

**CS 30700**

**Design Document**

**Team 18**

Andrew Furman

Nick Galecki

Kevin Matson

Josh Shepler

Doug York

Index

[**Purpose**](#_moojtoqi59oy) **3**

[Functional Requirements](#_22yfe1t0qvnd) 3

[Playlist Import](#_8hwf8yudd7qp) 3

[Playlist Export](#_xmlk38xf6r0h) 3

[Platform Access](#_6frui2vfo4fd) 4

[Playlist Management](#_owrstb4siqc) 4

[Interpersonal](#_r1y6iu4g2g2v) 4

[Platform](#_up01xedpvq1q) 4

[Other](#_32h9o87xvygv) 5

[Backend](#_uptlbf5va9fs) 5

[Non-Functional Requirements](#_lg74usogpk99) 5

[Architectural](#_el04il73t7be) 5

[Security](#_xwb9j5fn369g) 5

[Usability](#_w6b9vgfpaog2) 6

[Legality](#_gkokpfnyj6mv) 6

[**Design Outline**](#_4sjbu6f4f052) **7**

[High Level Overview](#_q1gzpw4xlfft) 7

[Webapp Client](#_j1gmiskknpv6) 7

[Android Client](#_zdxi4yud0v5g) 7

[Backend Server](#_83w8eeya6uvv) 7

[Database](#_xax6ler5zh9n) 8

[**Design Issues**](#_3ohm459bv0za) **9**

[Functional Issues](#_b4u47s2lsy5z) 9

[Non-Functional Issues](#_4kl503jsokxc) 11

[**Design Details**](#_ozgg1u9i0jga) **12**

[Data Class Level Design](#_jeq8n0mcnglx) 12

[Data Classes and Interactions](#_hsq3mcnxb5dl) 12

[User](#_by7t3w84sidx) 12

[Playlist](#_vfz0teihko3v) 13

[Song](#_ndnsca4y5dr8) 13

[Database](#_fwsev3iqucpn) 13

[Streaming Service](#_504tzzofg8od) 13

[Sequence Diagrams](#_5ttkzros657) 15

[Application Start and Login](#_fn7i7r8mcziw) 15

[Import Playlist](#_amqbslc29h6q) 16

[Merge Playlist](#_8ccj3tqw577e) 17

[Export Playlist](#_o7xiozga3s9t) 18

[Edit Imported Playlist](#_57ym07t76uyu) 19

[UI Mockups](#_5rsswfqv8az6) 20

[Login Page](#_r8bxzpw3okbp) 20

[Home Page](#_pjxjkp4ao91i) 21

[Transfer Page](#_v01idllwgae7) 22

[Merge Report](#_mvhxix7zchdr) 23

# 

# **Purpose**

With today’s music streaming services increasing in popularity, some users feel that once they start using one service, they’re stuck with it. They may feel that all of their playlists are stuck on their account, and the only way they can transfer playlists to another music streaming service would be by hand. We plan on eliminating this process of transferring playlists by hand, because this can take hours, depending on the size of the playlist.

InterLinked will allow users to import their playlist from one service, then export it directly to another. Users will also be provided with the functionality to combine, manage, and delete their playlists on the go. We believe that many users want to change music streaming services, but can’t get over the hurdle of losing their playlists or manually making them in the new service. InterLinked will be beneficial for consumers in the music streaming market, as it allows them to switch services easily if they’re dissatisfied with their current one.

## **Functional Requirements**

### Playlist Import

As a user:

* 1. I want to import a playlist from Amazon Music.
  2. I want to import a playlist from Apple Music/iTunes.
  3. I want to import a playlist from Spotify.
  4. I want to import a playlist from Youtube.
  5. I want to be notified when a playlist is imported successfully.

### Playlist Export

As a user:

* 1. I want to export a playlist to Amazon Music.
  2. I want to export a playlist to Apple Music/iTunes
  3. I want to export a playlist to Spotify.
  4. I want to export a playlist to Youtube.
  5. I want to be notified when a playlist is exported successfully.
  6. I want the transfer of playlists to be continued if it is interrupted.
  7. I want to be notified when a song can't be exported to a site.

### Platform Access

As a user:

* 1. I want to search for songs in the app.
  2. I want to open a searched song in the corresponding platform for listening.
  3. I want to choose which platforms I search from.
  4. I want to filter search results.
  5. I want to add songs from Amazon Music to my playlist.
  6. I want to add songs from Apple Music/iTunes to my playlist.
  7. I want to add songs from Spotify to my playlist.
  8. I want to add songs from Youtube to my playlist.
  9. I want to know which music streaming services are supported.

### Playlist Management

As a user:

* 1. I want to remove songs from my playlist.
  2. I want to save my playlists.
  3. I want to delete my playlists.
  4. I want to revert a playlist back to a state it was in the past.
  5. I want to organize my playlists by song title.
  6. I want to organize my playlists by album title.
  7. I want to organize my playlists by artist name.
  8. I want to organize my playlists by music genre.
  9. I want to merge two or more playlists into a new playlist.

### Interpersonal

As a user:

* 1. I want to share my playlist with a friend.
  2. I want to suggest songs to a friend.

### Platform

As a user:

* 1. I want the program to run on a computer.
  2. I want the program to run on an Android device.
  3. I want to use this service on a web browser.
  4. I want to have my playlists automatically synchronized between platforms.
  5. I want the app playlists to be transferable cross-platform.

### Other

As a user:

* 1. I want the app to listen to a song and put it into my playlist.
  2. I want the program to suggest songs to add to my playlists.
  3. I want the GUI to be intuitive and simple.

### Backend

As a developer:

* 1. I want to ensure security of user data.
  2. I want an API that allows for universal importing from sites without needing a separate method for each site.
  3. I want an API that allows for universal exporting to sites without needing a separate method for each site.
  4. I want the playlists to be independent from their source(s).

## Non-Functional Requirements

### Architectural

* The program backend is to be a Java-based program running on a device/server. The back end will be capable of handling at least 100 user queries simultaneously, each query being resolved ideally within 500 ms, but this depends on the music platform’s network speed and size of the query. This would allow for at least 17 million queries to be resolved per day.
* The frontend will depend on the platform developed on. On Android devices, the Android Studio Layout Editor will be used. On PC, the Java Swing and AWT libraries will be used to interface the back-end with the user.

### Security

* As a user, I want proper data security protocols to be followed to ensure the privacy of my user data from various music streaming services, and also that any data needing to be stored is protected by encryption. The data itself will be stored on the device with the application, therefore, the security is ultimately reliant on the user to not share the files with authentication data.

### Usability

* The user interface(s) are to be both intuitive and responsive, with simple menu layouts that can be easily used by the average music consumers. The interface will be similar across platforms to ensure that the user will have minimal confusion when using a different platform.

### Legality

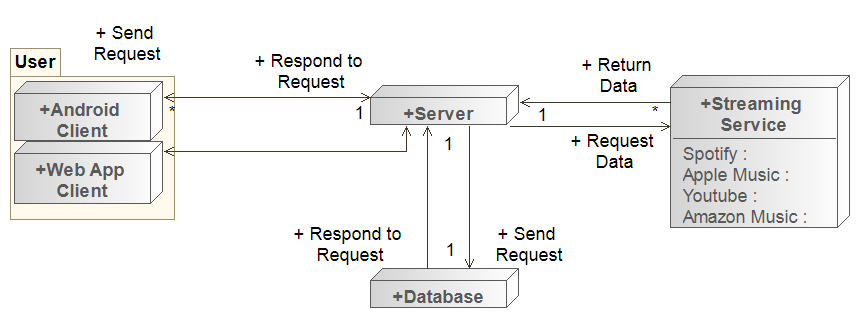
* The program will function within all current copyright law, especially in regards to music. As a result, playing music will only be accessible through third parties.

# 

# Design Outline

## High Level Overview

Our project allows users to import, export, and modify playlists from popular music streaming services such as spotify, apple music, and google play music. They can use our app from an android device, or from our webapp. They can access their playlists using an account they create on our website. Our service will use the client-server model, with a database to store user information such as playlists.



### Webapp Client

* The webapp will be one of the two ways the user interacts with our system.
* The webapp will have a similar structure and aesthetic to the android app.
* The webapp will be where the user can import, export, manage, and merge playlists.

### Android Client

* The android app will be one of the two ways the user interacts with our system.
* The android app will have a similar structure and aesthetic to the webapp.
* The android app will be where the user can import, export, manage, and merge playlists.

### 

### Backend Server

* The server will respond to requests made by each client.
* The server will house all of our security measures, ensuring that a user can’t get another’s data without permission.
* The server will query the database to fetch user information such as playlists.

### Database

* The database will respond to requests made by the backend server.
* The database will store user information used by our service.
* The database will store playlists owned by a user.
* The database will store songs on the playlist.

# 

# Design Issues

## Functional Issues

* Should User information be stored for streaming platforms?

**Option 1:** Yes

**Option 2:** No

**Choice:** Option 1 - Yes

**Explanation:** User login information should be stored between sessions. This ensures that login information does not need to be entered every time one wishes to use Interlinked. However, data will only be stored if its security can be ensured.

* How should conflicts with duplicate songs in a playlist be handled?

**Option 1:** Allow the user to keep duplicate songs in a playlist.

**Option 2:** Give the user an option to either keep duplicate songs in or to delete duplicate songs in a playlist.

**Option 3:** When merging playlists, never delete duplicates.

**Choice**: Option 2 - Give the user an option to either keep duplicate songs in or to delete duplicate songs in a playlist.

**Explanation:** We chose to notify a user after they have merged two playlists if there are duplicate songs in the newly merged playlist. This option makes our application more user friendly, giving users more options for playlist management.

* How will it be handled when a song isn’t found on the platform that is being exported to?

**Option 1:** Do nothing, and create the playlist without the missing song.

**Option 2:** Display an on-screen message notifying the user that the song cannot be located.

**Choice:** Option 2 - Display an on-screen message notifying the user that the song cannot be located.

**Explanation:** Our team wants to focus on creating a very user-friendly application that provides the user with many options and tools to use. A user should be aware of issues with a playlist so they know why exporting to a playlist may not be completely successful.

* How will playlists be exported to Youtube, where multiple instances of one song can exist?

**Option 1:** Only offer support for artists with official YouTube channels.

**Option 2:** Attempt a nearest-match playlist builder.

**Option 3:** Only offer import functionality for Youtube, not export.

**Option 4:** Don’t support Youtube on our platform.

**Choice:** Option 3 - Only offer import functionality for Youtube, not export.

**Explanation:** There are many good music playlists on Youtube, which users may wish to transfer to other streaming sites. Youtube being an open platform means that several uploads of the exact same song are likely to exist. It would be difficult to parse through all copies of a song and systematically determine the best version. Only allowing a user to import their playlists from Youtube solves this issue. Other choices either cut functionality entirely or were too complex for us to implement accurately. This decision allows us to retain some of the features we wish to support.

## Non-Functional Issues

* What database should be used?

**Option 1:** SQL Server

**Option 2:** Firebase

**Choice**: Option 1 - SQL Server

**Explanation:** We chose to use an SQL Server rather than Firebase because SQL Server has a Java library that we can use, whereas Firebase does not. Firebase also seems to have a larger learning curve than SQL Server and contains a lot more unnecessary features that our group would never utilize for this project.

* What language should the backend be developed in?

**Option 1:** C/C++

**Option 2:** Java

**Option 3:** Python

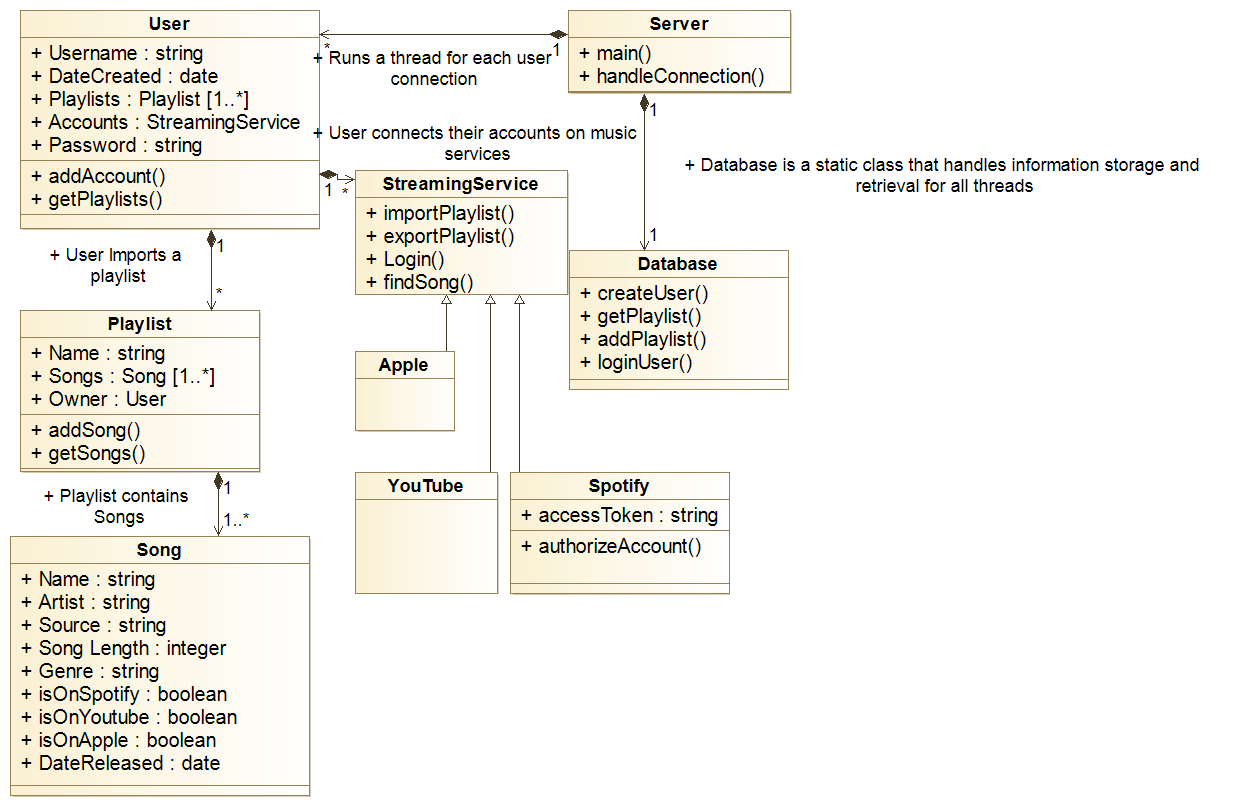
**Option 4:** NodeJS

**Choice**: Option 2 - Java

**Explanation:** Java is an object oriented language, which complements the structure of our program well. Our group is also the most comfortable with this programming language. Programming the backend in Java will also be beneficial for also making the app accessible on Android Platforms.

# Design Details

## Data Class Level Design



## Data Classes and Interactions

### User

* A potential user is required to create a User account before Interlinked can be used.
* Requires a login token e.g. email and password. This then becomes a secure unique key in the database.
* Has getter methods for this user’s playlist data.
* Stores a user’s playlist information
* Stored in the database. Used as a key in the database by the backend in order to find associated playlists that the user possesses.

### Playlist

* Contains a dynamically resizable array of songs that are in the playlist.
* Has methods for adding or removing songs from a playlist.
* Songs can be added individually and via importing from websites.
* Has capabilities of being merged with another playlist, while minimizing the number of duplicate songs.
* Stored on the database. Retrieved from the database by the backend using the User as a key.

### Song

* Stores relevant data, one instance per song.
* Stored on the database. Found as an element of a Playlist.
* Should have enough data that the individual instantiations of the song are able to be found on any platform that is supported and that would contain the song.
* Data is stored in a general manner so it can be formatted as needed by Streaming Service classes.
* Attributes populated via Streaming Service, which can be searched or imported from by the User.

### Database

* Houses API calls for communication between the backend and the database.
* The database holds User information as well as Playlists that the users own and the Songs contained within that Playlist.
* Ensures User security.
* Single static object that prevents concurrency issues from compromising correct database functionality.
* Allows for the User to retrieve previously created/imported/merged playlists that were instantiated on other platforms e.g. a user creates Playlist P on Android and can then can have access to P on the web application.

### Streaming Service

* Abstract class for handling connection to a music streaming service.
* This class is inherited by a specialized class for a specific site (Spotify, Youtube, etc.)
* Supports the importing and exporting of playlists, as well as relaying any errors that may have occurred.
* Supports locating potential Songs on the platform via querying. May require web crawling.
* These specialized classes handle interfacing with the service, through an API if applicable.
* If no API is available, one will be written in the specific site’s specialized class via webhooks or other similar protocols that are supported by the platform. This makes even site APIs abstract to the programmers.
* Spotify extension of this class stores an access token so that the user is only required to login once.

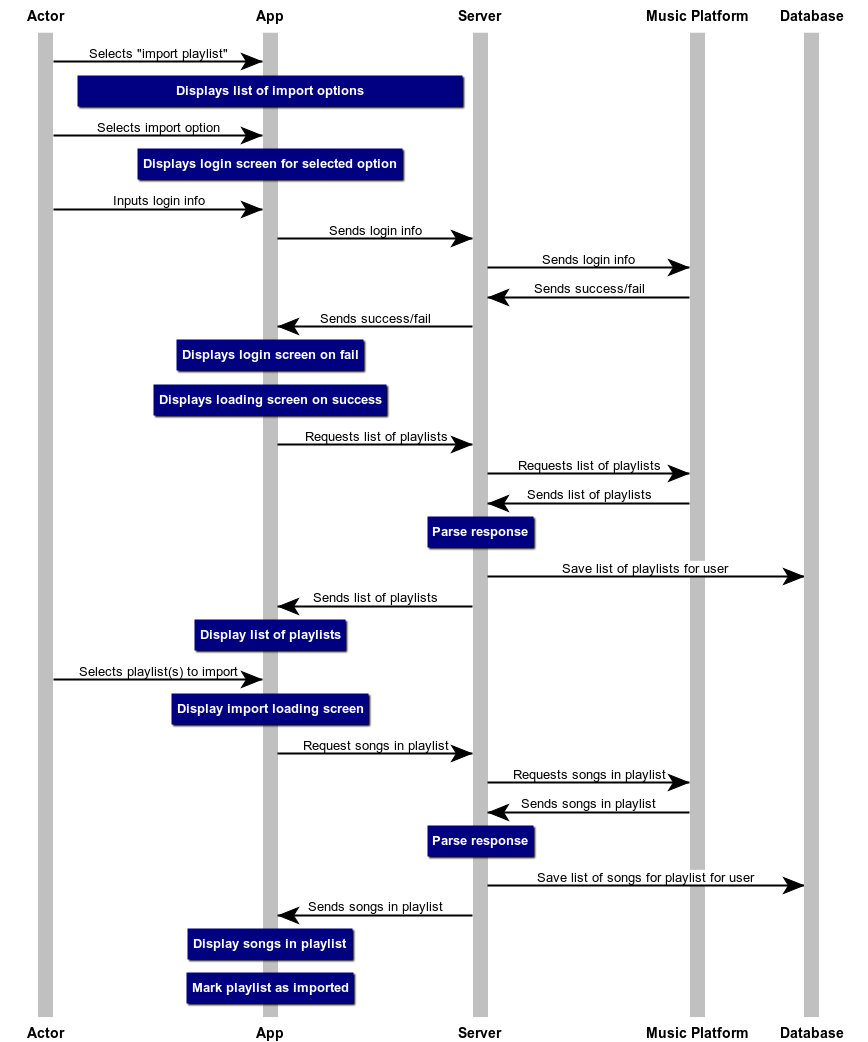
## Sequence Diagrams

### Application Start and Login

### 

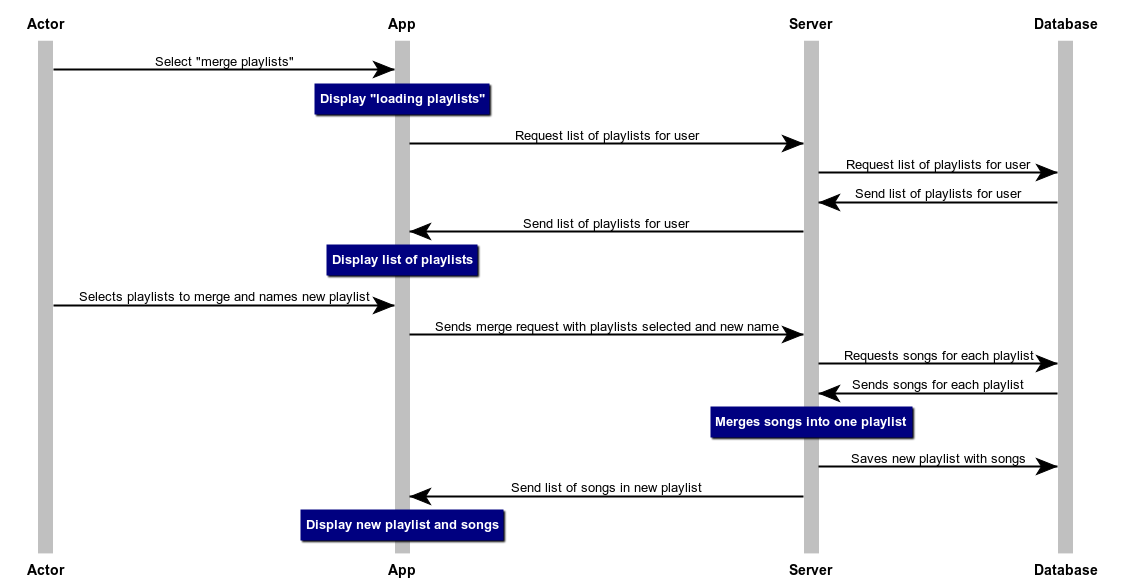
The above sequence diagram illustrates the process that occurs when a user first opens the app. Shown is the user login process and how the app receives the saved list of playlists from the database to be displayed on the main screen.

### Import Playlist



The above sequence diagram illustrates the process that occurs when a user chooses to import a playlist. Shown are the interactions between the user and the app, as well as how the app requests information from the server, database, and music platform.

### Merge Playlist



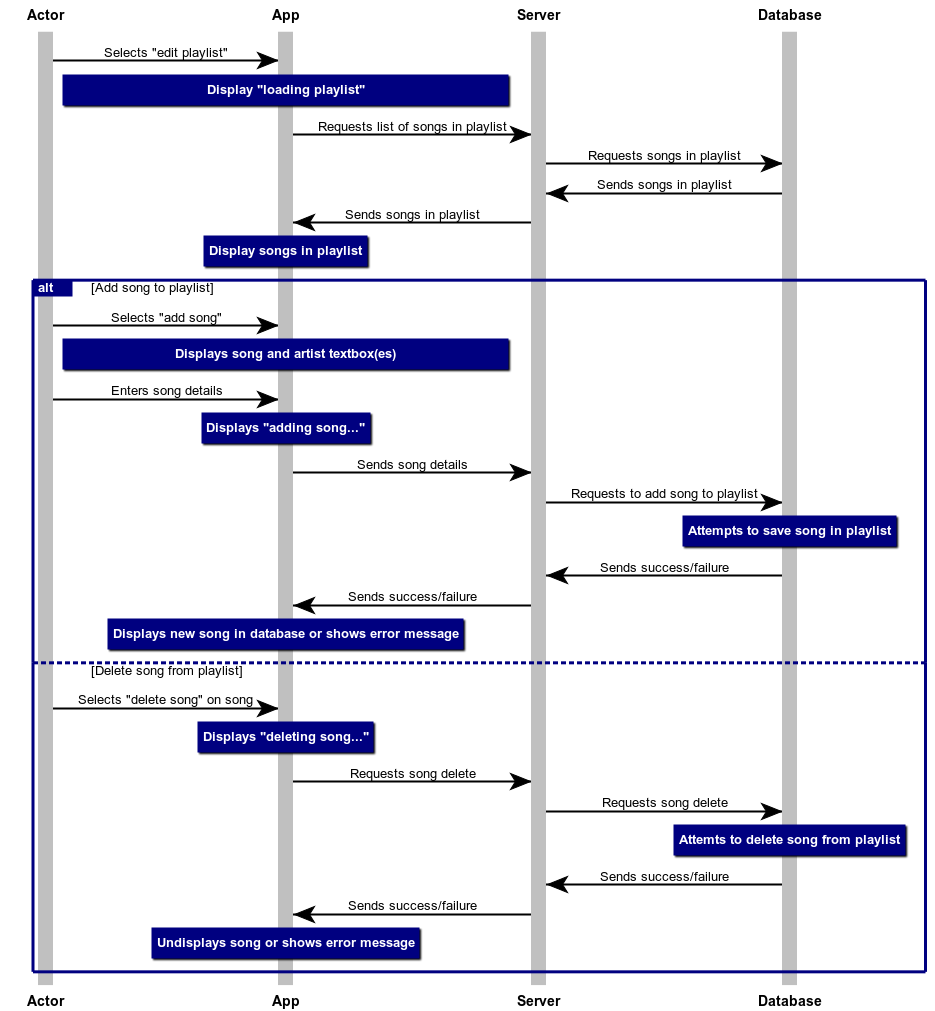
The above sequence diagram illustrates the process that occurs when a user decides to merge two playlists together, i.e., create a new playlist that contains all the songs from both playlists (with no duplicates). Shown are the interactions between the app and the user, as well as the server’s process and access to the database, which contains the necessary playlist data.

### Export Playlist

### 

The above sequence diagram illustrates the process that occurs when a user decides to export a playlist to a specific music platform. Shown are the interactions between the user and the app in order to communicate the user’s intent, along with the interactions between the processing done by the server, the data requested from the database, and the export requests sent to the selected music platform.

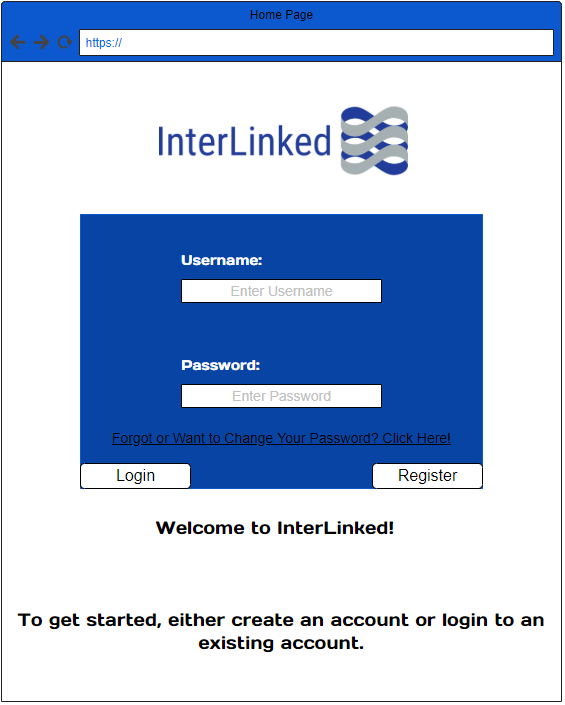
### Edit Imported Playlist



The above sequence diagram illustrates the process that occurs when a user decides to edit a playlist that has been imported into the app. Shown are the interactions between the user and the app in order to communicate which songs should be added or deleted, as well as the interactions between the server and the database to handle storage of the updated playlist data.

## UI Mockups

### Login Page



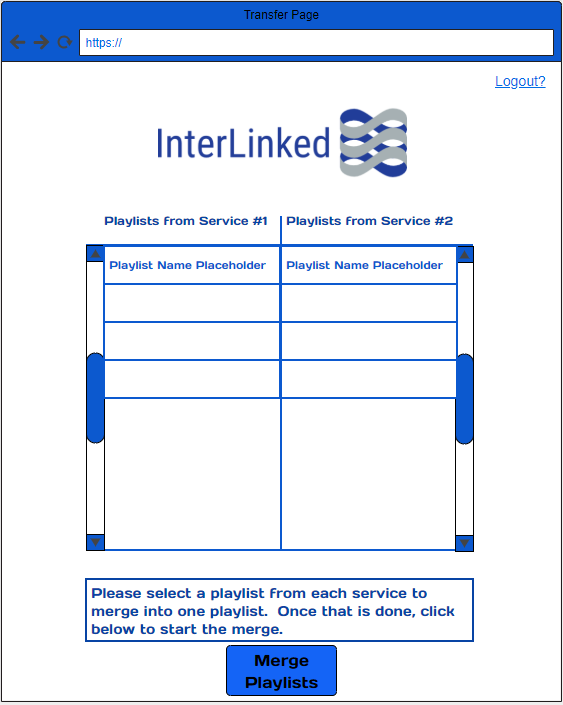
The login page provides access to the program for those who have registered accounts. Here, a user can easily create a new account by entering a username and password, and then clicking the “Register” button, or login to a preexisting account. There is also a link that will guide a user through the process of changing their password if they wish to do so.

### Home Page



After a user logs in they will be directed to this screen. On all pages after the login screen we include a logout link in the upper right-hand corner that allows the user to log out of their account and be moved back to the login page. On this page users are given the option to import a playlist from a music streaming service to their account, export a playlist to a particular music streaming service, or select two services that contain playlists that they want to combine.

### Transfer Page



If a user chooses to merge two playlists from different music streaming services then they will be sent to this page. It will display which services the user is choosing playlists from and will display the user’s playlists from each service in an organized table. They can then choose one playlist from each service and then click the “Merge Playlists” button to combine both playlists into a single playlist.

### Merge Report



This screen will display whether or not the merge was a success or not. They will also be able to view a list of all the songs in the now merged playlist. Next, they are able to choose which music streaming service they want to upload this playlist to. Once they select which service and click the “Transfer Playlist” button, they will be notified whether or not the transfer was a success. The two buttons on the bottom of the screen give the user access to start another merge process or to return to the main menu. The user still also has the ability to logout of the program using the logout link in the upper right-hand corner.