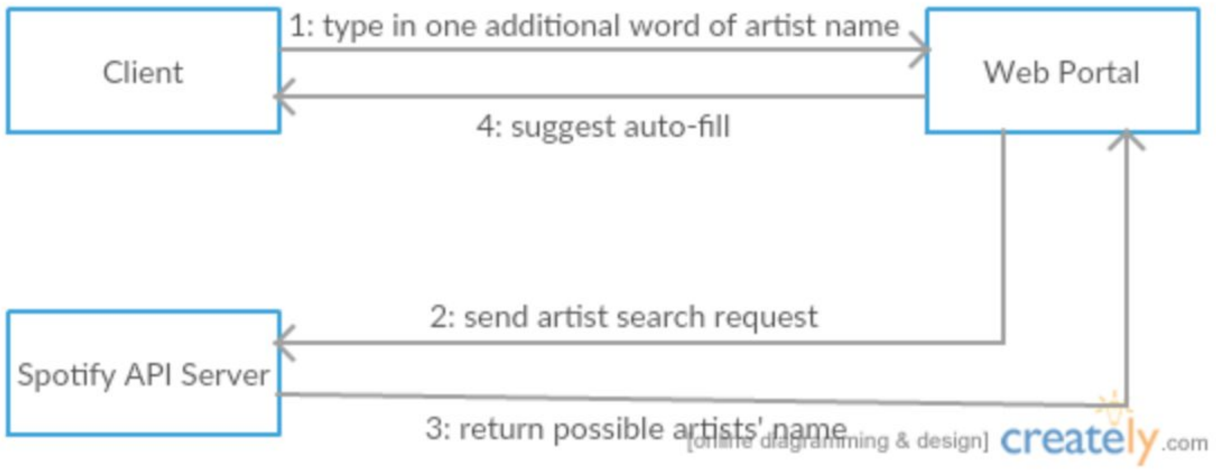
2.4 Sequence Diagram


Client
Web Portal
Spotify API Server
Facebook API Server
type in one additonal word of artist name
send artist search request to Spotify API Server
return possible artists' name
Call-Reply Sequence #1
suggest auto-fill
click OK button
send lyrics request via the specified artist to Spotify API Server
return lyrics for only one artist
Call-Reply Sequence #2
generate and display lyrics cloud
click ADD Button
send lyrics request via the specified artists to Spotify API Server
return lyrics for multiple artists
Call-Reply Sequence #3
generate and display lyrics cloud
click SHARE button
send image share request to Facebook API Server
request login information
Call-Reply Sequence #4
ask user to login
enter login information
send login request to Facebook API Server
return login result
Call-Reply Sequence #5
display login result
click one keyword
send artist-keyword search request to Spotify API Server
return song list
Call-Reply Sequence #6
display song list with frequency of the keyword
click one song
send song-keyword search request to Spotify API Server
return lyrics
Call-Reply Sequence #7
diaplay lyrics with the keyword hightlighted

Lyrics Cloud: Sequence Diagram	
Modules	client, Web Portal of Lyrics Cloud, Spotify API Server, Facebook API Server
Calls	Calls that stimulate formatted request sent to Spotify API Server: 1. Typing in one additional word stimulates an artist search request sent to Spotify API server. 2. Clicking OK button stimulates a lyrics search request via the specified artist's name sent to Spotify API server. 3. Clicking ADD button stimulates a lyrics search request, via the names of the most recently entered artist plus those pre-existing ones, sent to Spotify API server. 4. Clicking SHARE button stimulates an image share request sent to Facebook API Server. 5. Entering login information stimulates a login request sent to Facebook API Server. 6. Clicking one keyword on the generated word cloud stimulates an artist-keyword combined search request sent to Spotify API server. 7. Clicking one song in the list stimulates a song-keyword combined search request sent to Spotify API Server.
Data	Formatted request sent to Spotify API Server or Facebook API Server
Stimulus	Buttons clicked by the clients
Replies	Replies corresponding to calls (see above): 1. Spotify API Server returns possible artists' names, based on which the auto-fill will suggest. 2. Spotify API Server returns all the lyrics of this specified artist, so that the web portal can generate and display the Lyrics Cloud for this specified artist. 3. Spotify API Server returns all the lyrics for all specified artists combined, so that the web portal can generate and display the Lyrics Cloud for these specified artists. 4. Facebook API Server returns a request to ask for login information, which will be displayed on the web portal. 5. Facebook API Server returns the result of that login entry, which will be displayed on the web portal. 6. Spotify API Server returns a list of songs that contain the clicked keyword and sung by the specified artist(s), so that the web portal will display the returned list and the frequency of the keyword next to each song. 7. Spotify API Server returns the lyrics of the specified song, so that the web portal will display the lyrics with the keyword highlighted.

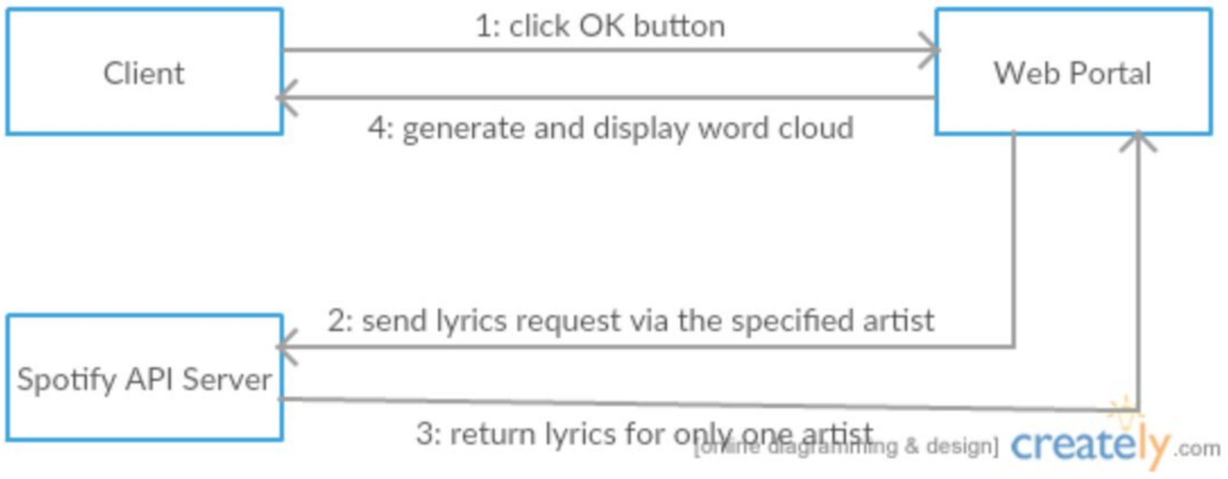
2.5 Communication Diagram

2.5.1 Call-Reply Sequence #1


Client
1: type in one additional word of artist name
Web Portal
4: suggest auto-fill
2: send artist search request
Spotify API Server
3: return possible artists' name
[online diagramming & design] creately.com

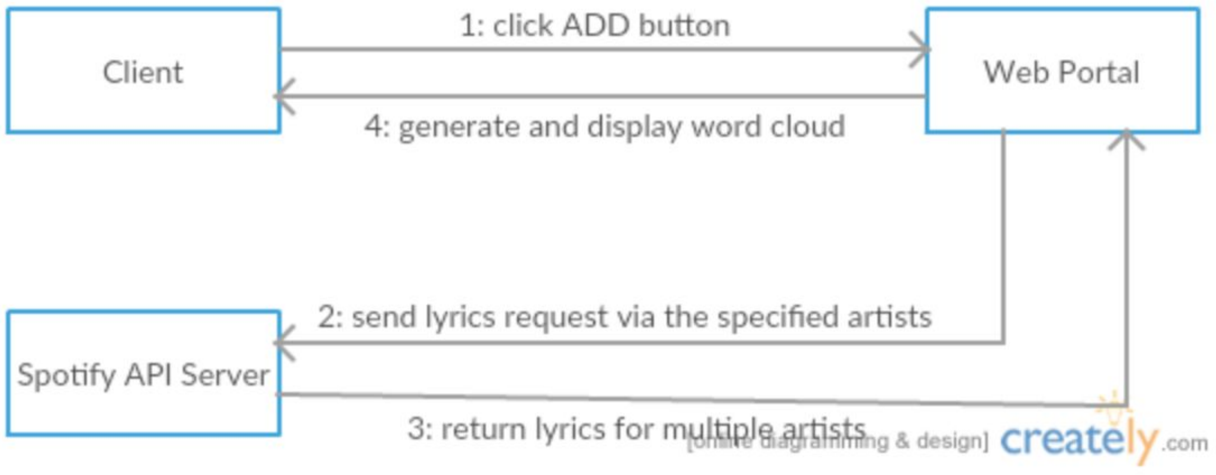
As the client types in one additional word of artist name, the web portal shall send artist search request to Spotify API server based on client's input. After the Spotify API server return possible artists' names, the web portal shall suggest auto-fill below the search box.

2.5.2 Call-Reply Sequence #2


Client
1: click OK button
Web Portal
4: generate and display word cloud
2: send lyrics request via the specified artist
Spotify API Server
3: return lyrics for only one artist
[online diagramming & design] creately.com

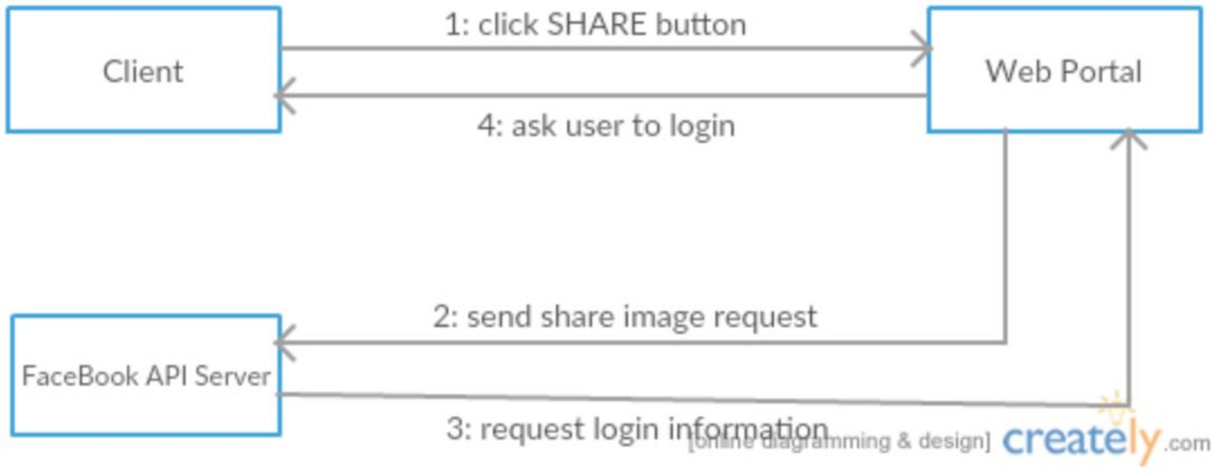
When the client clicks the OK button, the web portal shall request Spotify API server to generate the specified artist's lyrics of all songs. After the Spotify API server return lyrics for the artist to the web portal, the web portal shall generate and display word cloud to the client.

2.5.3 Call-Reply Sequence #3


Client
1: click ADD button
Web Portal
4: generate and display word cloud
2: send lyrics request via the specified artists
Spotify API Server
3: return lyrics for multiple artists
[online diagramming & design] creately.com

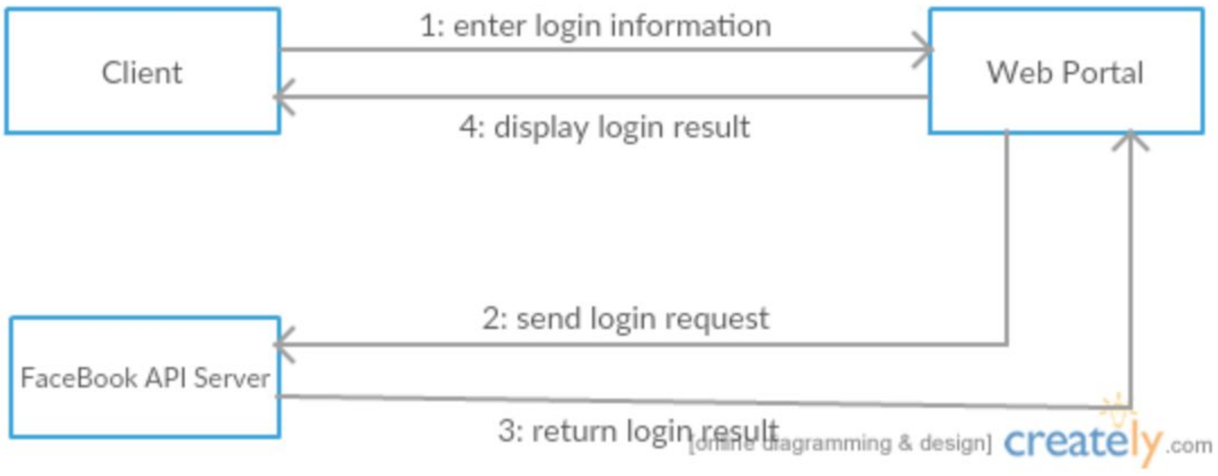
When the client clicks the ADD button, the web portal shall request Spotify API server to generate the specified artists' lyrics of all songs. After the Spotify API server return lyrics for the artists to the web portal, the web portal shall generate and display word cloud to the client.

2.5.4 Call-Reply Sequence #4


Client
1: click SHARE button
Web Portal
4: ask user to login
2: send share image request
FaceBook API Server
3: request login information
[online diagramming & design] creately.com

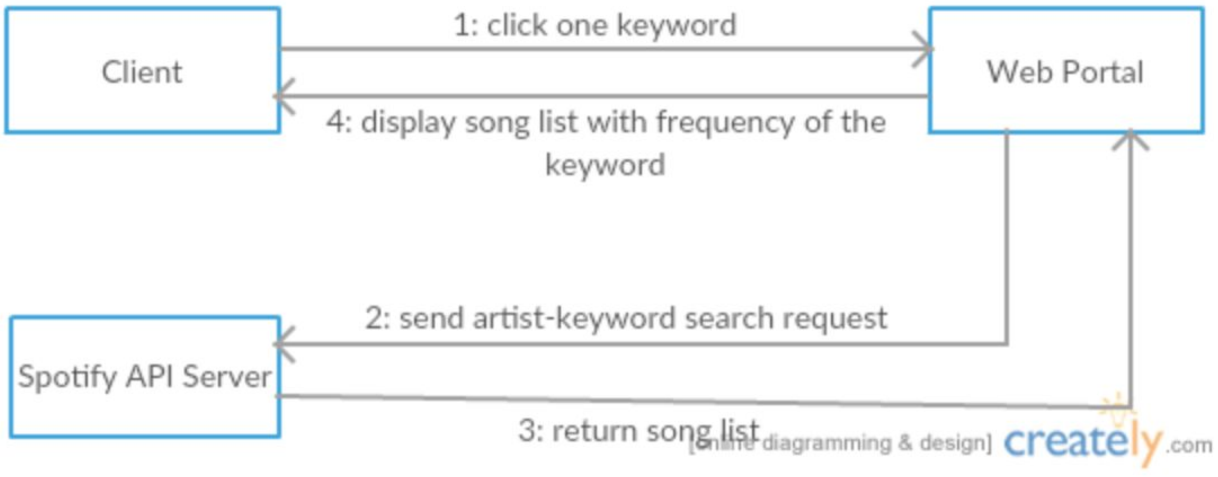
When the client clicks SHARE button, the web portal shall send share image request to Facebook API server. When Facebook API server receives the request, it shall request login information to web portal. Then the web portal shall ask client to login.

2.5.5 Call-Reply Sequence #5



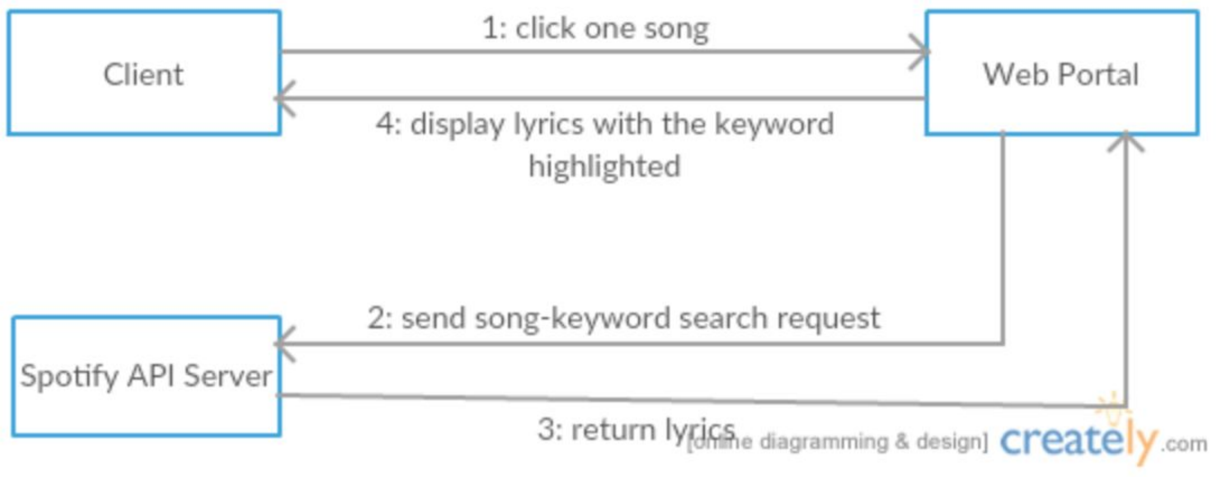
After the client enters login information, the web portal shall send login request to FaceBook API server. After Facebook API server receives the request, it shall request return the login result to web portal. Then the web portal shall display login result to the client.

2.5.6 Call-Reply Sequence #6


Client
Web Portal
Spotify API Server
1: click one keyword
4: display song list with frequency of the keyword
2: send artist-keyword search request
3: return song list
[online diagramming & design] creately.com

When the client clicks one keyword, the web portal shall send search request to Spotify API server based on the specified artist and the keyword the client has clicked. After Spotify API server receives the request, it shall return a song list in which every song contains the keyword in its lyrics to the web portal. Then, the web portal shall display the song list with frequency of the keyword to the client.

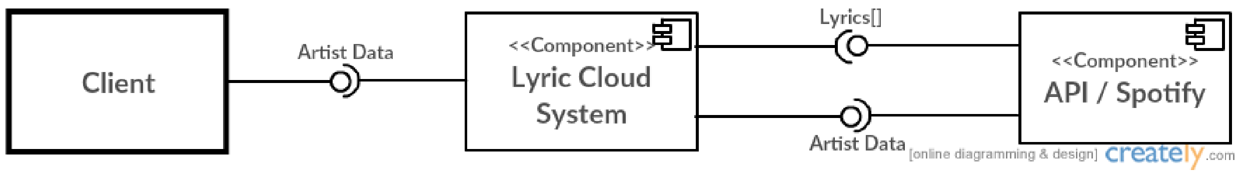
2.5.7 Call-Reply Sequence #7


Client
1: click one song
Web Portal
4: display lyrics with the keyword highlighted
2: send song-keyword search request
Spotify API Server
3: return lyrics
[online diagramming & design] creately.com

After the client clicks one song, the web portal shall send song-keyword search request to Spotify API server. After Spotify API server receives the request, it shall return lyrics to the web portal. Then, the web portal shall display lyrics with the keyword highlighted to the client.

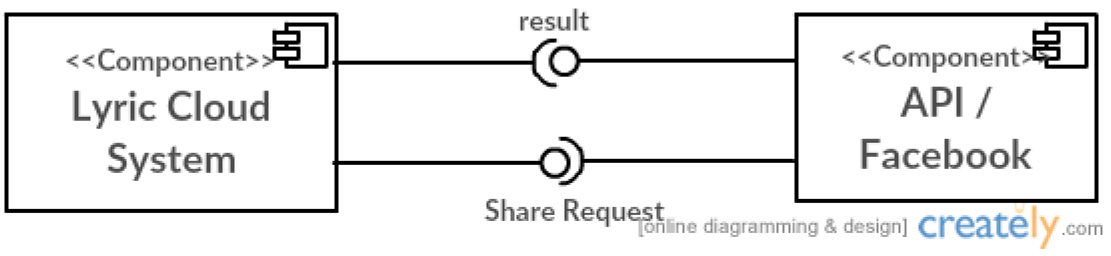
2.6 Component Diagram

2.6.1 ***Component Diagram for Spotify API***



This Component Diagram shows the dependencies that Lyric Cloud System has on other software components, the Spotify API, in the system. clients components, the client, has to provide artist data for the software so that it can process the artist. In other words, the Lyric Cloud System depends on the client's' inputs, artists names. For the software, the Lyric Cloud System has to provide the names/ name of the artists/ artist for the Spotify API components to search for results. It means that Spotify API is dependent on the Lyric Cloud System's artist data. The Lyric Cloud System is also dependent on Spotify API to provide the outputs of the search to handle and generate the word cloud.

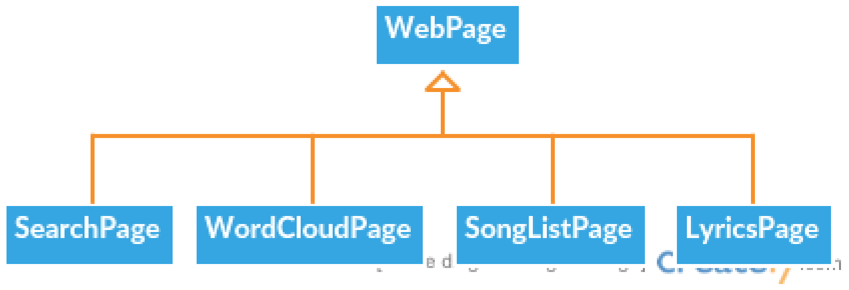
2.6.2 Component Diagram for Facebook API



This Component Diagram shows the dependencies that the Lyric Cloud System has on other software components, the Facebook API. The Facebook API is dependent on the share request made by the Lyric Cloud Software. The Lyric Cloud System is also dependent on the share results from the Facebook API.

2.7 Class Diagram

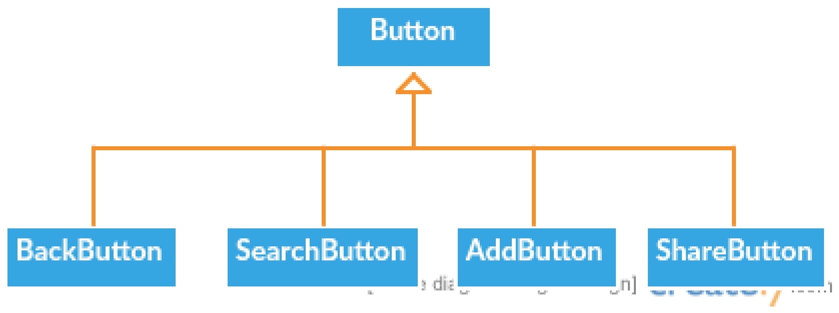
2.7.1 WebPage Class



WebPage is the basic webpage that all the detail pages are to build from.

- SearchPage, refers to SRS 3.1.1.1, is the webpage which provides search bar for client to search.
- WordCloudPage, refers to SRS 3.1.1.2, is the webpage which shows generated word cloud and services other functionalities.
- SongListPage, refers to SRS 3.1.1.3 which shows the song list containing a specified word.
- LyricsPage, refers to SRS 3.1.1.4 which shows the entire lyrics given a chosen song.

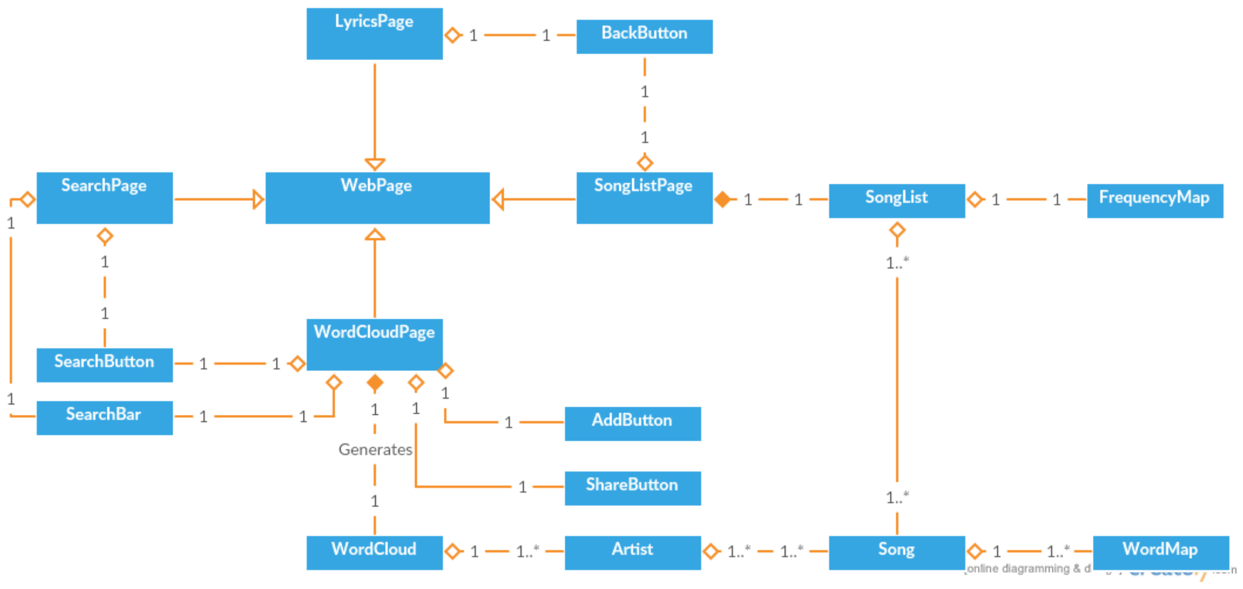
2.7.2 Button Class



Button is the main interaction between client and the lyrics system.

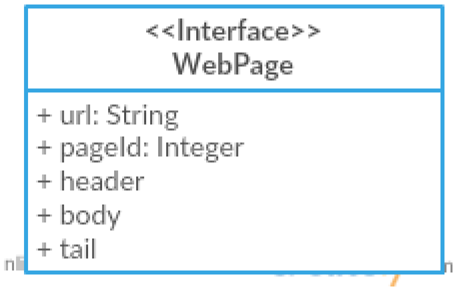
- SearchButton, refers to SRS 3.2.2, is the button which stimulus searching functionality.
- AddButton, refers to SRS 3.2.5, is the button which allows the client to search two or more artist at the same time.
- ShareButton, refers to SRS 3.2.6, is the button which allows the client to share the image of the word cloud on their Facebook.

2.7.3 Class Diagram

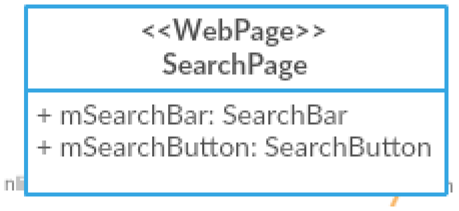


This diagram demonstrates the relationship between each class. Each entity represent a class which only the class name is shown there. The detail of each class will be described below.

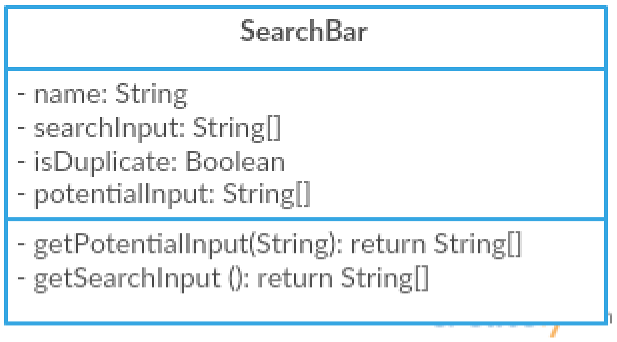
2.7.4 Class Detail



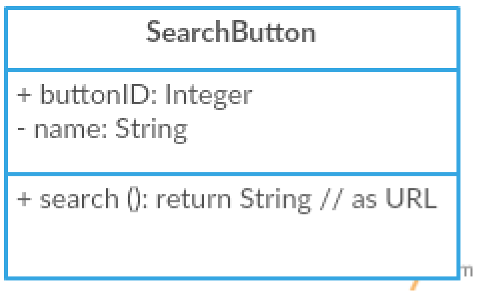
WebPage is the interface of the other four webpage class. It has a String which records the website url. It has pageId to identify. It consists of three basic parts, header, body and tail.



SearchPage is the subclass class of Webpage. Besides the variables that WebPage contains, it has a SearchBar in which client can type text and a SearchButton which are to trigger the search functionality.


SearchBar
- name: String
- searchInput: String[]
- isDuplicate: Boolean
- potentialInput: String[]
- getPotentialInput(String): return String[]
- getSearchInput (): return String[]

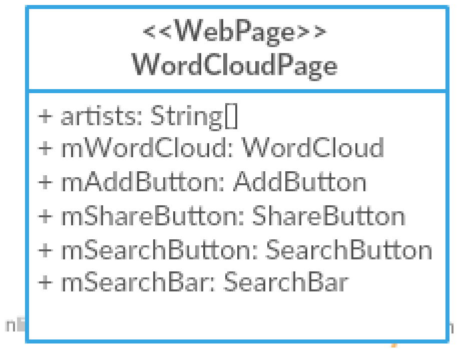
SearchBar is a textbox where the client can search for song artists, with a suggestion drop-down which sometimes visible beneath the textbox. Refers to SRS 3.2.1. It has a String for its name, a String array for searchInput(in case of searching multiple artists). It has a Boolean value to check if the requested name is duplicate. And it owns a String array for the suggestion drop-down. getPotentialInput(String) will return the String array, potentialInput. getSearchInput() will return the searchInput String array.


SearchButton

+ buttonID: Integer
- name: String

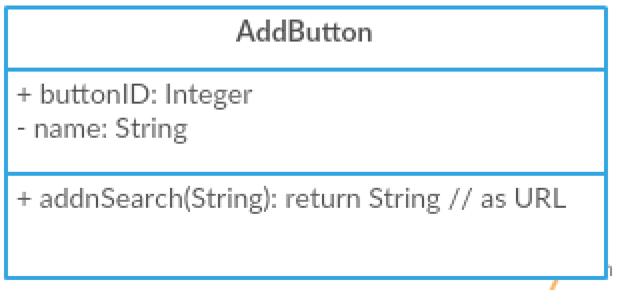
+ search (): return String // as URL

SearchButton is the subclass of Button. It stimulates the system to search and generate word cloud by the given Artist name(s). It has a buttonID integer to be called and identifies and a String for name. search() serves as the stimulus which will generate a WordCloud and navigate to the WordCloudPage.


<<WebPage>>
WordCloudPage

+ artists: String[]
+ mWordCloud: WordCloud
+ mAddButton: AddButton
+ mShareButton: ShareButton
+ mSearchButton: SearchButton
+ mSearchBar: SearchBar

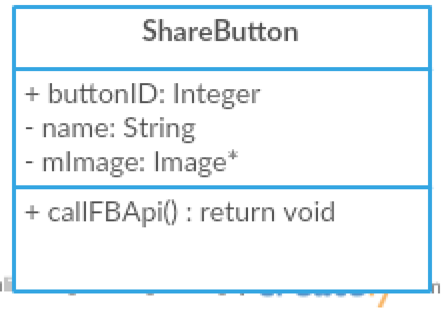
WordCloudPage is the subclass of WebPage. It has a String array of artists name which is passed from SearchPage and will use for Word Cloud Title. Refers to SRS 3.2.3. It has a WordCloud which presents the demanded word cloud. Refers to SRS 3.2.4. It as well has three different buttons and a SearchBar which has the same functionality as the one in SearchPage.


AddButton

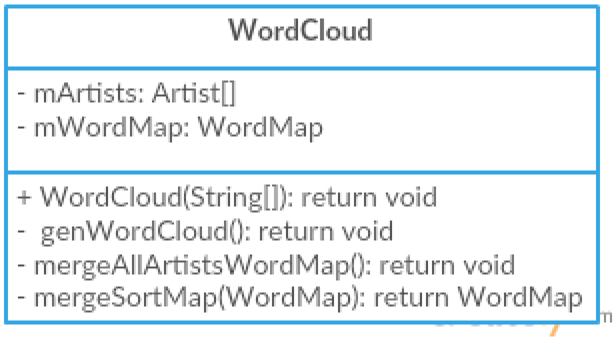
+ buttonID: Integer
- name: String

+ addnSearch(String): return String // as URL

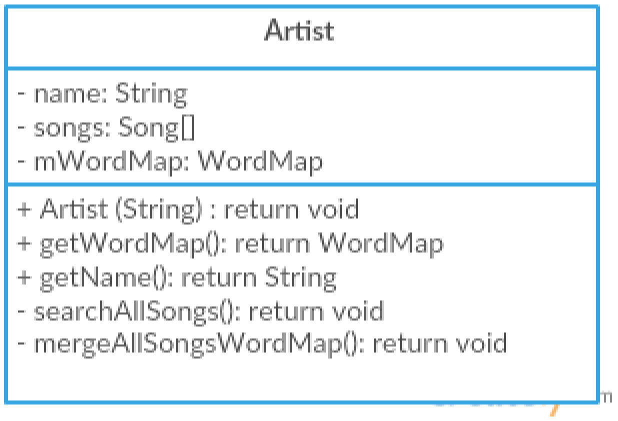
AddButton is the subclass of Button. It simulates the system to add another artist to the previous searchInput, search and generate a new WordCloud. addnSearch(String) function serves for this purpose. It has a buttonID integer to be called and identifies and a String for name.



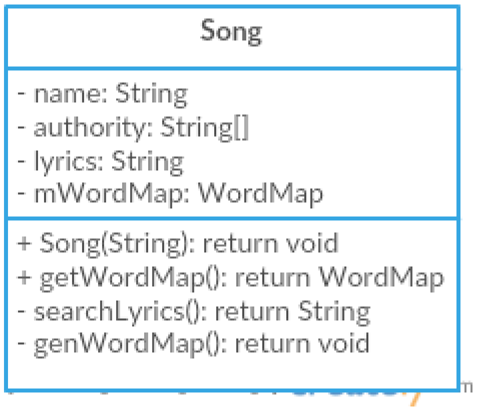
ShareButton is the subclass of Button. It simulates the system to connect Facebook API and share the image of WordCloud on Facebook. callFBApi() function serves for this purpose. It has a buttonID integer to be called and identifies and a String for name.


WordCloud
- mArtists: Artist[]
- mWordMap: WordMap
+ WordCloud(String[]): return void
- genWordCloud(): return void
- mergeAllArtistsWordMap(): return void
- mergeSortMap(WordMap): return WordMap

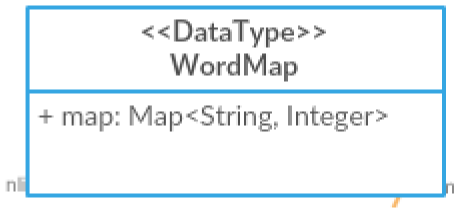
WordCloud consists of an Artist array which generates from the parameter passed in and a WordMap which contains all the words and their importance. WordCloud(String[]) takes in the artist name String array from SearchPage. It constructs the Artists by the String array, call mergeAllArtistsWordMap() which merges the WordMap in each given artist, and then call mergeSortMap(WordMap) which computes the importance and sorts the map, and finally call genWordCloud() to display the WordCloud on WordCloudPage.


Artist
- name: String
- songs: Song[]
- mWordMap: WordMap
+ Artist (String) : return void
+ getWordMap(): return WordMap
+ getName(): return String
- searchAllSongs(): return void
- mergeAllSongsWordMap(): return void

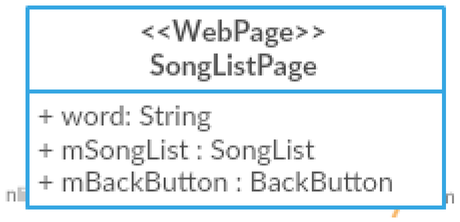
Artist is the aggregation of the WordCloud. It contains a name String, a Song array,songs , including all his/her songs and a WordMap, mWordMap, holding the data of WordMap of songs. Artist(String) takes the name as parameter from WordCloud constructor and call searchAllSongs() and mergeAllSongsWordMap() to search and merge all WordMap to mWordMap. getWordMap() will be called by WordCloud::mergeAllArtistWordMap(), return mWordMap.


Song
- name: String
- authority: String[]
- lyrics: String
- mWordMap: WordMap
+ Song(String): return void
+ getWordMap(): return WordMap
- searchLyrics(): return String
- genWordMap(): return void

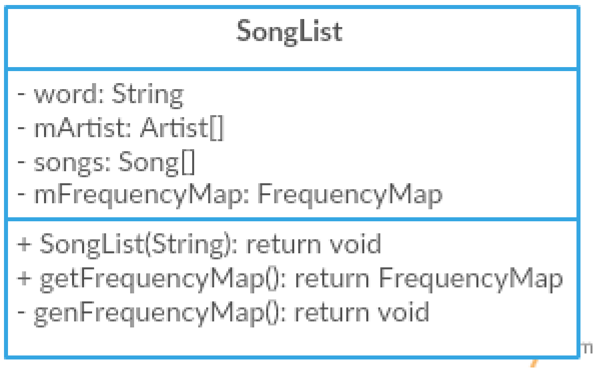
Song is the aggregation of Artists. It contains a name String and a String array, authority, indicating the author of this song. It also has a String lyrics and a WordMap. Song(String) is the constructor which will call searchLyrics() to search the entire lyrics and genWordMap() to generate the WordMap. getWordMap() is supposed to be called by Artist::mergeAllSongsWordMap(), returning the mWordMap.



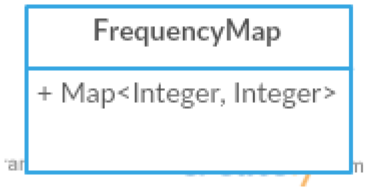
A WordMap is a hashmap which links one word and its occurrence.



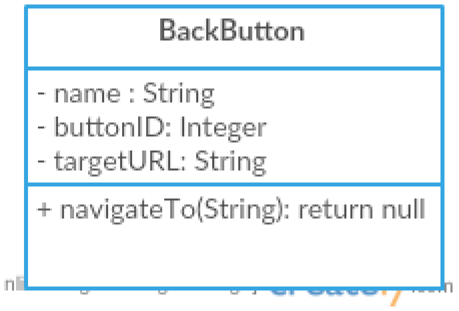
SongListPage is the subclass of Webpage. It has a String word for Song List Title, a SongList mSongList and a BackButton mBackButton.


SongList
- word: String
- mArtist: Artist[]
- songs: Song[]
- mFrequencyMap: FrequencyMap
+ SongList(String): return void
+ getFrequencyMap(): return FrequencyMap
- genFrequencyMap(): return void

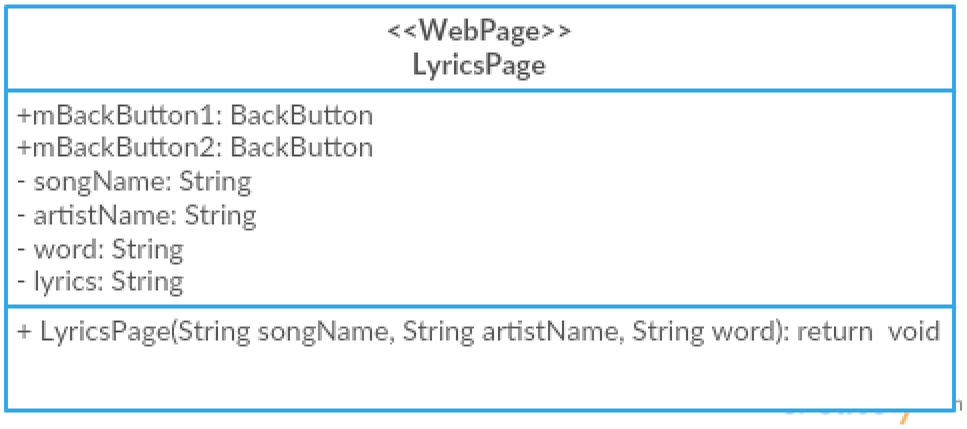
SongList is the composition of SongListPage. It has a String word as the targeted word, a Artist array mArtist, Song array containing the searching result and mFrequencyMap as the recording the occurrence of each word in each songs.



FrequencyMap is a hashmap that link the index of songs in SongList and the frequency of that word shows up.



BackButton is the subclass of Button. It simulates the system to navigate back to the targetURL. Since the BackButton here only suppose to navigate to WordCloudPage, the parameter of navigateTo(String) will simply be targetURL.


<<WebPage>>
LyricsPage
+mBackButton1: BackButton
+mBackButton2: BackButton
- songName: String
- artistName: String
- word: String
- lyrics: String
+ LyricsPage(String songName, String artistName, String word): return void

LyricsPage is the subclass of WebPage. It has four String songName, artistName, word and lyrics which are used for display the entire song lyrics with the targeted word highlighted. It also has two BackButtion. mBackButtion1 will navigate back to WordCloudPage and mBackButton2 will navigate back to SongListPage.

3. Design Evaluation

This chapter evaluates the design at an integrated level for this system. The evaluation emphasizes on the design qualities, such as abstraction, modularity, coupling, cohesion, information hiding, simplicity, and hierarchy.

3.1 High-Level Design

We build the project design by stepwise refinement. Top-down design helps us generate the very beginning thought about how the detail of this system should be.

Firstly we create a Use Case Diagram which generalizes the interactions between clients and the Lyrics Cloud System to one action: requesting the website. Requesting action includes several specific actions which have been illustrated in the tabular chart of Use Case Diagram. At the same time, we see the Lyrics Cloud System as a whole application which will be detailed in the following design. By doing this abstraction, we can easily demonstrate the following diagram without considering much about the detail implementation.

For modularity, we define the Lyrics Cloud System, Spotify API and Facebook API as three different modules in high level design. Lyrics Cloud System and Spotify API module are sequential cohesion and data coupling, while Lyrics Cloud System and Facebook API module are logical cohesion and external coupling. This will allow us separate the coding task evenly and independently from one another.

Requesting action as an entity includes all kinds of interactions that will possibly take place. We then use Data Flow Diagram to demonstrate the way data is processed through the Lyrics Cloud System. This gives us a rough picture of how the system will be designed.

From the data flow diagram, we produce three more detailed diagram: State Machine Diagram to simulate clients interactions with Lyrics Cloud System on different stage, Sequence Diagram and Communication Diagram to better present how the Lyrics process requesting from the clients.

The sequence diagram reflects high level of abstraction and modularity, and sufficient amount of information hiding. It distributes the overall Requesting action into seven specific Calls . Each abstract action has corresponding replies listed in the chart. The design of Lyrics Cloud heavily employs Spotify API and Facebook API. Therefore, every formatted request sent to Spotify API Server or Facebook API Server returns a formatted result. The complexity of processing each request is simplified to one single request-return action, which provides a high level of abstraction. The design of Lyrics Cloud also features a high level of modularity, due to its sequential cohesion, and loose coupling. According to the sequence diagram, every single action inside each request-return sequence, including client's physical action, request sent from the web portal to the API Server, result returned by the API Server, and display of the return results, is processed in sequence: one action is be immediately followed by the next. Hence the sequential cohesion. Due to the heavy use of formatted request-return action via API Servers, this system is independent of the mechanism of API Servers. Thus, it demonstrates loose coupling. Combining strong sequential cohesion and loose coupling gives this system a high level of modularity. Since this system will be using formatted request-return actions, returned results from API Server are stored in carefully constructed data structures, whose information can only be accessed by calling getter methods. Therefore, this features good information hiding.

The sequence diagram mainly focus the order of interactions, while the communication diagram illustrates the communication between modules through each interactions. It describes the same staff from different viewpoint.

There are lots of information hiding in this level. For Lyrics Cloud System module, we know it will take in requests and output the corresponding results. However, we have no idea how it implements on ground. The detail implementations of the other two API modules are also hiding. The information hiding provides more feasibility, allowing the future change in each module without affecting other module.

3.2 Detailed Design

In low level design, we distribute the Lyrics Cloud System module into smaller 15 modules:

- WebPage abstract class including SearchPage, WordCloudPage, SongListPage,LyricsPage;
- Button abstract class including SearchButton, AddButton, ShareButto andBackButton;
- Data structure like WordMap and FrequencyMap;
- Other classes like WordCloud, Artist, Song, SongList and SearchBar;

Abstraction to this level gives us the advantage of seeing how to implement each modules in detail and what the relationship will be.

The class diagram clearly shows how those 15 modules relate to each other. The following chart is used to demonstrate the modularity of each pair of modules that are related.

Module A	Module B	Cohesion	Coupling	Explanation
WordCloud	Artist	Procedure Cohesion	Data Coupling	WordCloud will generate Artist which makes them process in order. WordCloud only get WordMap data in Artist.
Artist	Song	Procedure Cohesion	Data Coupling	Artist will generate array of Song which follow a certain sequence of execution. Artist only get WordMap data from Song
Song	SpotifyAPI	Function Cohesion	Data Coupling	Song will get lyrics from SpotifyAPI by using function and use that to generate WordMap. Song only get lyrics from SpotifyAPI.

SongList	Song	Procedure Cohesion	Data Coupling	SongList will generate array of Song which follow a certain sequence of execution. SongList only get WordMap data from Song
SearchPage	SearchButton SearchBar	Temporal Cohesion	Control Coupling	SearchPage will call to construct a SearchButton and a SearchBar when itself being constructed. SearchPage can control execution of both.
WordCloud Page	SearchBar SearchButton AddButton ShareButton WordCloud	Temporal Cohesion	Control Coupling	WordCloudPage will call to construct SearchBar, SearchButton, AddButton, ShareButton, WordCloud when itself being constructed. WordCloudPage can control execution of all of them.
SongList Page	SongList BackButton	Temporal Cohesion	Control Coupling	SongListPage will call to construct a Songlist and a BackButton when itself being constructed. SongListPage can control execution of both
SearchButton	AddButton	Logical Cohesion	Data Coupling	AddButton serves the similar searching functionality as SearchButton. AddButton passes one more artist name to SearchButton to call its function.

*For those pairs which are not listed in the chart, it means that they have no cohesion or coupling related to each other

In all the modules, the variables and functions that are assigned with plus symbol, represent as the information need to be acknowledged, and those with minus symbol are information can be kept in the dark. These private variables are hidden because they should no longer be modified in other modules. Similarly, these private functions are hidden for the reason that they serve as unique functionalities that would only be used once in that particular module.

What's more, the usage of Spotify API and Facebook API are hiding under the module's functions. Since api and modules are only data coupling, it is quite reasonable to hide the specific calls to API in the functions.

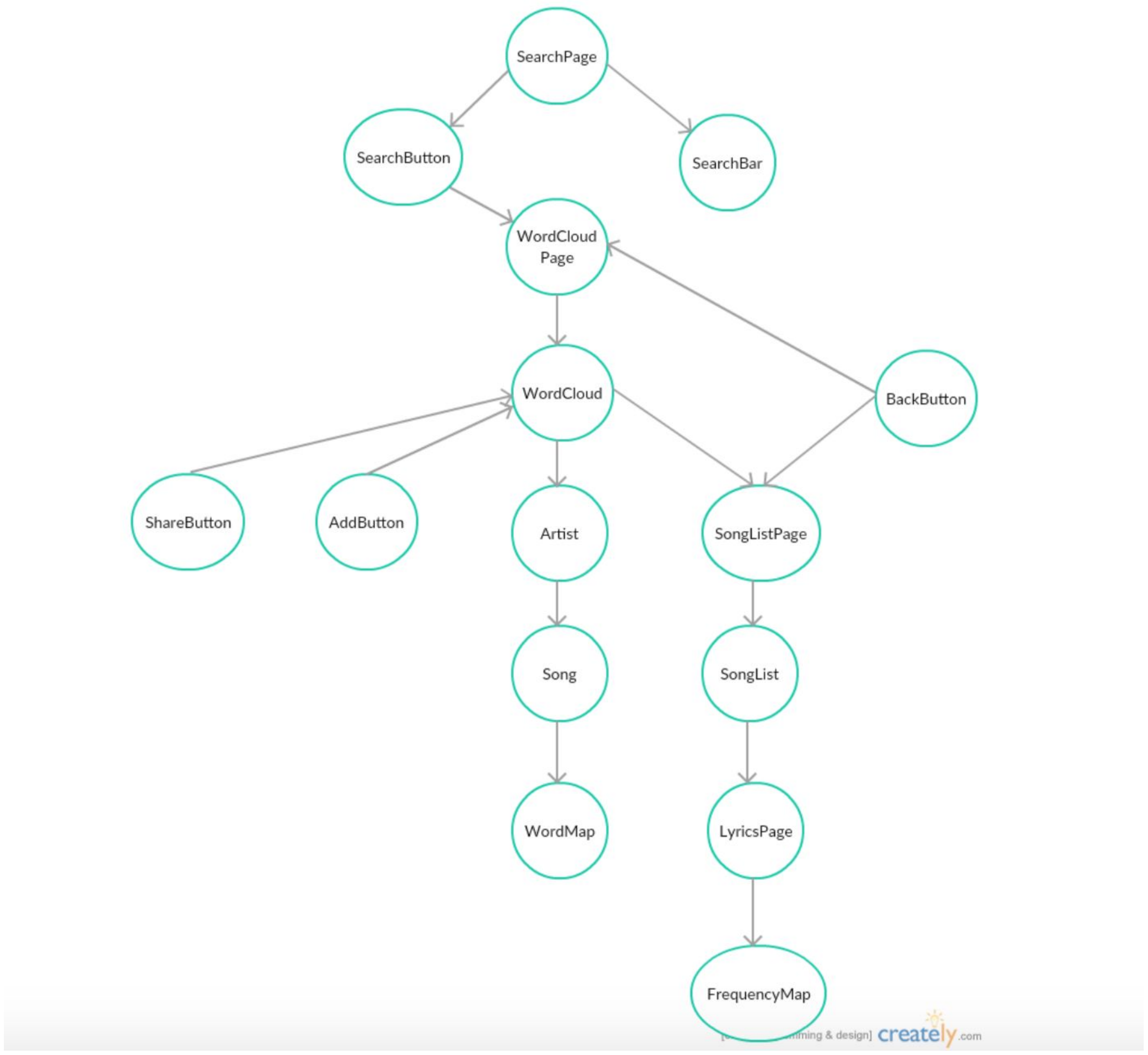
Although complexity is roughly discussed in the High Level Design part, the detailed complexity will be discussed in this part. This portion will introcude both intra-complexity and inter-complexity.

First, for intra-complexity, we has estimated the runtime for functions that will be used in future implementation.

Generate Word Cloud: From the very bottom of our design, we will get the lyrics from SpotifyAPI O(1) and then use the linear search in Song::genWordMap() that scans through all text and find the exact keyword, and that gives the runtime of O(n). After traversing all lyrics, an Artist module will use its private function to collect all the WordMaps from his/her songs. This step, Artist::mergeAllSongsWordMap(), will take O(n) runtime to accomplish. In order to generate the lyrics cloud, the approach we uses is to compare all words with their frequencies, and display them in a decreasing order. A decent sorting algorithm will be Merge Sort, because that has the minimum runtime among all comparison-based algorithms, which is O(nlogn). Therefore, the runtime of WordCloud::genWordCloud() will take O(nlogn) in total.

Generate Song List: From the very bottom of our design, we will get the lyrics from SpotifyAPI O(1) and then use the linear search in Song::genWordMap() that scans through all text and find the exact keyword, and that gives the runtime of O(n). After traversing all lyrics, a SongList module will collect all the WordMaps from Songs. The SongList::getFrequencyMap() will merge them all together, taking O(n) time. And then MergeSort function will as well take O(nlogn) time.

Second, the Class Diagram shows the relationships among the 15 modules. The following graph is the call graph that demonstrate our system's call hierarchy based on the Class Diagram. The entity with an arrowhead end is called by the entity on the other side. It demonstrates the complexity of the 15 modules. The height of this graph is 8; the width is 2; the number of nodes is 15; the number of edges is 15.



Appendix I.
Description of Design Process

After handling PMP, we began our routined per-three-day meeting. During the first meeting, we reviewed the lecture notes and discovered that the previous staff allocation did not contain the high level design. The main part of the previous planning was focusing on low level design. Therefore, we studied the lecture notes 6 through 8 on that meeting, prepared questions and slightly revised our design plan. We drew a Use Case Diagram and a Data Flow Diagram during that meeting, and this became the base of our design. Each subteam was assigned with one diagram of high level design. Subteam 1 was assigned with Sequence Diagram and Communication Diagram. Subteam 2 was assigned with State Machine Diagram. Subteam 3 was assigned with Component Diagram. While working on the newly assigned diagram, each team was still required to accomplish the research of their own task.

On Feb 9th, each subteam member talked about his/her research on wechat. And before the meeting on Feb 11th, each subteam handled their rough low level design for its particular part and the high level design diagram to the Google Doc as the PMP Monitoring Plan Task 2, 4, 6 required.

During the meeting on Feb 11th, we discovered that our separated low level designs could not be used directly. We, then, firstly, decided to finish the high level design, not as the PMP Monitoring Plan Task 7 required. After the combination of each assigned diagram, we used the stepwise refinement and started to modify our low level design based on high level diagrams.

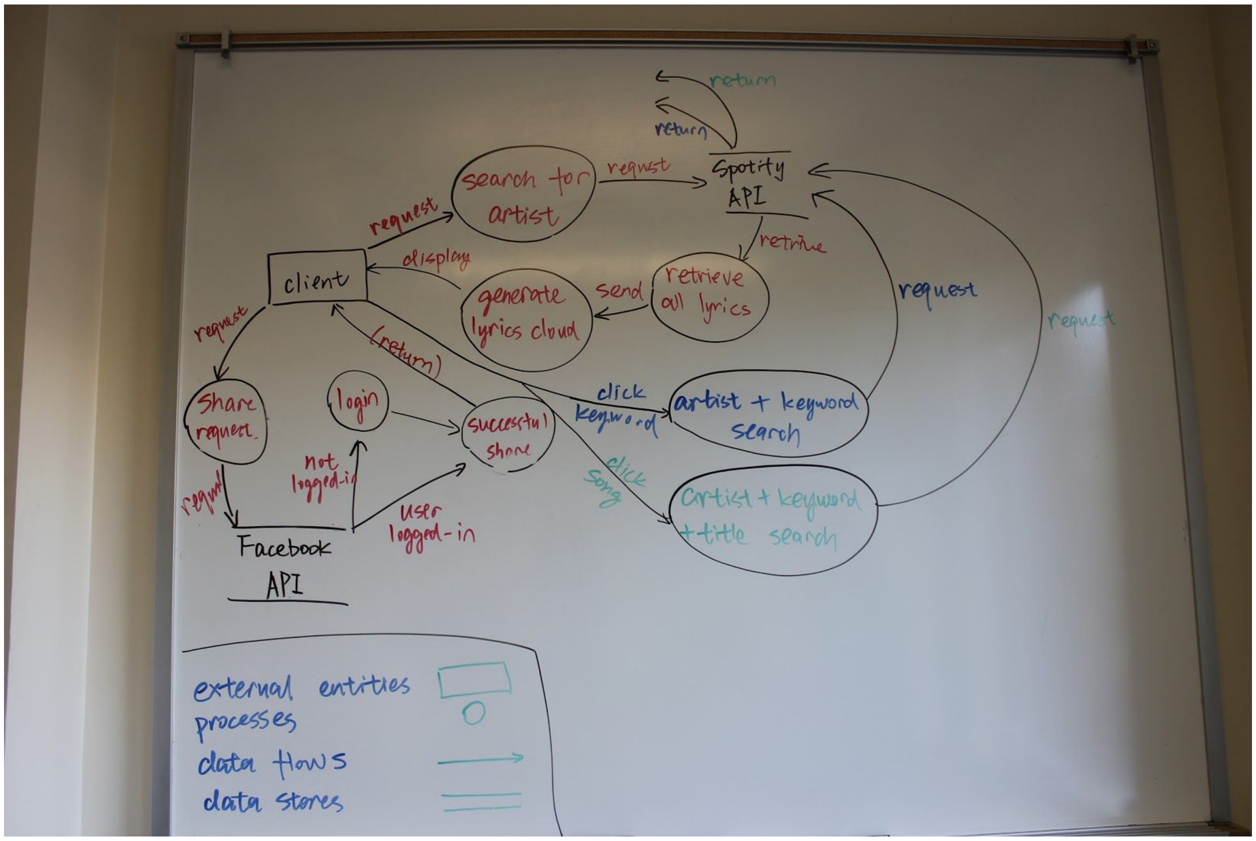
After that meeting, Subteam 1 responsed for the study of Spotify API and related module design. Subteam 2 responsed for the study of Word Cloud and detailed the WordCloud, Artist, Song module. Subteam 3 responsed for the study of UI and implemented the WebPage modules and Button modules.

Before the meeting on Feb 14th, each subteam provided their part of the Class Diagram on Google Doc as the PMP Monitoring Plan Task 8-10 required.

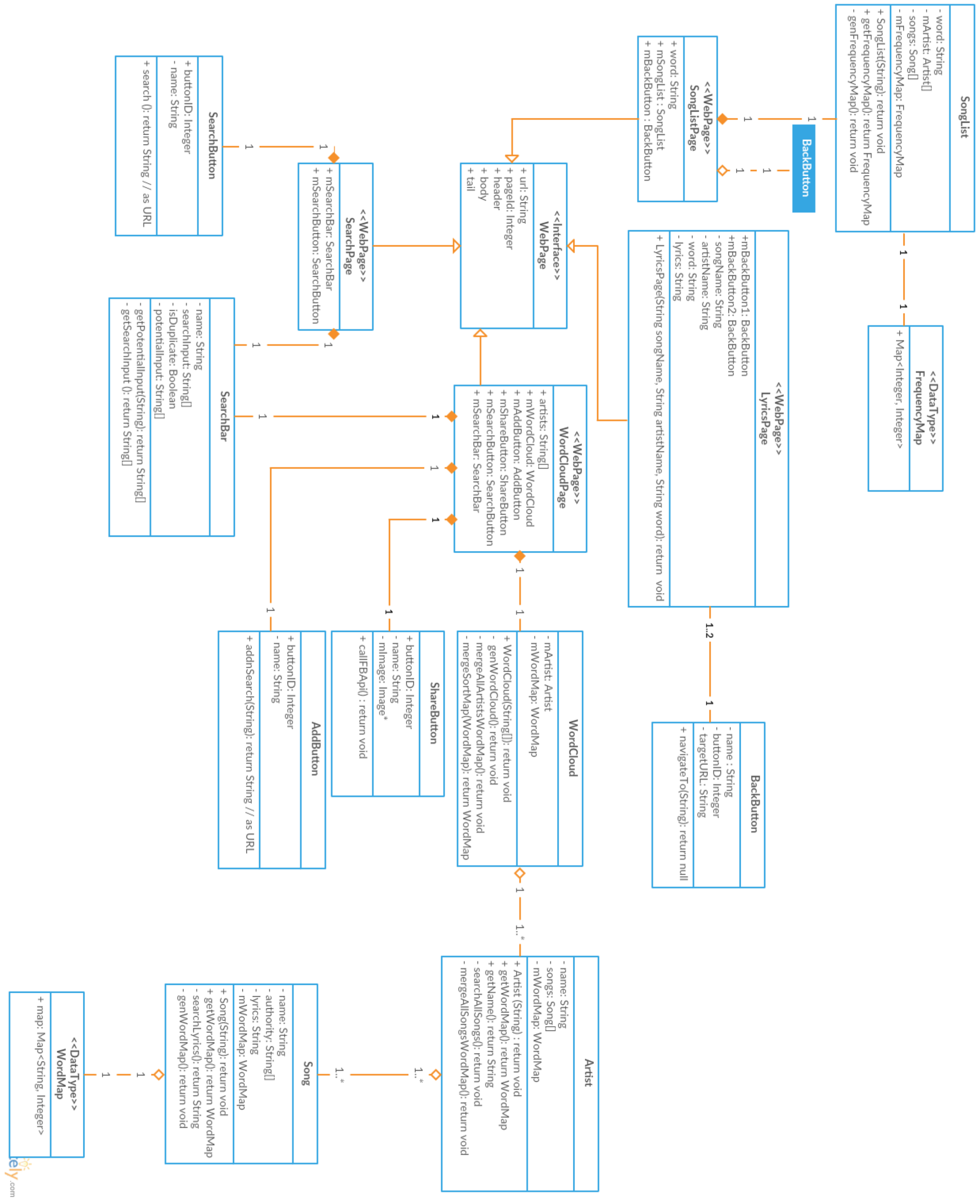
During the meeting on Feb 14th, we integrated all the subparts of Class Diagram to an entire Class Diagram and added explanation to Class. Then we started to write evaluations based on Lecture 7, “Design Principle”. Since we didn’t know that the requirement of evaluation when we were planning the project, we did not assign schedule on this. Luckily, we have had one extended day; therefore we met on Feb 15th to complete the entire documentation. Refers to PMP Monitoring Plan Task 11.

To monitor effort, we discussed the time we spent on researching on the first two meetings. The average effort time is one hour and eleven minutes per day per person which is lower than our expectation. The effort on Feb 15th is huge. Most of the team members spent more than three hours on completing the documentation.

Appendix II.
Data Flow Diagram During Meeting 1



Appendix III.
Class Diagram During Meeting 2


SongList
- word: String
- mArtist: Artist[]
- songs: Song[]
- mFrequencyMap: FrequencyMap
+ SongList(String): return void
+ getFrequencyMap(): return FrequencyMap
- genFrequencyMap(): return void
<<WebPage>>
SongListPage
+ word: String
+ mSongList : SongList
+ mBackButton : BackButton
BackButton
<<DataType>>
FrequencyMap
+ Map<Integer, Integer>
<<WebPage>>
LyricsPage
+mBackButton1: BackButton
+mBackButton2: BackButton
- songName: String
- artistName: String
- word: String
- lyrics: String
+ LyricsPage(String songName, String artistName, String word): return void
<<Interface>>
WebPage
+ url: String
+ pageId: Integer
+ header
+ body
+ tail
<<WebPage>>
SearchPage
+ mSearchBar: SearchBar
+ mSearchButton: SearchButton
SearchButton
+ buttonID: Integer
- name: String
+ search (): return String // as URL
SearchBar
- name: String
- searchInput: String[]
- isDuplicate: Boolean
- potentialInput: String[]
- getPotentialInput(String): return String[]
- getSearchInput (): return String[]
<<WebPage>>
WordCloudPage
+ artists: String[]
+ mWordCloud: WordCloud
+ mAddButton: AddButton
+ mShareButton: ShareButton
+ mSearchButton: SearchButton
+ mSearchBar: SearchBar
WordCloud
- mArtist: Artist
- mWordMap: WordMap
+ WordCloud(String[]): return void
- genWordCloud(): return void
- mergeAllArtistsWordMap(): return void
- mergeSortMapWordMap(): return WordMap
ShareButton
+ buttonID: Integer
- name: String
- mImage: Image*
+ callFBApi() : return void
AddButton
+ buttonID: Integer
- name: String
+ addnSearch(String): return String // as URL
BackButton
- name : String
- buttonID: Integer
- targetURL: String
+ navigateTo(String): return null
Artist
- name: String
- songs: Song[]
- mWordMap: WordMap
+ Artist (String) : return void
+ getWordMap(): return WordMap
+ getName(): return String
- searchAllSongs(): return void
- mergeAllSongsWordMap(): return void
Song
- name: String
- authority: String[]
- lyrics: String
- mWordMap: WordMap
+ Song(String): return void
+ getWordMap(): return WordMap
- searchLyrics(): return String
- genWordMap(): return void
<<DataType>>
WordMap
+ map: Map<String, Integer>
1
1..2
1..*