Project 2

▼ Feature 1: Better user management

This feature required new users to be able to register themselves without logging into an admin account and creating a new user manually. Hence, we created a separate page for registration and linked it on the `Login` page.

The two main files created for this feature are `Register.html` and `register.js`. The former file contains the frontend code to display fields like `username`, `email`, and `password` while the latter file has the controller to call the APIs.

The following lines were commented out in `UserResource.java`

```
if (!authenticate()) {
    throw new ForbiddenClientException();
}

checkPrivilege(Privilege.ADMIN);
```

The lines check if the user that is trying to register a new user has Admin privileges. Since, we want to allow anyone to register, we remove these checks.

▼ Feature 2: Better library management

Implementation Details

We edited the `Playlist` class to include another parameter `isPublic` taking a `Boolean` value to implement this feature. Due to this, we had to make changes in the SQL files to change the table structure. And finally, changes were made in the front-end to add the functionality of making playlists public and private.

Base Code (Backend)

- Changes were made to `PlaylistDto.java`, `PlaylistCriteria.java` and `Playlist.java` to include the parameter `isPublic` in the class structure.
- `PlaylistMapper.java` was edited to map the `ispublic` parameter from the SQL query result to `isPublic` parameter of the `PlaylistDto` class object parameter.

- Changes were made to `PlaylistDao.java` to change the SQL `SELECT` query, `CREATE` query and the `UPDATE` query to support the table with `ISPUBLIC` column.
- In `PlaylistResource.java`, new API calls were defined to return public playlists, toggle the public and private status, ignore authentication to view public playlists and integrate `isPublic` parameter in playlists.
- We edited `UserResource.java` to return the user id when fetching user data.

Database

We added the following statement to `dbupdate-001-1.sql` to update the structure of the table storing the playlists to add a new boolean column that tells whether the playlist is public.

```
alter table
  T_PLAYLIST
add
  ISPUBLIC NUMBER(1);
```

Frontend

- In `NamedPlaylist.js` we added another array to the root scope named `publicPlaylists` to store the data of all the public playlists.
- In `Playlists.js` in `app/controller` we also store the current user data in the scope, which we get from the API calls defined in `UserResource.java`.
- In `playlist.html` all of our changes are reflected.
  - We add a button to make playlists public and private.
  - We disable buttons to edit playlists when one user accesses another user's public playlist.
- In `main.html` we display all the public playlists below the user's playlist.

▼ Feature 3: Online Integration

In this feature, we needed to add Spotify's and Last.fm's `search` and `recommendation` integrations. To start off, we went through their API documentation

and figured out the APIs we needed to call for these. The APIs we have used are linked below:

1. `Spotify`
    a. Search: Search for Item
    b. Recommendation: Get Recommendations
2. `Last.fm`
    a. Search: Track.search
    b. Recommendation: Track.getSimilar

All of the four APIs require OAuth or API Key. In case of Last.fm, we used the API key provided in the repository itself while in for Spotify, the user has to provide the Bearer Token.

These APIs were called through Angular's `$http` service in `Search.js` and `Playlist.js` and appropriate changes were made in `search.html` and `playlist.html` to display the buttons and the responses clearly on the frontend.

The implementation code for the search APIs is given below:

1. Spotify

```
$scope.searchSpotify = function () {
    console.log(spotify_api_site);
    var heads = {authorization: "Bearer Token"};
    $http.get(spotify_api_site, {headers: heads}).then(
        function (data) {
            console.log(data.data.tracks.items);
            $scope.tracks = data.data.tracks.items;
        },
        function (error) {
            console.log(error);
        }
     );
 };
```

2. Last.fm

```
$scope.searchLastFM = function () {
    console.log(lastfm_api_site);
    $http.get(lastfm_api_site).then(
    function (data) {
       console.log(data.data.results.trackmatches.track);
```

```
        $scope.tracks = data.data.results.trackmatches.track;
    },
    function (error) {
        console.log(error);
    }
  );
};
```

A similar approach for recommendation APIs is also used.