The Creepers: Milestone Deliverable

Contents

[Document Purpose: 1](#_Toc289044014)

[The Problem: 1](#_Toc289044015)

[Our Implementation: 2](#_Toc289044016)

[Software Architecture 2](#_Toc289044017)

[Database Design 3](#_Toc289044018)

[Screen Mockups 4](#_Toc289044019)

[Expandability 5](#_Toc289044020)

# Document Purpose:

The purpose of this document is to outline what this project requires and how our team, The Creepers will attempt to do so over the following weeks. Our team is comprised of Joshua DeVinney from the databases class, Matthew Brannick, and T.J. Moats from the GUI class, and Bill Casey, Drew Jackson, and Devin Mullenix from both. This document is our plan, but due to implementation, time, or other constraints, we may deviate from this plan and our final changes will be reflected in our final design report.

# The Problem:

The goal of this project is to develop software that allows users to create playlists from a master list of songs and allows 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 updating, 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.

# Our Implementation:

## Software Architecture

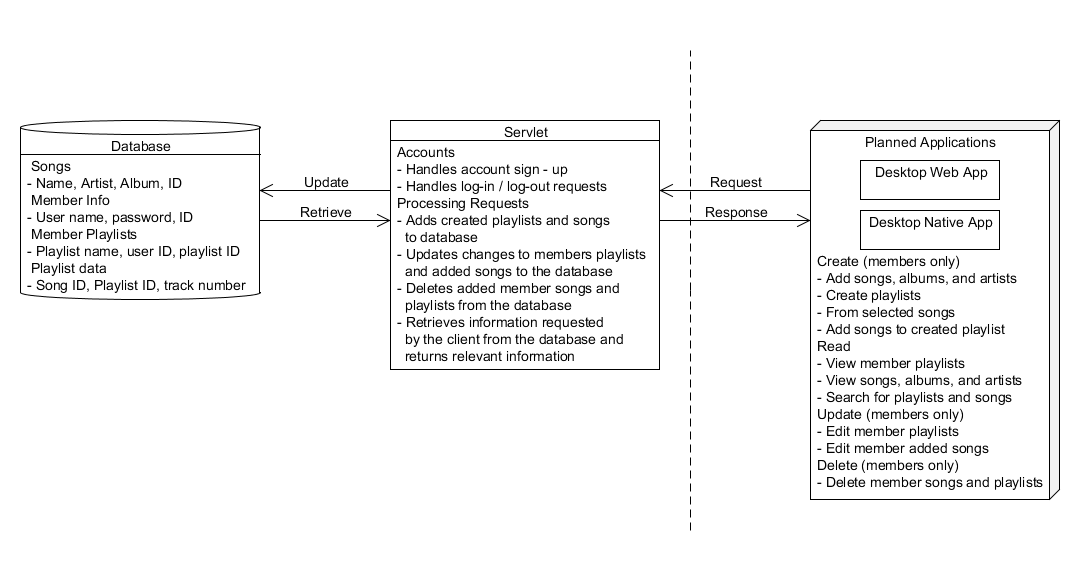


Figure – Software architecture diagram

Shown in Figure 1 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 app 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.

Handling the clients’ requests will be a single servlet. 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 in the next section.

## Database Design

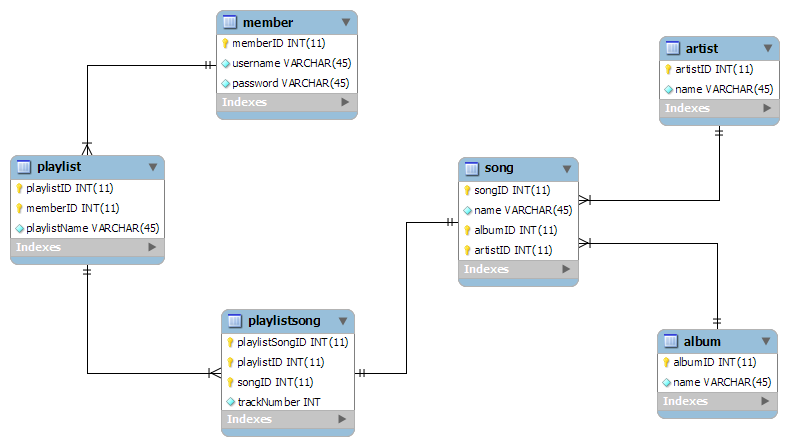


Figure - Database Design

Our current database plan is designed to contain all the needed information for the requirements of this project. It consists of three major parts, the members, the playlists, and the songs. The members are defined in a single table each with a unique ID and name and a set password. The playlists are defined by two tables; playlist, to define the name of the playlist, to which member it belongs, and its unique identifier, and playlistSong, identifying a song as part of a playlist and assigning it a trackNumber and unique ID. The songs are defined by three tables; song, giving a name and id to a song while referencing the tables album and artist which define the name of the songs’ album and artist respectively. The use of this system helps reduce data redundancy by using foreign key relationships . For example, rather than defining and naming an album in every entry of the song table, the song table refers to the id of an album defined in the table named album. This helps reduce the amount information required in the song table, makes it easier to keep the data consistent, and eases song searches based on albums as well since case sensitivity won’t be an issue. Likewise, rather than listing the songs in each playlist in a single entry in the playlist table, we used playlistSong to link individual songs to a playlist.

## Screen Mockups

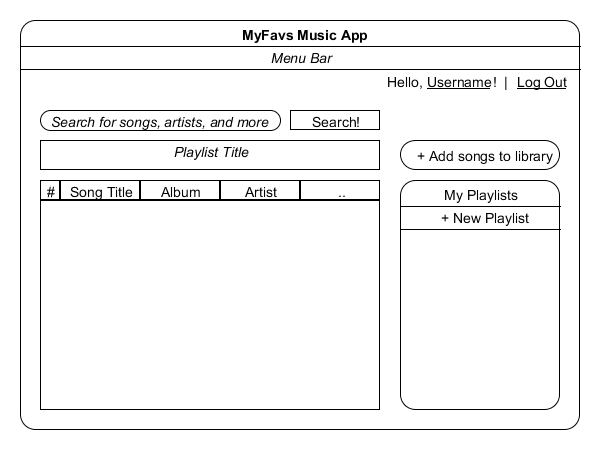


Figure - Application UI

Figure 3 depicts a prototype of our intended design for our desktop applications. 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.

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 will also be sortable based on such.

Any playlists the user creates will be listed under a menu on the right hand labeled “My Playlists”. It will also contain a button that allows a user to create a new playlists, and then select the songs from the ones listed in the table. Also implemented is the ability for a user to add to the list of known songs by clicking the “Add songs to library” button. Once added, songs will be instantly searchable by any user on the system.

## Expandability

As we were developing the basics of our software solution, a lot of ideas were presented that were not part of the basic requirements. While not all of these ideas may be implemented, we plan to add these features as a way of achieving the “Wow” factor. We plan on developing a third application for the mobile side, whether it be a native iOS/Android app or a mobile website app has yet to be decided. In addition to regular playlists, we want to implement private playlists that allows only the user who created it to view it. This may be expanded to include “friends” of that member. Also on the list included administrator privileges that would allow certain members to remove, update, and delete any song or playlist from the system. Lastly, we wish to implement a password-less log-in system that validates users based off of a different system that can be likened to a decoder ring.