CSC 431

Spotify Playlist Finder & Social Media

System Architecture Specification (SAS)

Team 17

Allegra Papera	Project Management, Product Designer
Grant Yaniv	Full Stack Developer
Yasir Nemat	Full Stack Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	4/1/2021	Papera, A., Yaniv, G., Nemat, Y.	Filled out the system overview.
2.0	4/13/2021	Papera, A., Yaniv, G., Nemat, Y.	Fixed TA suggestions and added system, sequence, and class diagrams
3.0	5/4/2021	Papera, A., Yaniv, G., Nemat, Y.	Fixed formatting and changed various TA suggestions.

Table of Contents

Table of Figures

1 System Analysis

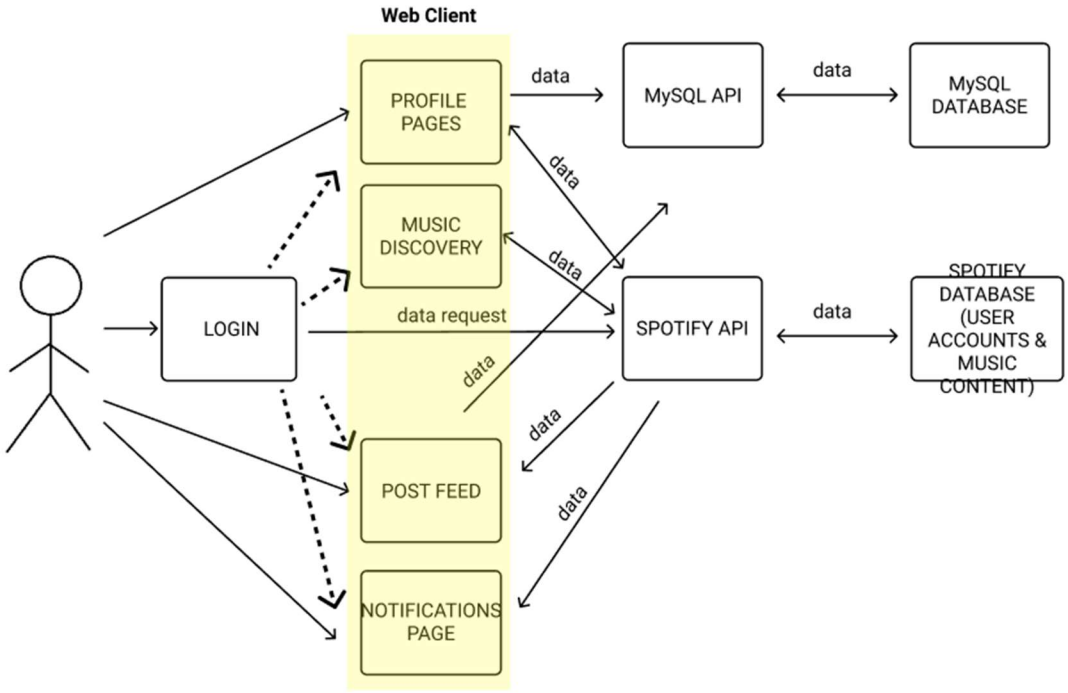
1.1 System Overview

The application behaves as follows: (1) The user logs into our application by logging in with their Spotify account credentials. This allows our application to have access to Spotify's database, as well as the individual's Spotify account data via the Spotify API. (2) The user fills out the Music Discovery Preferences form which our application will use to retrieve a sample of data from Spotify about which playlists fit the user's preference criteria. Our application will then randomly select certain playlists from this sample and automatically follow those playlists on the user's Spotify account. (3) The user will be notified under the Notifications Page about the newly followed playlists, as well as any activity such as comments on their posts. (4) The Music Social Media User Profile Pages and Social Media Post Feed will also be updated with information about the playlists that were followed by our app on the user's Spotify account. (5) Posts added to the Music Social Media Post Feed and User Profile Pages will be passed to our application's MySQL Database via a MySQL API.

The system will be compromised of the following functional requirements: Spotify Login, Music Discovery Preferences, Music Social Media User Profile Pages, Social Media Post Feed, and Notifications Page. The Spotify Login will allow our application to access user data from Spotify's database via the Spotify API. The Music Discovery Preferences feature will interact with Spotify's music database via the Spotify API to generate a sample of playlists retrieved from Spotify's database matching the user's preferences for music discovery. The Music Social Media User Profile Pages will also retrieve and display user profile data from Spotify's database via the Spotify API, in addition to our app's customizable profile features. The Social Media Post Feed and Notifications Page will not be directly connected to Spotify's database or its API, but rather will reflect our app's actions triggered by our Music Discovery Preferences feature (which does interact directly with the Spotify database via the Spotify API). Other in-app behaviors such as writing text posts, album reviews, and receiving likes on posts will also be reflected in these features of the application. In-app updates to the Social Media Post Feed and Music Social Media User Profile Pages will be passed to our application's MySQL Database via a MySQL API for storage.

1.2 System Diagram

Figure 1: Spotify Web Application System Diagram



1.3 Actor Identification

The human actors included in our Spotify Playlist Finder & Social Media platform is the user who must login with their Spotify account credential. Users will primarily interact with the application using their Spotify accounts to request new music to listen to via the Music Discovery Preferences Form and to make social media posts related to their music listening activity on Spotify.

The system will depend upon interaction with non-human actors, the Spotify API and the MySQL API to allow for interaction between the application and the Spotify User Account Database and our MySQL Database containing user post data related to our app's Social Media Feed aspect. The Spotify API will allow our application to update the Spotify User Account Database to create changes to represent the following of playlists on specific user accounts.

1.4 Design Rationale

1.4.1 Architectural Style

The application is designed using the Pipes and Filters architectural style. Pipes and Filters is a style in which the system has components called filters which perform transformations on data and process the input they receive, as well as pipes which serve as connectors for the stream of data being transformed. Specifically, filters will receive input about the user's Music Discovery Preferences, and it will transform that data into a selection of playlists that could match the user's preferences. A smaller random selection of those playlists that are not currently followed by the user will be chosen to be followed on the user's Spotify account. The Spotify API will serve as both a Pipe

and a Filter, as it will be used to inform Spotify's DB of the user's preferences, and it will also be used to sample the playlists that match those preferences. Our Pipes and Filters architectural will be built primarily using Node.js to handle back-end data, as well as JavaScript to display and interact with the back-end data.

1.4.2 Design Pattern(s)

- Based on this type of application and system we will be creating we believe that the Façade design pattern would be most applicable because it will give us the unified interface we are looking for on this project. The web client will act as the façade while the Spotify API, Spotify database, MySQL API, and MySQL database will function behind the scenes of this application.
- Due to the limited nature of our application, the Singleton design pattern might be applicable. We could have one instance of a class for the web client, and then have all the applications features fit into this one instance.
- The decorator design pattern could also be applicable for our application in that we have all the components built into the main component of the social media. Each separate component, such as the notification page, Music Discovery Preferences Page, and profile pages could be components that work the main central page for the web application.

1.4.3 Framework

We will be using the flagship and only Spotify API to be able to use the functionality of some Spotify features, such as their playlist catalog and user profiles, to lay the foundation of our application. Since this is such a robust API, we do not need to create another API for this application, we just need to take advantage the existing Spotify API's functionalities. Fortunately, it is completely free to use once they grant access to developers who sign up through their website. We will be able to make Spotify API calls using JavaScript code.

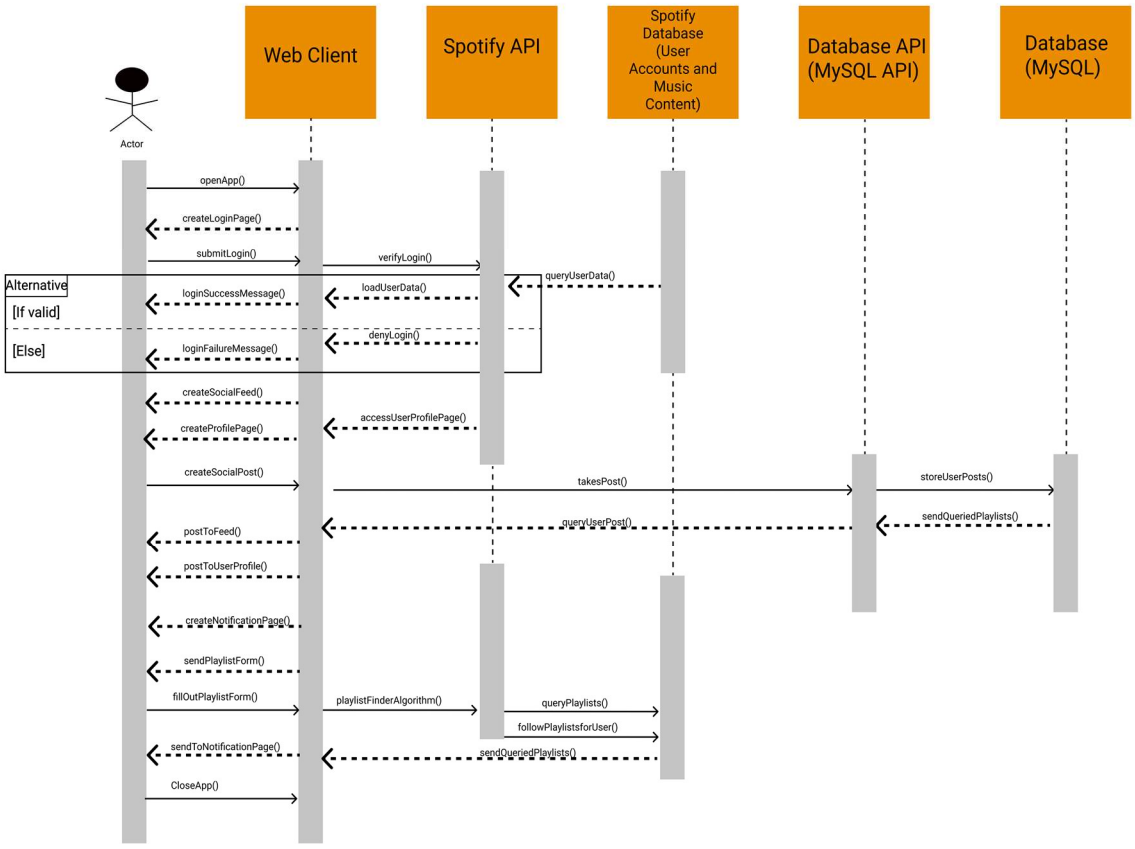
We will also be using the Node.js framework, as suggested by the Spotify API Developer Guidebook, to handle the back-end and server-side of this application since our front-end logic will be created with JavaScript. This is a great framework to use, not only because it is recommended by Spotify for use with Spotify API application, but because it is a well-maintained framework that has a large community surrounding it. We are not Node.js experts, but it should be an easy framework to learn considering there are many resources out there for it. This is a big factor into why we wanted to use Node.js for this application.

2 Functional Design

2.1 Sequence Diagram for General Use Case

This diagram shows the general sequence of usage for our application, outlining how the user begins using the application, what they can do with it, and how they terminate use.

Figure 2: Sequence Diagram for General Use Case of Application



- Once the app is opened, the login screen is displayed.
- The user submits login information, which is verified and loads user data if success, or denied and displays a failure message if it fails
- The profile page and social feed is then shown.
- The user may then create a social media post or fill out the playlist form.
- If the user creates a social media post, it is stored in the MySQL database.
- The post is then sent to the client and visible on the social media feed.
- Once a post is visible, a notification is sent to the user's notification page.
- A Playlist form is sent to the user to be filled out.
- Once the user fills out the playlist form, our algorithm sifts through the form and API music data to determine what types of music should be applicable.
- The Spotify database is queried to determine suitable databases according to the form data.
- The playlists are followed for the user and a notification is sent to the user's notification page.
- When the app is closed manually or unused for long enough, it is closed.

3 Structural Design

Figure 3: Spotify Application Class Diagram