Final Report

Project team name: 06 MusicGeneration

Team member: Nicholas Montoya and Yifan Li

Vision: Provide Spotify users with a way to quickly find and add new music to their libraries.

Description: Django based web application that searches Spotify's API for songs similar to a pre-existing playlist in the users library. Users are able to select and add their favorites from the generated playlists.

List the features that were implemented (table with ID and title)

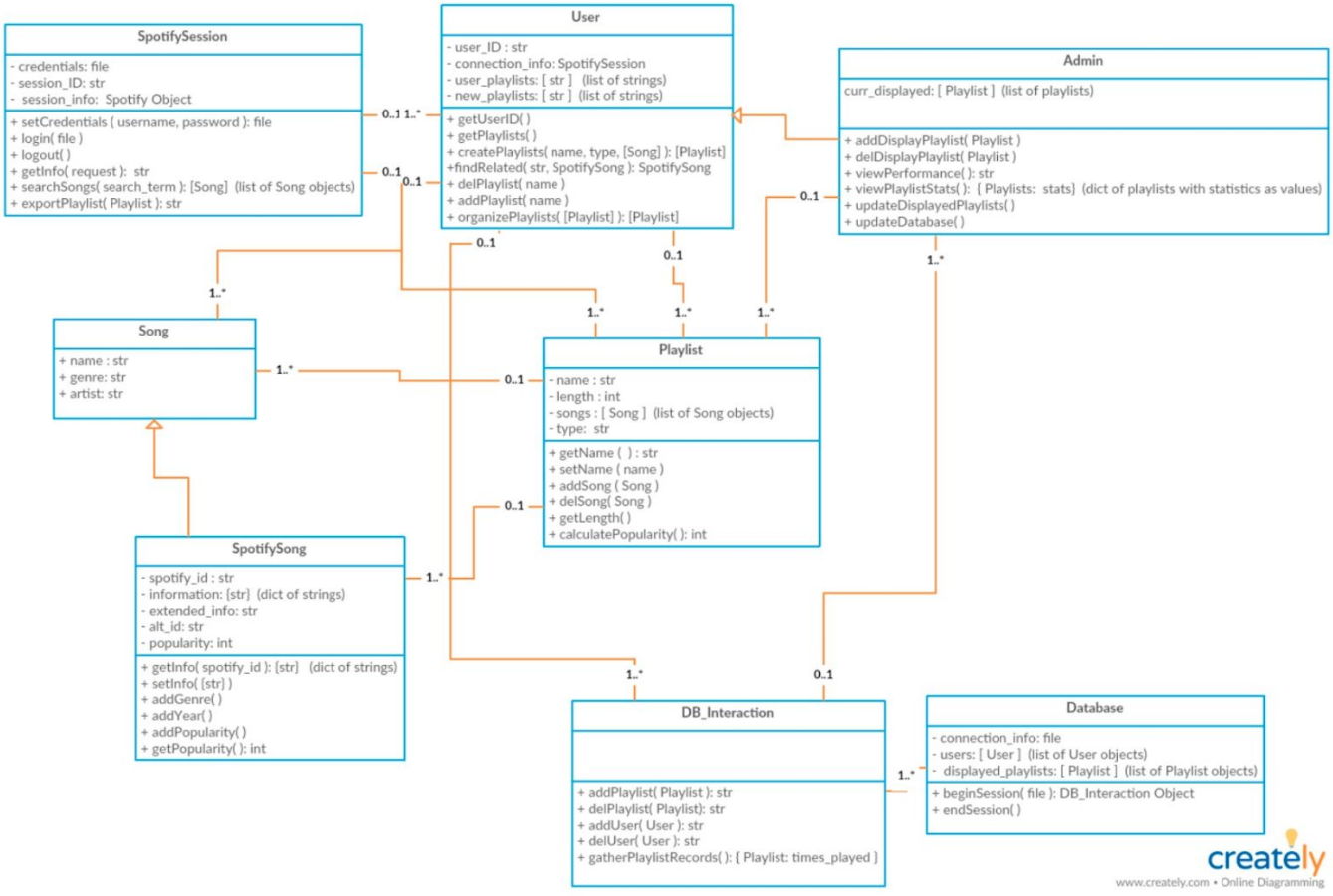
Implemented Features		
ID	Requirement Type	Features
UR - 02	User Requirement	Users are able to add auto-generated playlists to their Spotify account
UR - 03	User requirement	Auto-generated playlist can based on year
UR - 04	User Requirement	Users are able to find music based on genre
UR - 05	User Requirement	Admins are able to view backend performance metrics to view any failed requests or other issues
UR - 08	User Requirement	Admin can remove certain playlists

UR - 09	User Requirement	Admin can disable/remove harmful users
UR - 11	User requirement	Users are able to login into the system
FR - 01	Functional requirement	System allows people login with their Spotify account
NFR - 01	Non-Functional Requirement	Pages load quickly so users see pages in a short time. Using dynamic templating in Django to fill pages as they render.
BR - 01	Business Requirement	Users need to have a unique ID
BR - 03	Business Requirement	Users cannot edit or add to others playlists.
UC - 01	Add auto-generated playlists	Add auto-generated playlist into user’s spotify account
UC -02	List the popular playlists	Based on input, list most popular playlists

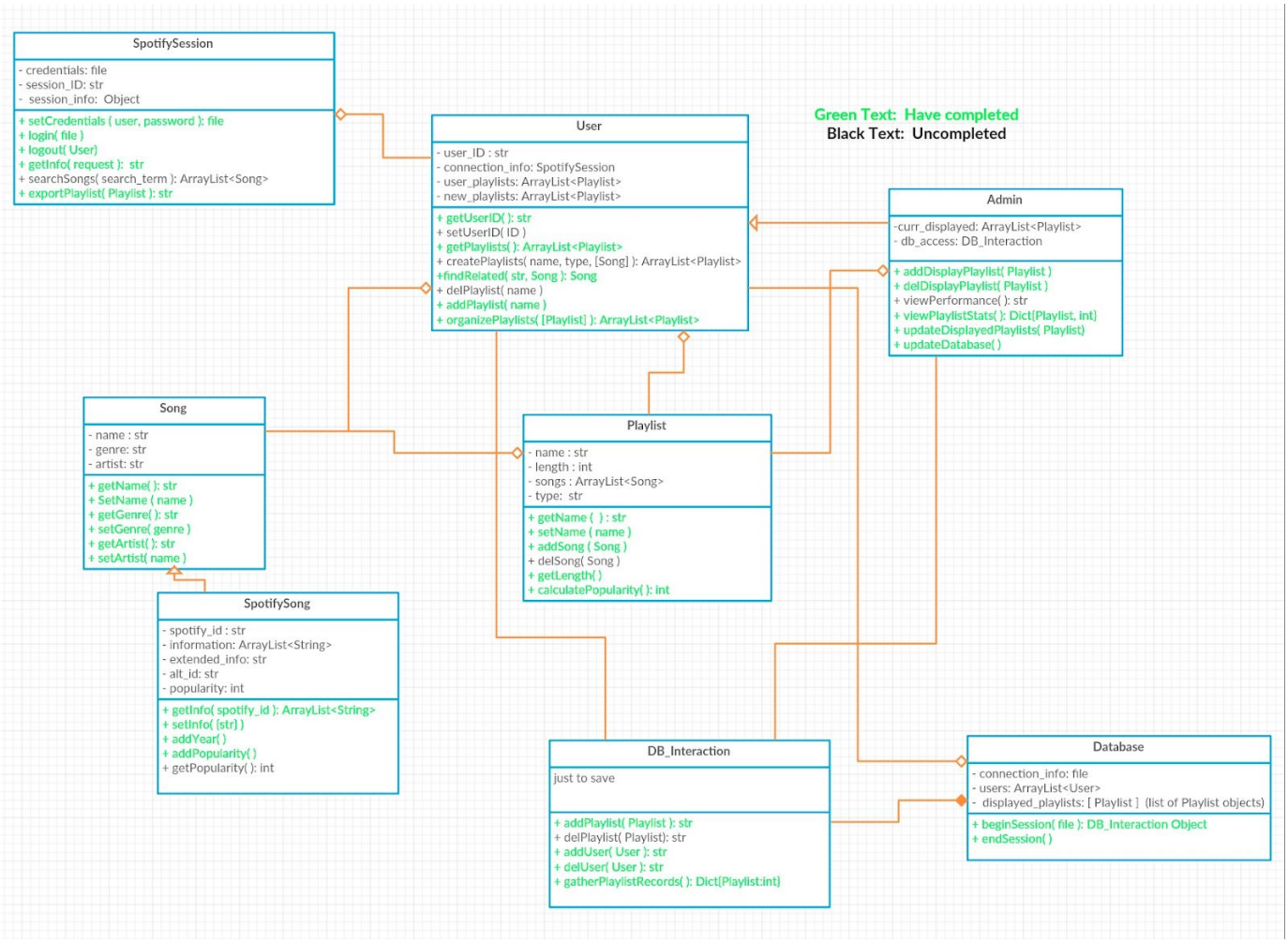
List the features were not implemented from Part 2 (table with ID and title)

Unimplemented Features		
ID	Requirement Type	Features
BR - 02	Business Requirement	Admin can only update 8 popular lists
UR - 01	User Requirement	Users need to be able to search for songs by name
UR - 06	User Requirement	Admin can update a ‘popular playlists’ page based on what users are listening to
UR - 07	User Requirement	Users can view the popular music playlists page
UR - 10	User Requirement	Users can sign up in our system. (ended up requiring users to have a spotify account already)
FR - 02	Function Requirement	System should notify users if they haven’t been active on their account
NFR - 02	Non-Functional Requirement	User account number and password should be stored securely. (account number is stored securely but no password ended

The class diagram from part two:



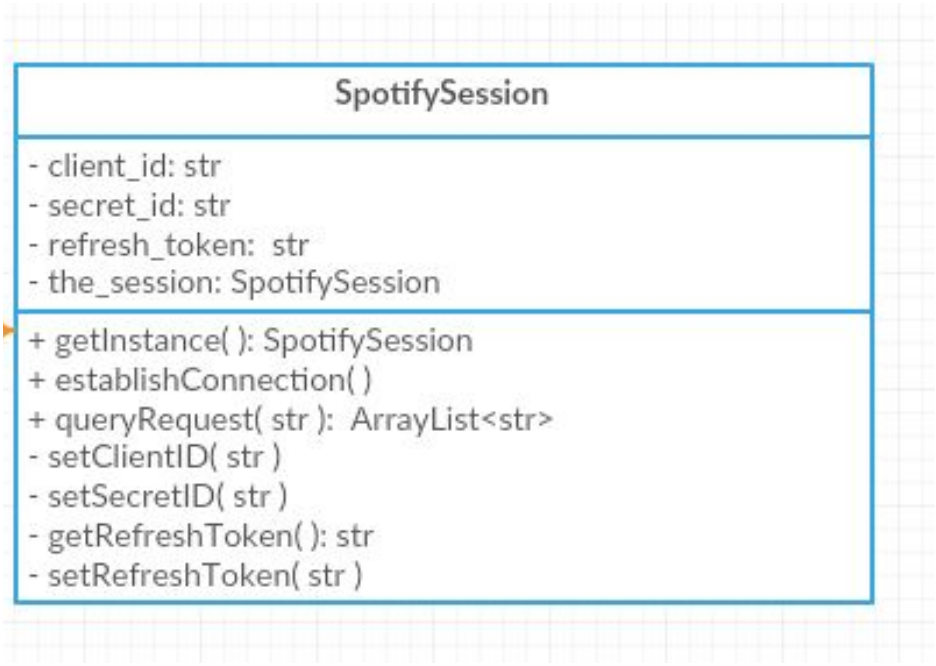
The final class diagram:



Discussion:

Our two class diagram don't change a lot. Setting up design first is a very efficient way for creating a project. When you are implementing functions for project, always look back to your class diagram that you created before will make your logic more clear. This will definitely improve the quality of your project. Saving time is a reason from another perspective. Additionally, front design will always allow developer to check what have been done and what parts are still missing. In general, this is a great step to move for project development.

Design Pattern: Singleton pattern



What we learned:

We believe that the most important thing from this class is learning a concept of how to maintain a code not just for now, but for the future. This can be achieved by using design patterns. We stepped through our project using singleton. After finishing this project, we know how important it is to having design part done before actually starts coding. This can help us a lot when we move forward on implementing functions. Once we have all of design of our project, we can always check which parts have already done, and which parts are still missing. Thus, this is a very high efficient way to start implementing a project.