C-lyrics - A Word Cloud for Lyrics

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 Design Document describes the architecture and design of the C-lyrics software system. The intended audience is the development team, consisting of the six members whose names are on the cover of this document.

1.2 Overview

This document provides a layout of the different components, classes, state machines, architectures, designs, and other diagrams related to the C-lyrics software design. Each diagram is clearly explained in section 2 and 3 and

justifications for the particular design choices or component configuration choices are given in section 4. Metrics of quality are also discussed. In section 5, the appendices show the staff allocation plans from meeting notes in order to comply with standards set out in the Project Management Plan. Note that the figures used to illustrate diagrams for this document were made using the Gliffy tool [5].

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)

1.4 Definitions And Acronyms

Fill In From Old Stuff MVC FR-Functional Requirements PMP

2 System Architecture

This system architecture diagram displays the communication paths of C-Lyrics. There are three logical entities in our architecture: the browser, the server, and the external API. The clients exclusively interact with the front end browser running angular.js. The front end then makes requests to the server which communicates with the API to obtain artist data. These data are then communicated back through the server and displayed on the front end which then communicates with the clients.

2.1 Data Flow Diagram

Above is the data flow diagram for C-Lyrics. Data in the form of requests are sent to the front end (Browser) as user input. This input is sent to the server and processed into API requests by the back end PHP. The input can be of various types (i.e. artist searches, word selections, song selections, etc.) and the processed API request goes to the external API and returns the relevant song/lyric data. This data is formatted by the backend into the proper form of output (song list, word cloud, etc.) which is then displayed by the front end browser. The client then receives the output as acknowledgement.

3 System Design

3.1 UML Class Diagrams

3.1.1 PHP Class Diagram (Server Side)

The PHP class diagram illustrates the connection between the EchoNest_Client, which is an instance of the EchoNest API, the EchoNestConnection, along with the subclasses Lyrics and Autocomplete.

The EchoNestConnection class will act as the parent class that has a single, static connection to the EchoNest API. Both the Lyrics and Autocomplete will inherit from the EchoNestConnection to query the API.

The class Autocomplete will satisfy the functional requirement of Search Bar autocompletion. Before the user clicks on the Submit button to generate the word cloud, the front end will send the current user input to the getsuggestions php file containing an instance of the Autocomplete class. The Autocomplete class will then process the input by using the getSuggestions function, which returns a list of artists that contain the user input to the getsuggestions page in json format. The list of artist names will be sent back to the front end to be used as autocompletion suggestions in the Search Bar.

Once the user clicks on the Submit button, the front end will send the final artist name to the getlyrics php file containing the instance of the Lyrics class. The song lyrics for every song by that artist will be generated by the querySongs function and returned to the getlyrics php page in json format. The front end will then parse the json data returned on getlyrics and generate the Word Cloud according to requirements stated in the SRS document.

3.1.2 Client Side Class Design

The class diagram exposes the principal relationships between the main components of the application. The diagram includes some specific annotations according to the UML standard. The only slight modification made to the standard occurs with standard dotted lines; the meaning is that the two connected components are “semi-codependent” - they work separately, but it really makes sense to match them together. Note that the diagram also includes some components defined by AngularJS, as those components really help to make sense of the overall design choice.

The layout of the diagram also shows how AngularJS enforces the MVC pattern. Models are on the left, views on the right and controllers in the middle. A “closed” arrow means inheritance, whereas an open one involves dependency. Also note that a plus is the sign for a publicly accessible method or attribute, whereas a minus is private.

An interesting pattern choice is that the HomepageController is the only component having access to the services and factories. It might appear differently to the user, for example if he loads a songs page he previously loaded. The page should still be displayed as it is in the cache of the browser. However, strictly speaking the user will first be redirected to the Homepage which will interact with the cache and present the data to the requested page.

3.2 UML Component Diagrams

The diagram above illustrates the various components in the C-Lyrics system. These components are key features the user will interact with during their experience with the C-Lyrics system. It demonstrates the three interfaces that the user interacts with, such as the Home Page, the Songs List Page, and the Lyrics Page, as well as which components each interface has access to. Additionally, the diagram lays out which components each of the back end communication outlets interacts with.

3.3 UML Use Case Diagrams

In the figure above, a use case diagram is used to represent all interactions a user, noted as "Actor/Actress", will have with the C-Lyrics system. The interactions a user will have are as follows:

- Submit Search: A user will interface with the Home Page and can submit their search in the Search Bar to the C-Lyrics system.
- Share Word Cloud (WC) to Facebook (FB): A user will interface with the Home Page and can share their generated WC to FB.
- Add to Cloud: A user will type in the Search bar and add to their initial search. This will generate a new WC that can be displayed to the user.
- Select Word: A user will select any chosen word from the generated WC that can lead them to interface with the Songs Page.
- Select Song: On the Songs Page interface, a user can select any song from the generated songs list, which will lead them to interface with the Lyrics Page.
- Go to Home Page: A user that will interface with either the Songs Page or Lyrics Page can be directed back to the Home Page.
- Go to Songs Page: A user that will interface with the Lyrics Page can be directed back to the Songs Page.

3.4 UML State Machine Diagrams

In the state machine diagram above, there are five possible states once the C-Lyrics system is started. In order to move to another state the conditions listed must be fulfilled, each condition is represented as a boolean variable for simplicity. When the user interacts with the C-Lyrics system, the value of various variables changes depending on the action performed. Below is an in-depth description of each of the five states.

- Home State
  - This is the beginning state for the system and represents when the user first accesses C-Lyrics. In this state there is no WC displayed, only the Search Bar and the Submit Button are on the screen. The only actions the user can take from this state are to type in the Search Bar and to press the Submit Button.
  - While the Submit Button is not pressed, submit = false and the state will not change. If the user presses the Submit Button making submit = true then the state will change dependent on the value of the type_artist variable.
  - The type_artist variable represents the truth value of what the user typed into the search bar. If the user entered a valid artist name, meaning the artist was found by the API, then type_artist will be true, otherwise it will be false. If type_artist is false then the state will change to Error Message Visualization State, but if it is true then the state will change to Word Cloud Visualization State.
- Error Message Visualization State
  - This 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. The only actions the user can take from this state are to type in a new artist name to the Search Bar and to press the Submit Button.
  - The state will change to Error Message Visualization State every time type_artist = false and either submit = true or add_to_cloud = true.
  - The the only way to exit this state is when the type_artist and submit variables are both true.
- Word Cloud Visualization State
  - This state represents when the user is on the Home Page and a WC is displayed. From this state the user can perform multiple actions some of which will not lead to a state change, as described in the bullet points below, and some of which will lead to a state change.

  - If the user types an artist name and both type_artist and submit are true then the user will stay in this state, but a new WC with the new artist's information will be displayed. However, if the user types an artist name and type_artist = false but submit = true then the state will change to Error Message Visualization State.
  - If the user types an artist name and both type_artist and add_to_cloud are true then the user will also stay in this state, but the word cloud will be modified to include the information from the second artist. However, if the user types an artist name and type_artist = false but add_to_cloud = true then the state will change to Error Message Visualization State.
  - If the user presses the share button making share = true, then the user will stay in this state, but will be able to share the image to Facebook via the Facebook API.
  - The state will change to Songs State when the user clicks a word in the word cloud making click_word = true.

- Songs State
  - This state represents the list of songs that the selected word appears in by any given artist. The user can either select a specific song title or press the Back to Cloud button from this state, both of these actions will lead to a different state.
  - If the user selects a song then select_song = true and the state will change to Lyrics State
  - If the user presses the Back to Cloud Button making back_to_cloud = true then the state will change to Word Cloud Visualization State.

- Lyrics State
  - This state represents the lyrics of the song that was selected in the Songs Page state. The only actions the user can take from this state are pressing the Back to Cloud Button and the Back to Songs Button.
  - If the user presses the Back to Songs Button making back_to_songs = true then the state will change to Songs Page.
  - If the user presses the Back to Cloud Button making back_to_cloud = true then the state will change to Word Cloud Visualization State.

3.5 UML Sequence Diagrams

In the figure above, a sequence diagram is used to represent one of the processes, namely the Select Songs feature, within C-Lyrics. This sequence diagram shows how the process will interact with the server and the EchoNest API in order to arrive at the Select Songs feature. Several terms used in the diagram can be referenced in section 1.4. The diagram is as follows:

- The user will submit their search to the server.
- The server will send a request to the API.
- The API will receive the request based on the information given from the user, and send all data on that search to the server.
- The server will then generate a Word Cloud (WC) and display it back to the user
- The user can select a word on the WC.
- The server will take in the request from the user and generate the second interface, the Songs Page.
- The user can select a song from the Songs Page list.
- The server will take in the request from the user and generate the third interface, the Lyrics Page.

4 Design Evaluation

4.1 Validation

The following list of requirements comes from the Functional Requirements (FR) section of the SRS. Next to each requirement is a reference to where in this design document that requirement is satisfied.

- FR1. Web Application-Search Bar: sections 3.1.1 and 3.1.2, front and back end search bar functionality
- FR2. Web Application-Word Cloud: sections 3.1.1, 3.1.2 and 3.4 (state machine)
- FR3. Web Application-Song List: sections 2.1.1 and 3.4
- FR4. Web Application-Lyrics: sections 3.1.1, 3.1.2, 3.2 and 3.4
- FR5. Access Through Web:
- FR6. Web Application-Share
- FR7. Web Application-Add to Cloud

4.2 Justification of Design Choices

4.2.1 Data Flow Diagram Evaluation

The initial level of discourse chosen for the abstraction of our data flow design is of the application as a whole (step-wise refinement). The data is viewed as travelling from module to module and from function to function. Each module encapsulates smaller modules and functionalities which are hidden from the user of the application. The design exhibits both strong coupling and cohesivity making it strong in modularity. For example, the front and back end display communicational cohesion as well as procedural and sequential cohesion. The front and back end also display data, control and common coupling as the data passed from the user is simple text. This data coupling and communicational cohesion is also present between the front end, back end and API. The backend and API design is of high quality since the API is essentially a black box whose functionality is a secret that neither the front end or back end are aware of, thus reducing coupling between these elements and the API.

4.2.2 UML Class Diagram Evaluation

These design principles for the php backend create optimal abstraction because of the separation of the API client and the classes making the queries. The Lyrics and Autocomplete php classes each have one purpose and are broken down to their simplest levels, which only return the type of requested data in json format. This reduces the complexity of each class and reduces the number of lines of code, following Object Oriented Principles of class simplicity. Both classes use communicational cohesion and external coupling by operating on the same instance of EchoNest_Client in EchoNestConnection. In itself, the backend is a black box for the user. The Lyrics and Autocomplete classes do not rely on each other, only on the parent class, and can work independently, reducing the risk of any problems associated with coupling.

By using the AngularJS framework, the design strongly enforces an MVC pattern. This allows to clearly decouple some parts of the application in order to gain flexibility and modularity. This schema is underlined by the different columns of the figure, where the system's communication, logic and visual interface are separated.

By using directives, the systems allows for reusability of components as well as a template-directive hierarchy. For example, the ActionButton class will be used for both the submit button and the add to cloud button. In addition, the visual template used (ButtonView) is also shared by the NavigationButton class, which in turns will be used for both the back to cloud button and the return button.

Finally, this design provides an easy way to extend the application. In the case where some new functionality should be implemented, the problem could be

approached in two ways. Either the client would like to enhance the current web pages, in which case few modifications to the controller, template and a new factory should provide enough flexibility for the implementation. Another approach would be to add a new web page to the application which would mean creating a new template and controller. In both cases, the offered level abstraction by the system allows to easily implement one's own features in the system. In addition, the use of existing components is simplified as for each of them understanding their interface only requires to understand their public methods. This approach of hiding unnecessary information is also greatly simplified thanks to the use of AngularJS.

4.2.3 UML Component Diagram Evaluation

The component diagram, as seen in section 3.2, is an abstraction of all of the main parts of the C-Lyrics system and how they interact with the user through each interface as well as how they interact with the back end of our system through either the server or chosen APIs. The features chosen to break into components by following the communicational cohesion model, features that interact with the same data are coupled together. Furthermore, the diagram follows the common coupling model because any component that shares data through a given interface or back end communication outlet is linked together. These methods of cohesion and coupling lend themselves to proper information hiding because only components that need to share data have access to it. There is no hierarchy in this diagram, but its intra-modular approach is a very simple way to display the components of C-Lyrics and how they are connected.

4.2.4 UML Use Case Diagram Evaluation

The purpose of a class diagram is to create a high level representation of the interaction a user will have with the C-Lyrics system. The class diagram, referenced in section 3.3, represents this well by abstracting all key functions the system offers and allowing the user to visualize what types of interactions are possible. Hidden information that is not needed for function understanding is implied by these abstractions.

4.2.5 UML State Machine Diagram Evaluation

While runtime complexities make our system difficult to accurately represent as a finite state machine, the main functions of our system can be abstracted to fit the finite state machine model as explained in section 3.4. The modularity of each state is based on logical cohesion, each state is defined by its functionality within the system as the whole, and control coupling, the actions performed in one state directly drive the transition between states. Although the diagram is not laid out in any specific hierarchy, there are four hierarchical levels in the

state machine. The highest level is the Home State followed by the Word Cloud Visualization and the Error Message Visualization states, followed by the Song State and finally the Lyrics State.

4.2.6 UML Sequence Diagram Evaluation

The sequence diagram that can be found in section 3.5 demonstrates one of the processes that a user will go through while using C-Lyrics. The select song functionality was chosen against other functionalities because this process best portrays the capabilities of the C-Lyrics system, demonstrating the natural direction of requests and generating pages. The class diagram also represents modularity, specifically procedural cohesion that shows the order in which a process is being done. Multiple types of coupling are also utilized including content coupling, having certain processes as predecessors for others which can directly affect a process, and common coupling, in which all data received from the EchoNest API is shared with the interfaces. Although the sequence diagram does not have a hierarchical level, it demonstrates the process flow for the selected features as an intra-modular approach.. One process follows another and will be completed once the predecessor has finished the request and displayed the request back to the person.

5. Appendices

5.1 Design Process

5.2 Meeting Documentation

Based on the Project Management Plan, we followed our consistent current group process. We agreed to meet as a group through various communication channels and established a meeting time and location. At each of our meetings we conducted planning and task division.

February 17, 2015 Met in Annenberg to discuss and plan for our next meeting. Also did preliminary task division and abstract discussion of architecture and diagrams.

February 18, 2015 Met in Leavey Library with all members of the group. Tasks were divided as follows: Justine and Kelsey: System Design and Evaluations Mark: PHP Class Diagram and Evaluation Sebastien: Angular JS Class Diagram and Evaluation Jeff: Architecture and Data Flow Diagrams, Appendix Milad: Introduction, Data Flow Diagram Evaluation, Appendix