# Spotify Playlist Generator

Team: **Evan Easton and Adam Hayes**

## <u>Final Project Statement</u>

All of our main project features from Project 5 were implemented in the end, however we did not implement our "stretch goal" features.

Implemented:

- Create account and login
- Validate Login and create account credentials
- Search for specific songs
- Generate playlist based on user chosen songs, and name playlist
- Display previously generated playlists for given user

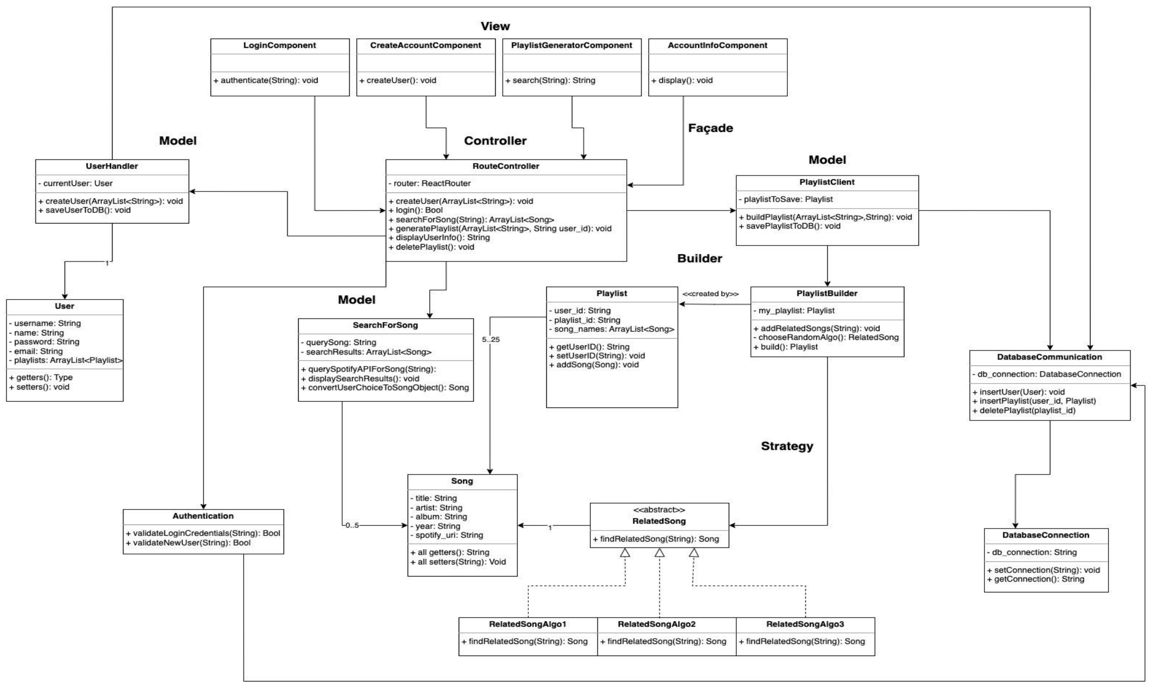
Not implemented:

- Allow user to link personal spotify account
- Allow user to save playlist to spotify account

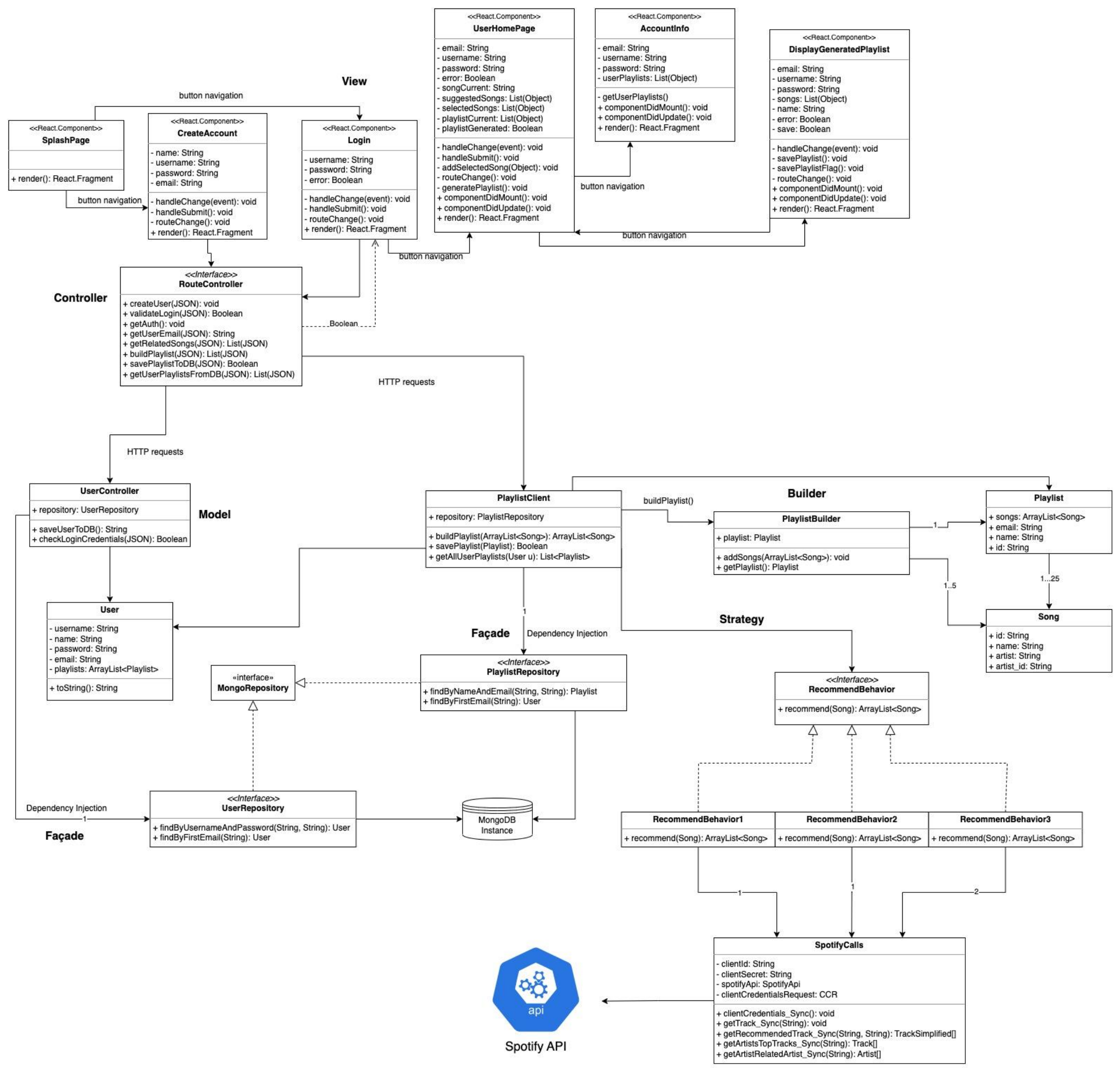
Overall, we are really happy with how much we were able to implement along with designing an appealing user interface. When we started we didn't exactly know how we'd set up the frontend and backend communication, but ended up just doing a simple RESTful API for the backend, and used HTTP requests to communicate back and forth. This seemed to work seamlessly, and allowed us to deploy our backend and frontend as separate entities.

## <u>UML Class Diagram</u>

**Project 5:**


View
LoginComponent
+ authenticate(String): void
CreateAccountComponent
+ createUser(): void
PlaylistGeneratorComponent
+ search(String): String
AccountInfoComponent
+ display(): void
Model
Controller
Façade
UserHandler
- currentUser: User
+ createUser(ArrayList<String>): void
+ saveUserToDB(): void
RouteController
- router: ReactRouter
+ createUser(ArrayList<String>): void
+ login(): Bool
+ searchForSong(String): ArrayList<Song>
+ generatePlaylist(ArrayList<String>, String user_id): void
+ displayUserInfo(): String
+ deletePlaylist(): void
Model
PlaylistClient
- playlistToSave: Playlist
+ buildPlaylist(ArrayList<String>,String): void
+ savePlaylistToDB(): void
1
User
- username: String
- name: String
- password: String
- email: String
- playlists: ArrayList<Playlist>
+ getters(): Type
+ setters(): void
Model
SearchForSong
- querySong: String
- searchResults: ArrayList<Song>
+ querySpotifyAPIForSong(String):
+ displaySearchResults(): void
+ convertUserChoiceToSongObject(): Song
Builder
Playlist
- user_id: String
- playlist_id: String
- song_names: ArrayList<Song>
+ getUserID(): String
+ setUserID(String): void
+ addSong(Song): void
<<created by>>
PlaylistBuilder
- my_playlist: Playlist
+ addRelatedSongs(String): void
- chooseRandomAlgo(): RelatedSong
+ build(): Playlist
5..25
DatabaseCommunication
- db_connection: DatabaseConnection
+ insertUser(User): void
+ insertPlaylist(user_id, Playlist)
+ deletePlaylist(playlist_id)
Strategy
Song
- title: String
- artist: String
- album: String
- year: String
- spotify_uri: String
+ all getters(): String
+ all setters(String): Void
0..5
1
<<abstract>>
RelatedSong
+ findRelatedSong(String): Song
Authentication
+ validateLoginCredentials(String): Bool
+ validateNewUser(String): Bool
DatabaseConnection
- db_connection: String
+ setConnection(String): void
+ getConnection(): String
RelatedSongAlgo1
+ findRelatedSong(String): Song
RelatedSongAlgo2
+ findRelatedSong(String): Song
RelatedSongAlgo3
+ findRelatedSong(String): Song

**Final:**


View
Controller
Model
Builder
Strategy
Façade
button navigation
<<React.Component>> SplashPage
+ render(): React.Fragment
<<React.Component>> CreateAccount
- name: String
- username: String
- password: String
- email: String
- handleChange(event): void
- handleSubmit(): void
- routeChange(): void
+ render(): React.Fragment
<<React.Component>> Login
- username: String
- password: String
- error: Boolean
- handleChange(event): void
- handleSubmit(): void
- routeChange(): void
+ render(): React.Fragment
<<React.Component>> UserHomePage
- email: String
- username: String
- password: String
- error: Boolean
- songCurrent: String
- suggestedSongs: List(Object)
- selectedSongs: List(Object)
- playlistCurrent: List(Object)
- playlistGenerated: Boolean
- handleChange(event): void
- handleSubmit(): void
- addSelectedSong(Object): void
- routeChange(): void
- generatePlaylist(): void
+ componentDidMount(): void
+ componentDidUpdate(): void
+ render(): React.Fragment
<<React.Component>> AccountInfo
- email: String
- username: String
- password: String
- userPlaylists: List(Object)
- getUserPlaylists()
+ componentDidMount(): void
+ componentDidUpdate(): void
+ render(): React.Fragment
<<React.Component>> DisplayGeneratedPlaylist
- email: String
- username: String
- password: String
- songs: List(Object)
- name: String
- error: Boolean
- save: Boolean
- handleChange(event): void
- savePlaylist(): void
- savePlaylistFlag(): void
- routeChange(): void
+ componentDidMount(): void
+ componentDidUpdate(): void
+ render(): React.Fragment
<<Interface>> RouteController
+ createUser(JSON): void
+ validateLogin(JSON): Boolean
+ getAuth(): void
+ getUserEmail(JSON): String
+ getRelatedSongs(JSON): List(JSON)
+ buildPlaylist(JSON): List(JSON)
+ savePlaylistToDB(JSON): Boolean
+ getUserPlaylistsFromDB(JSON): List(JSON)
Boolean
HTTP requests
UserController
+ repository: UserRepository
+ saveUserToDB(): String
+ checkLoginCredentials(JSON): Boolean
User
- username: String
- name: String
- password: String
- email: String
- playlists: ArrayList<Playlist>
+ toString(): String
PlaylistClient
+ repository: PlaylistRepository
+ buildPlaylist(ArrayList<Song>): ArrayList<Song>
+ savePlaylist(Playlist): Boolean
+ getAllUserPlaylists(User u): List<Playlist>
buildPlaylist()
PlaylistBuilder
+ playlist: Playlist
+ addSongs(ArrayList<Song>): void
+ getPlaylist(): Playlist
Playlist
+ songs: ArrayList<Song>
+ email: String
+ name: String
+ id: String
1
1...25
1...5
Song
+ id: String
+ name: String
+ artist: String
+ artist_id: String
Dependency Injection
«interface» MongoRepository
<<Interface>> PlaylistRepository
+ findByNameAndEmail(String, String): Playlist
+ findByFirstEmail(String): User
<<Interface>> UserRepository
+ findByUsernameAndPassword(String, String): User
+ findByFirstEmail(String): User
MongoDB Instance
<<Interface>> RecommendBehavior
+ recommend(Song): ArrayList<Song>
RecommendBehavior1
+ recommend(Song): ArrayList<Song>
RecommendBehavior2
+ recommend(Song): ArrayList<Song>
RecommendBehavior3
+ recommend(Song): ArrayList<Song>
2
SpotifyCalls
- clientId: String
- clientSecret: String
- spotifyApi: SpotifyApi
- clientCredentialsRequest: CCR
+ clientCredentials_Sync(): void
+ getTrack_Sync(String): void
+ getRecommendedTrack_Sync(String, String): TrackSimplified[]
+ getArtistsTopTracks_Sync(String): Track[]
+ getArtistRelatedArtist_Sync(String): Artist[]
api
Spotify API

**Comparison:**

The most notable change between our design from Project 5 and our final design is our mechanisms for communicating with the database and inclusion of our Spotify API communication routine. Instead of writing our own classes to communicate with MongoDB we leveraged the Spring Boot framework to simplify querying the database, and decoupled the communication from the User and Playlist clients. This was done by implementing the MongoRepository interface for our 2 database collections: Users and Playlists. We also decided to execute our Strategy pattern (for finding related songs based on user chosen songs) inside the Playlist client, instead of in the PlaylistBuilder itself. We also limited the backend access points which communicate with the RouteController (Controller in MVC) to 2 classes: UserController and PlaylistClient. Another big design difference that emerged during development was giving Playlist their own object and collection in the database, and we simply tied a user's email address to the playlist in order to identify all of a user's playlists that they have generated. In terms of patterns, we no longer used Façade for our RouteController, but instead used a Façade pattern for querying the database with UserRepository and PlaylistRepository.

## Third Party vs. Original Code

Backend framework: Spring Boot
Frontend framework: React
Tools: Material UI, Axios, Gradle, MongoDB, Spotify API
RouteController.js authentication function:
https://ritvikbiswas.medium.com/connecting-to-the-spotify-api-using-node-js-and-axios-client-credentials-flow-c769e2bee818
https://gist.github.com/donstefani/70ef1069d4eab7f2339359526563aab2

RoutingWrapper.js:
https://stackoverflow.com/questions/70143135/how-to-use-react-router-dom-v6-navigate-in-class-component

Spotify Calls Wrapper
https://github.com/spotify-web-api-java/spotify-web-api-java

## OOAD Process

1) MVC pattern: Creating an OO controller to handle communication between frontend and backend. We tried our best to program to the RouteController.js interface on the frontend, which would then handle all communication with backend models and update the state in the view. This was a difficult hurdle during development, but ultimately was the best design decision we could have made.
2) Strategy pattern: Integrating the Strategy pattern for choosing related songs into the Builder pattern for constructing playlists. It was difficult to visualize during the design phase how we would set up Builder and Strategy would work together, and ultimately ended up with a design which decoupled the Strategy implementation from the PlaylistBuilder.
3) The best OO-related experience we had with this project was learning how to leverage the Spring Boot framework to handle creating User and Playlist objects and saving them to and/or querying them from the database. Once we figured out how to use Dependency Injection and specifically the MongoRepository interface, we were able to seamlessly decouple the database communication from our backend business logic.