myFaves

Final Design Document

By

The Creepersss

Andrew Jackson, Matthew Brannick, William Casey,
Josh DeVinney, T.J. Moats, Devin Mullennix,

Table of Contents

Table of Figures 2
Project Approach: 3
Task Delegation: 3
Tools and IDE's Used: 4
Software Organization: 5
UML Diagrams: 7
Server Design: 8
Server Multi-Threading 9
Data Dictionary: 10
User Interface Mockups: 13
Usability Metrics: 14
User Responses: 14
Usability Problem and Defect Management: 15
Screenshots of Functioning Product: 15
Web Application: 15
Desktop Application: 17
WOW Factor: 18
Stumbling Blocks and Setbacks: 18
Server & JavaScript Setbacks 19
Web Development Setbacks: 19
Desktop Client Setbacks: 19
Additional Implementations: 20

Table of Figures

Figure 1: Software Architecture Diagram..... 5
Figure 2: Design Overview..... 6
Figure 3: Entity Relation Diagram..... 7
Figure 4: Use Case Diagram..... 7
Figure 5: Server Design Flowchart..... 8
Figure 6: Website Mockup..... 13
Figure 7: Desktop Application Mockup..... 14
Figure 8: Website Homepage..... 16
Figure 9: Website User Login..... 16
Figure 10: Website Sample Search Results..... 16
Figure 11: Desktop Home Screen..... 17
Figure 12: Desktop Sample Search..... 17
Figure 13: Desktop Adding a Song to a Playlist..... 18

Preface:

The goal of this project is to develop software that allows users to create playlists from a master list of songs and allow users to search the database of songs based on different criteria. Implementation must include a 3-tier software architecture solution including a database holding relevant data, a servlet to handle data retrieval and updates, and at least two user clients including, but not limited to a website application, a mobile application, or a desktop application. The database layer will be implemented in MySQL. The servlet will be in Java and use JDBC to connect to the database.

Project Approach:

To implement myFaves, we took a minimalist approach to software development. In the three tier software architecture design, we decided that there should only be one servlet to handle requests from each client's platform, there should only be one server and that the front end GUI should be as consistent across platforms as possible. Source control would be used so that team members could collaborate, publish changes and roll back to earlier versions if necessary.

Task Delegation:

To utilize each team member's talents, we decided to split the group into different teams. Each team is responsible for one of the tiers involved in this project's three tier architecture. There is a server team, a web team and a desktop team. The server side team's responsibilities included setting up the mySQL database, running tomcat and tweaking the different connector classes. This team was also responsible for the operation of the Linux server that served as our host. The web development team was responsible for the creation of the myFaves website and the scripts associated with JavaScript and JSON. The Java team was tasked with creating the desktop client using JFrame. Documentation would not be the sole responsibility of any team member. While each team member was expected to contribute some information to the final design document, some members took extra time to compile and organize all of the entries and diagrams generated throughout the course of the project.

Tools and IDE's Used:

Considering the size and scope of this project, several third party tools were considered to ease development. For the conceptual stage, UML and software architecture diagrams were created using *Software Ideas Modeler* from softwareideas.net and Creately from creately.com. The team members responsible for diagramming used their preferred diagram modeling tools. This did not create problems as the finished diagrams were published as images and not software-specific file types.

During the programming process the server side, web development side and java side all used different Integrated Development Environments. The mySQL database was created in using the mySQL workbench from wb.mysql.com. The mySQL workbench was chosen for this project because it offered functionality not present in the Ubuntu g++ text editor. Workbench allowed us to check for errors and compare generated ER models to our conceptual ER models.

Adobe Dreamweaver CS5 was used to create the myFaves website. Dreamweaver was chosen because it allowed us to abstract away the difficulties in building a website from a blank text file; Dreamweaver made the editing of objects and fields dramatically easier than modifying tags line by line. The *Firebug* web development tool was especially helpful in debugging the various JavaScript methods of the myFaves website. Lastly, on the web development side, color wheel tools were used to find split complementary color schemes for the myFaves website. Such color schemes are said to be the most attractive to human eyes from the field of color theory; they are also the most legible for text and object recognition.

The Java and JavaScript files were developed in BlueJ from La Trobe University and in Eclipse, the open source project. BlueJ proved especially helpful in the fact that it allows developers to test individual methods even if the project is not completed or continues to experience compilation errors. For example, we were able to test several methods related to the display of song information which was retrieved from the server. Despite the fact that the mySQL connector was not completed, BlueJ was able to pass the methods test data allowing us to test its functionality before the connector classes were implemented. Eclipse provided the same benefits as BlueJ, except for individual method testing. Eclipse is a useful tool in object oriented programming because it has an auto

suggest tool for code completion. This allowed the Java development team to save time when programming.

As for Source Control, we decided to use Git Bash and create a public repository on Github.com. Git proved to be a valuable tool for a project of this scale. Github made sharing and collaboration possible. Initially setup was difficult because of Git and Github's various security requirements but there were few problems with the source control software after it was set up properly. Git had proven especially useful in rolling back changes to our code when needed; this would not have been possible if we were not using source control.

Software Organization:

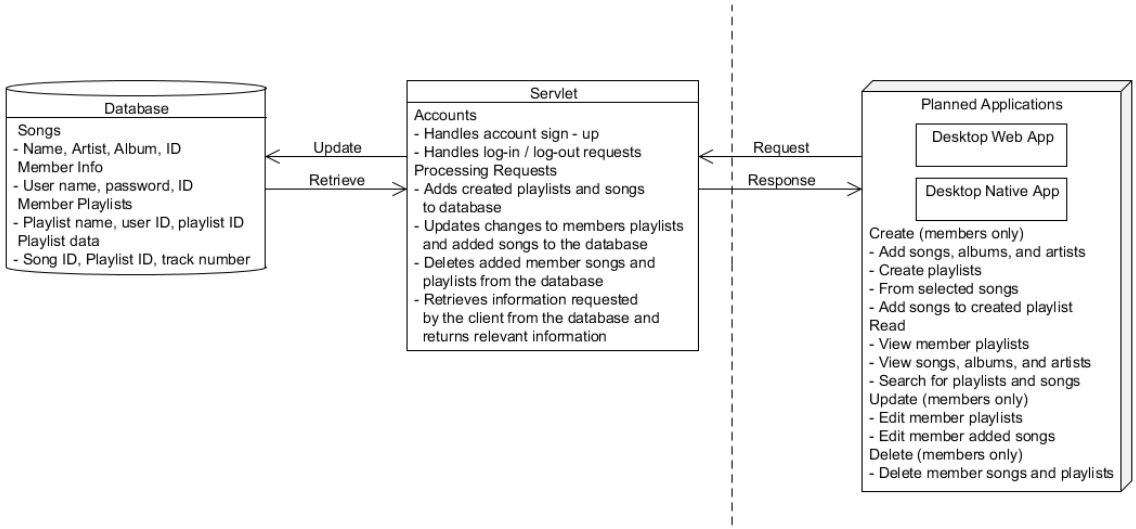

Figure 1: Software Architecture Diagram

Shown above is our software architecture diagram. For this project, we decided to focus on creating a strong foundation of software on which we can expand later to increase functionality. The project detailed the requirements for a database, servlet, and two user applications, but allowed us to define the implementation.

The users and visitors of our application use either a web-based desktop application or a native desktop application to request information about the database from the servlet. The web app will be based on HTML with JavaScript AJAX functionality and CSS for look-and-feel. The native desktop app will be programmed in Java using the Java

Swing libraries. The clients will allow users to log-in, search and browse available songs and member playlists, and create new playlists if they are signed in. All requests will be in HTTP form.

A single servlet will handle the clients' request; it will respond to the client with the appropriate response and, if required, will execute MySQL commands to interact with the database, retrieving, adding, and updating data. Responses to the client will be in HTML form with JSON implemented as needed for requested information.

The database will hold all relevant information about the software solution. It contains information about each song, playlist, and member. The specific design of the database is detailed later in this report.

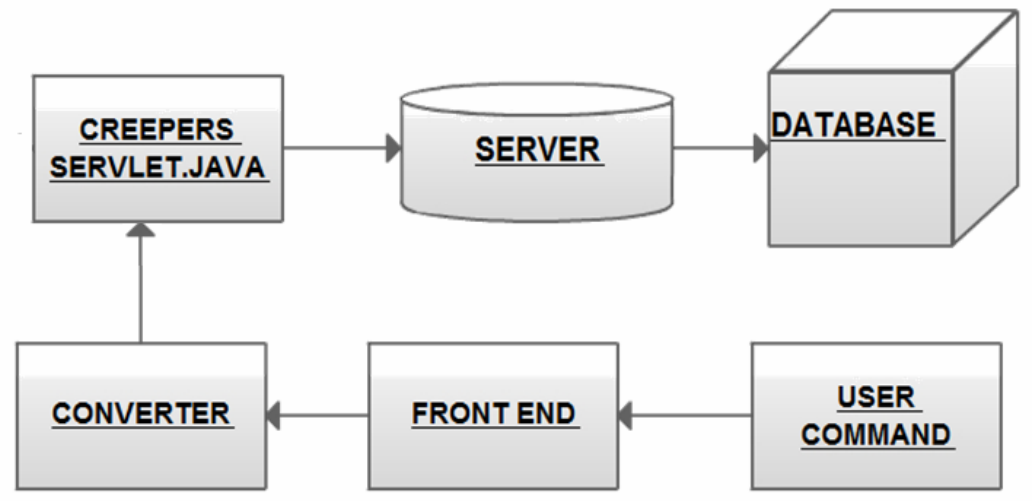

Figure 2: Design Overview

UML Diagrams:

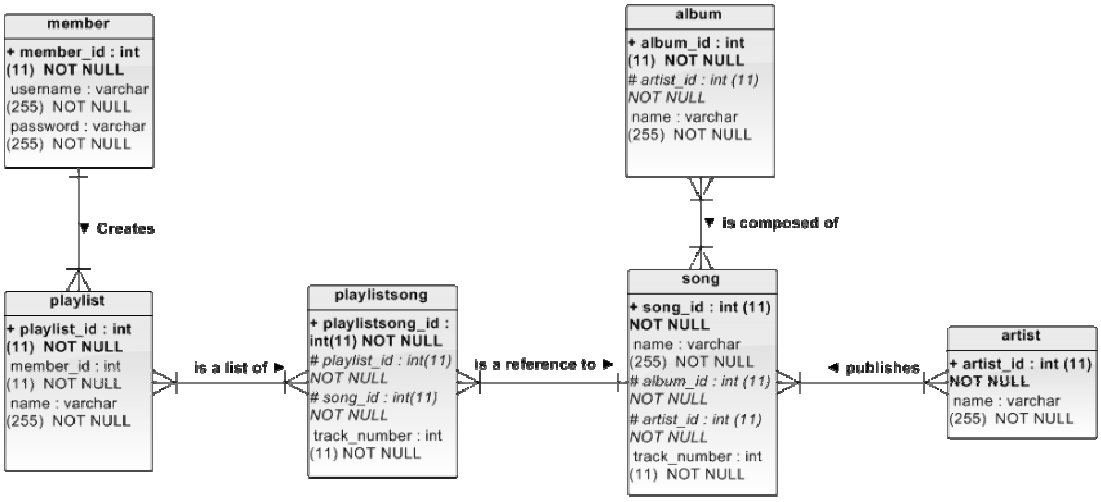

Figure 3: Entity Relation Diagram

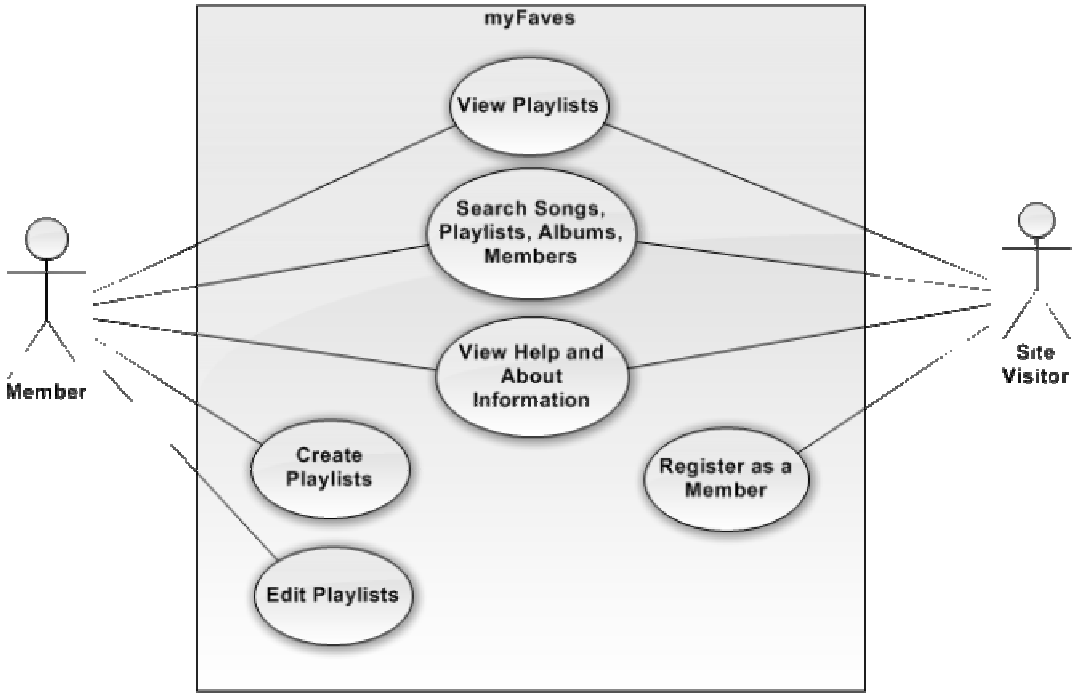

Figure 4: Use Case Diagram

Server Design:

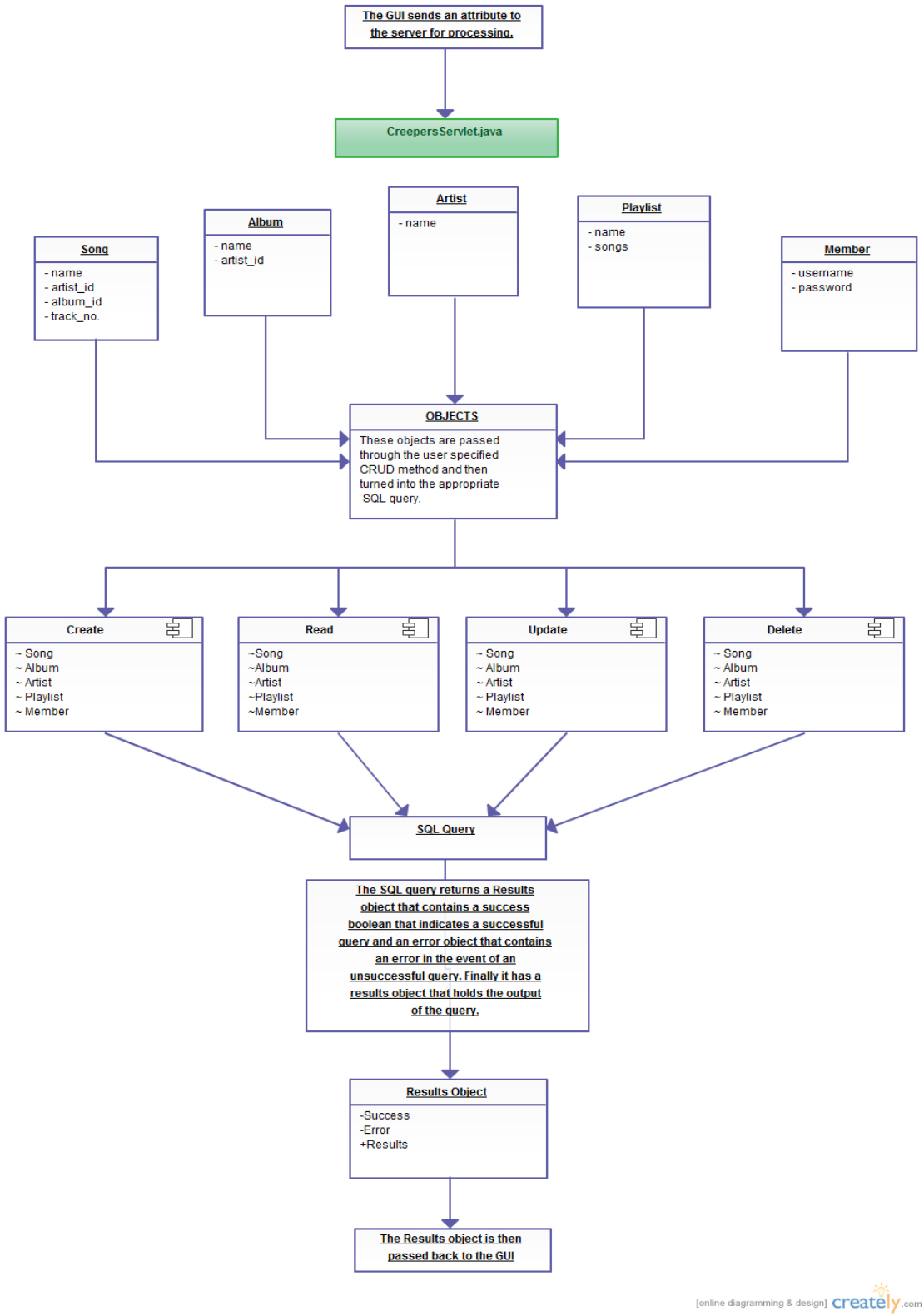

Figure 5: Server Design Flowchart

Server Multi-Threading

Apache tomcat runs natively using a **thread pool**, meaning that the server relieves itself from directly managing its threads. Instead of allocating new threads; whenever it needs a thread it asks for it from the pool, and when it is done, the thread is returned to the pool. The thread pool can now be used to implement sophisticated thread management techniques, such as:

1. Keeping threads "open" and reusing them over and over again. This saves the trouble associated with creating and destroying threads continuously.
    - Usually the administrator can instruct the pool not to keep too many idle threads, freeing them if needed.
2. Setting an upper bound on the number of threads used concurrently. This prevents the resources allocation problem associated with unlimited thread allocation.
    - If the container maxed out to the threads upper limit, and a new request arrives, the new request will have to wait for some other (previous) request to finish and free the thread used to service it.

This default setup allows our server to handle a large number of concurrent requests, improving its performance at very little cost. In the case of our implementation: every connection from a user is handled by the thread pool, whereas the connections to the SQL server is handled by the JDBC which queue's each command to the database, implementing them in the order in which they are received.

Data Dictionary:

Description of Entities:

Entity Name	Description	Aliases	Occurrence
artist	The term for a composer/ performer of a song or groups of songs in the case of an album		A performer is responsible for composing one or many songs and/or albums
album	A collection of songs released under the same label		An album has from one to many songs contained in it.
member	A user to the website/application		Each member can create many playlists or none.
playlist	A collection of assorted tracks chosen by the user		Each member can have no playlists or as many as they want.
song	A song is an object that is made/belongs to an album and an artist		Each song can be released under different albums but it cannot be released twice in the same album.
playlistsong	An instance of a song found in a user's playlist		Each playlist can have one to many references songs

Description of Attributes

Entity Name	Attributes	Description	Data Type & Length	Nulls	Multi-Valued
artist	artist_id	uniquely identifies an artist	11 digit integer	no	no
	name	artist the artist's name	255 variable characters	no	no
album	album_id	uniquely identifies an album	11 digit integer	no	no
	artist_id	uniquely identifies an artist	11 digit integer	no	no
	name	the name of the album	255 variable characters	no	no

member	member_id	the unique identifier of a member	11 digit integer	no	no
	username	the member's desired username	255 variable characters	no	no
	password	the members desired password	255 variable characters	no	no
playlist	playlist_id	the playlist's unique identifier	11 digit integer	no	no
	member_id	the member's unique identifier	11 digit integer	no	no
	name	the playlists name	255 variable characters	no	no
song	song_id	the unique identifier of the song	11 digit integer	no	no
	name	the title of the song	255 variable characters	no	no
	album_id	the unique identifier of the album	11 digit integer	no	no
	artist_id	the unique identifier of the artist	11 digit integer	no	no
	track_number	the unique identifier of the song	11 digit integer	no	no
playlistsong	playlistsong_id	the unique identifier of the playlist song	11 digit integer	no	no
	playlist_id	the unique identifier of the playlist	11 digit integer	no	no
	song_id	the unique identifier of the song	11 digit integer	no	no
	track_number	the unique identifier of the song's track number	11 digit integer	no	no

Description of Relationships:

Entity Name	Multiplicity	Relationship	Multiplicity	Entity Name
artist	1..*	composes	1..*	song
	1..*	has an	1..*	album
album	1..*	is composed of	1..*	songs
	1..*	is made by	1..*	artist
member	1..1	makes a	1..*	playlist
	1..1	adds a	1..*	song
playlist	0..*	contains	1..*	song
	0..*	is owned by	1..1	member
song	1..*	is composed by	1..1	artist
	1..*	is part of	1..*	album
playlistsong	1..*	is a reference to	1..1	song

User Interface Mockups:

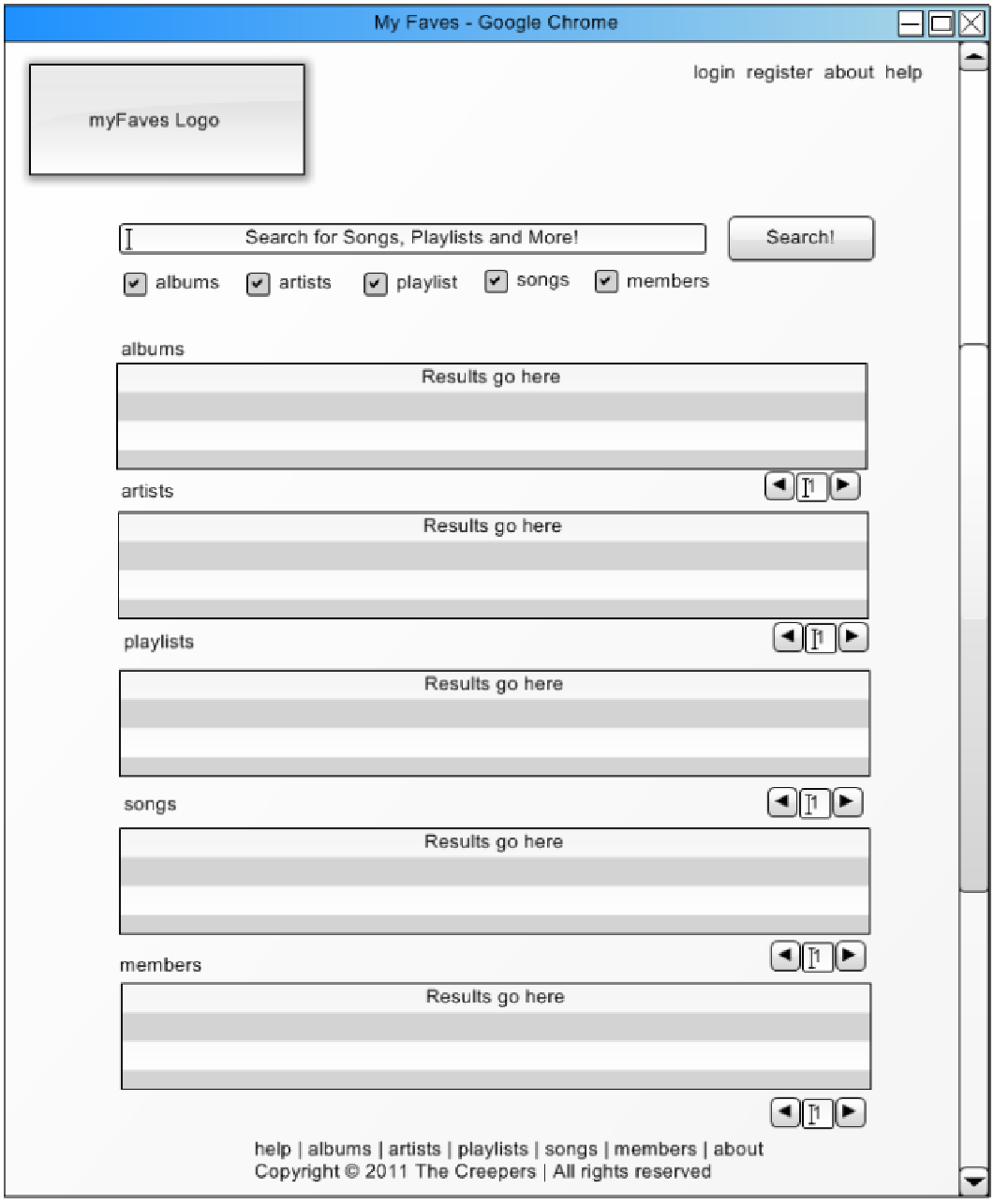

Figure 6: Website Mockup

The figure above is a prototype of our intended design for our desktop application. Aiming for a simplistic approach, there is only one main window from which all can be accessed. Logging in is handled by a simple dialog when “Log-in” is clicked on the top right of the window. Once a user is logged in, the button switches to log out and welcomes the user.

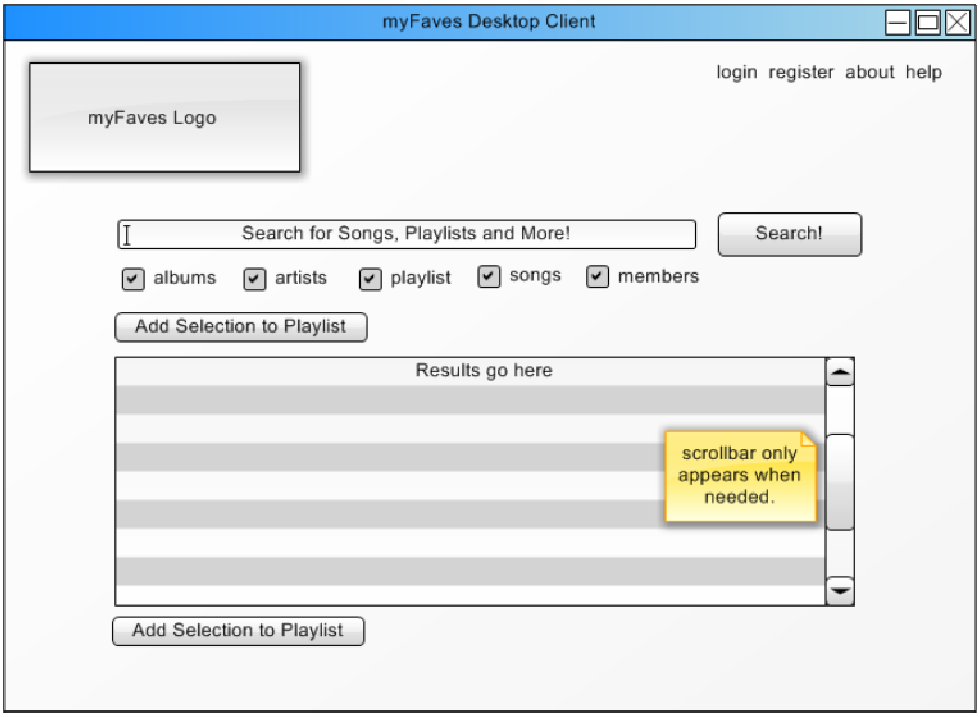

Figure 7: Desktop Application Mockup

The main feature will be the search bar which will allow the user to search for any song, playlist, or other member and display the info in the table below. The table will have customizable columns based on attributes like song name, album, artists, and are alphanumerically sort able depending on the chosen column.

Usability Metrics:

Following the "Test Early, Test Often" rule of good web design, we decided to demo the program to several different people during the development process. The questions posed to the users were open ended in nature, so that the users wouldn't be biased towards generating simple answers; there were no leading questions. Responses were written down and shown to the developers of the desktop GUI and website.

User Responses:

- Once you select & view a list, there's no back button to get back to the search results.
- The size of the table changes depending on results. It looks weird.

- When you select an artist, there is no indication that you are looking at albums, the column just says ‘name’. Also the song count for the albums always is 0.
- Adding a new song to a playlist causes and error that says duplicate songs are in the playlist.
- When you refresh the page, the numbering of the playlists is off.
- The help/about/support info on the footer is nice.
- It’s Fabulous!

Usability Problem and Defect Management:

MyFaves was designed to gracefully recover from errors during runtime. The Java and JavaScript programs all implement try, catch and finally blocks of code to handle exceptions. In the web development area, firebug was used to find common errors and address them before publication.

Screenshots of Functioning Product:

Web Application:

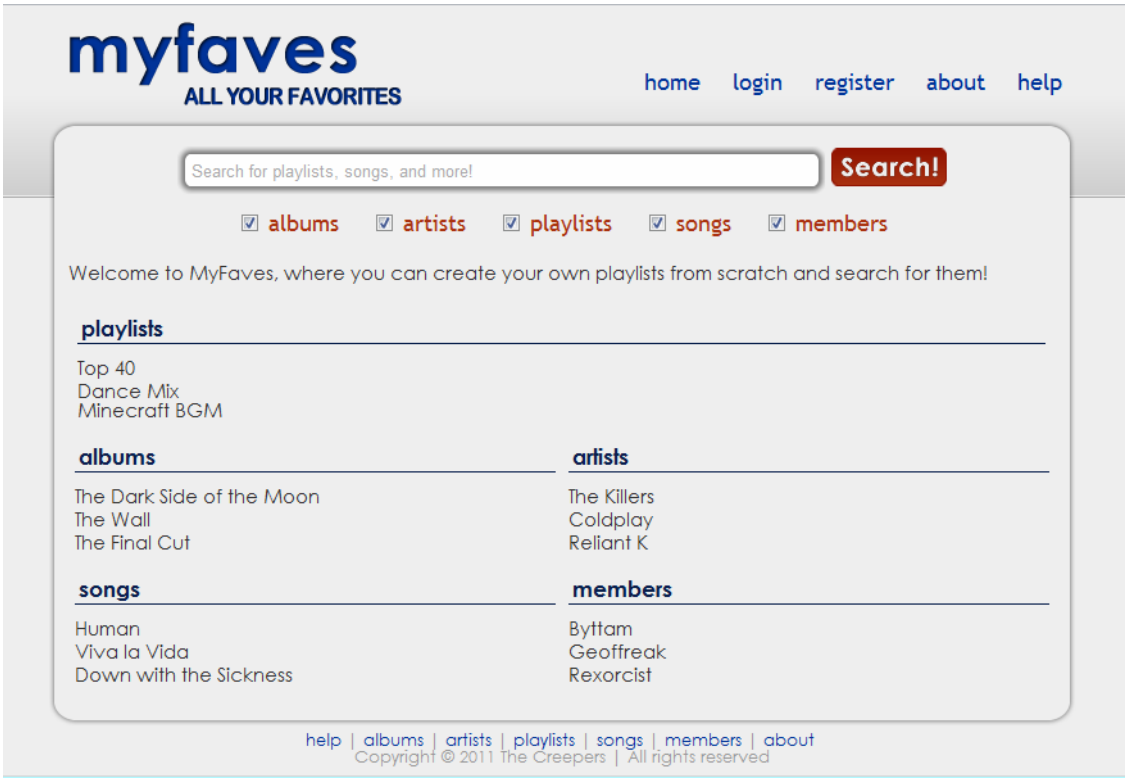

Figure 8: Website Homepage

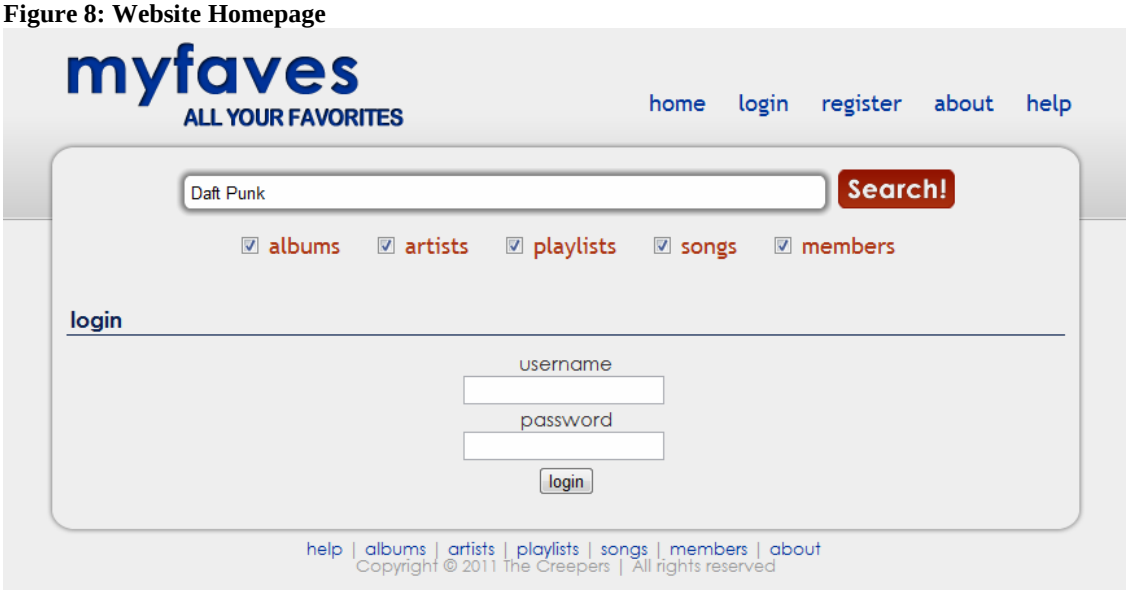

Figure 9: Website User Login

Figure 10: Website Sample Search Results

Desktop Application:

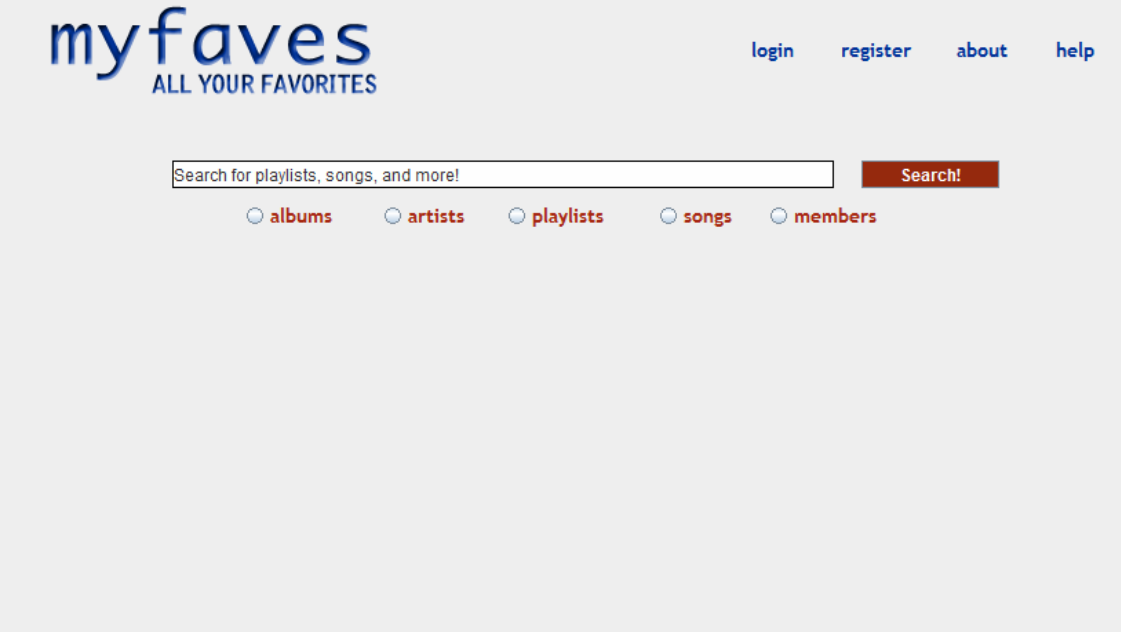

Figure 11: Desktop Home Screen

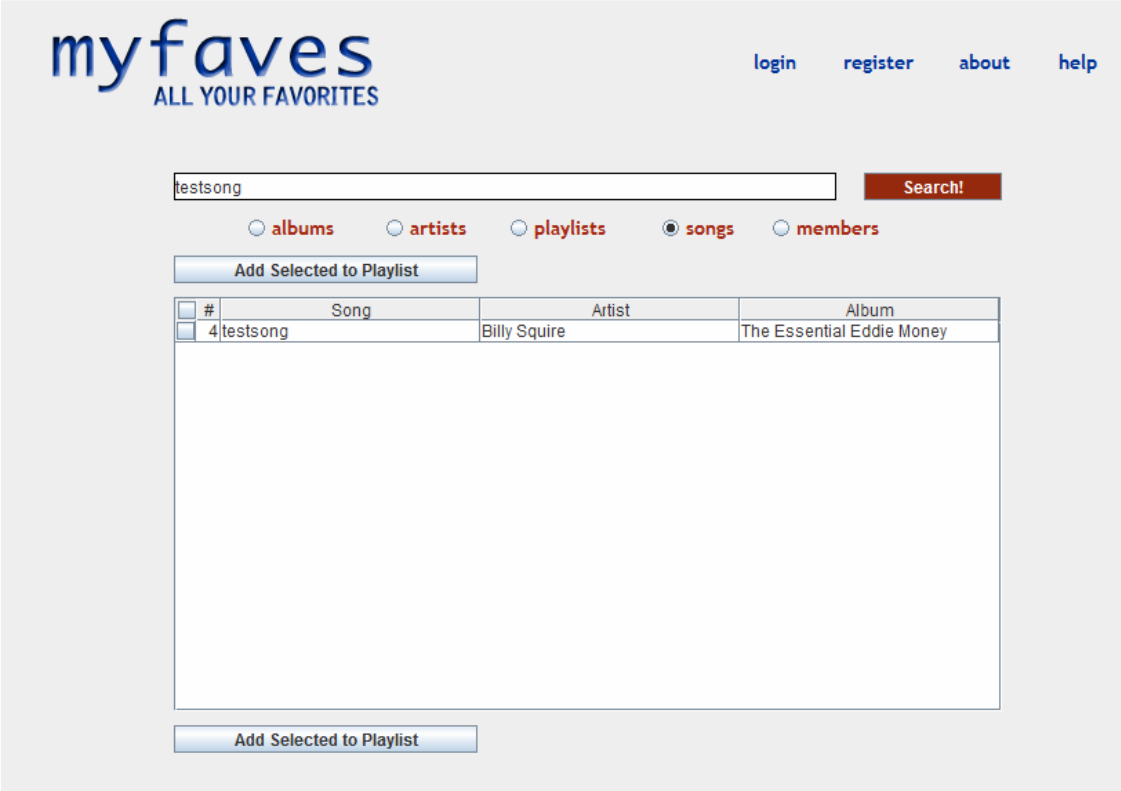

Figure 12: Desktop Sample Search

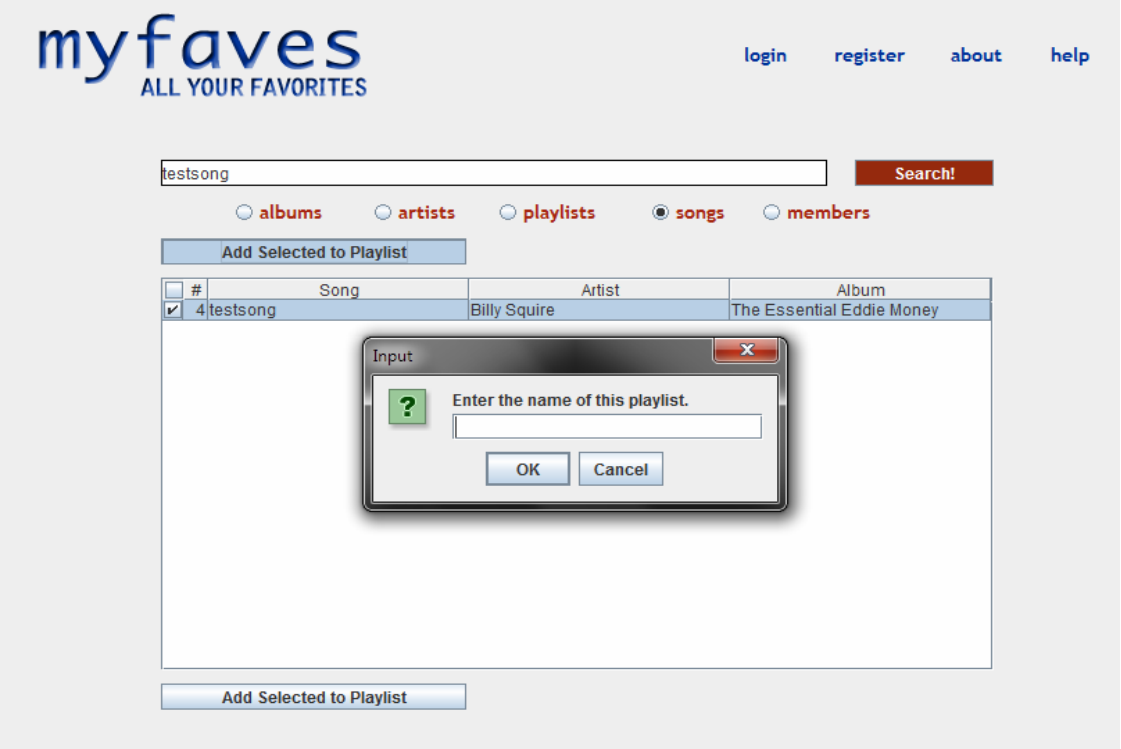

Figure 13: Desktop Adding a Song to a Playlist

WOW Factor:

The myFaves website is fully operational on both iPhone and Android mobile browsers. MyFaves is accessible from the World Wide Web, desktop clients and mobile devices, with no loss of functionality between the platforms. We created an API for all of the classes in the Java connector program we wrote, which will greatly ease third party developers should they chose to build onto the myFaves platform. The infrastructure is in place for future dedicated mobile applications, so the development of such applications will require no major changes to our current code.

Stumbling Blocks and Setbacks:

While Git's handling of our project files was very helpful, there were several occasions when it made the development process more difficult. Users cannot push to Github unless they have pulled recently, this created some frustration and commits had to be put on hold until changes were pulled from the master branch on Github. In addition, Git corrupted two files during this project. One of the files corrupted was the power point presentation for myFaves. We had to roll back to an earlier version, which meant we lost

work and had to redo much of the content in the PowerPoint. The second corrupted file was a .java file. We were able to revert to an earlier version because of Git; thankfully, little was lost because of the rollback.

Server & JavaScript Setbacks:

While trying to get the java server up and running, we encountered several problems with the Tomcat setup. No one had ever written code for a tomcat server, nor did we know which parameters to use when compiling the code. In the end this was remedied with a regular expression based compile to grab the libraries that were used in the project.

Another pitfall we encountered was the creation of the cookies by the tomcat installation. Because of the way tomcat handled the cookies we were unable to specify our own parameters for them, thus we had no way of handling the cookies client side you can't set a cookie before you connect in the default java API http connector. To remedy this we decided to use Apache's http connector class.

Web Development Setbacks:

The development of the myFaves website went much more smoothly than in the other areas of this project. That being said, there were still several problems relating to browser compatibility and JavaScript functionality. During usability testing, we found that older versions of Mozilla Firefox did not display transition animations smoothly, while updated versions did. While not a major bug, the problem could be considered somewhat jarring to the user. That being said, with automatic updates, the majority of web client users should view the website with no problems.

Desktop Client Setbacks:

Several setbacks were overcome in the course of developing the desktop interface of myFaves. The exact placement of JFrame objects such as text fields, buttons and textboxes was time consuming but not especially difficult; the look of the desktop application had to closely resemble the document object model of the myFaves website. Code wise, JTable proved especially temperamental. JCheckBoxes in the select songs

column was especially difficult and time consuming to implement. Getting the desktop client to communicate with the server through a Java connector class was also time consuming; it required a lot of changes to the data structures in the preliminary version of the desktop client.

Additional Implementations:

Despite setbacks, we were able to implement myFaves in a way that closely mirrored our original design concept. Users were able to access our application smoothly and our dedicated server is able to handle multiple requests at once. MyFaves is fully accessible on mobile browsers and recovers gracefully from runtime errors.

Given more time to develop myFaves, there are several additions we would implement. We would develop dedicated mobile applications for Android, iOS and Windows Phone. Sharing music and playlists is an inherently social activity. Given this fact, it is logical to add either our own social media system or integrate myFaves with Facebook. If we were to implement our own social media system, each member would have a profile with editable personal information and they could chat, message each other and share files. Should myFaves become very popular, we may scale upwards if the current server cannot handle the load of requests.