Software Engineering Project-2

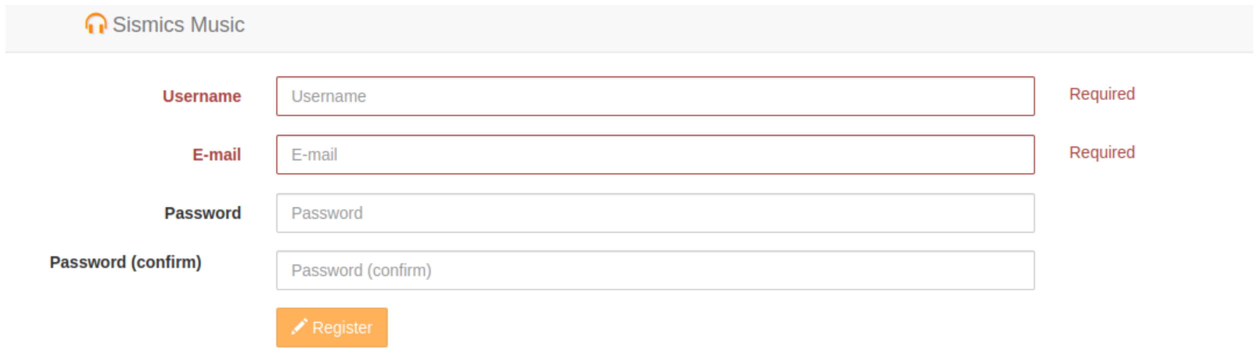
Team-12

Gangam Supreeth Reddy - 2020101021
Polavarapu Neeraj - 2020101026
Pothugunta Venkat- 2020101076
Nitin Rajasekar - 2020101117
Pusapati Romaharshan - 2020101069

Feature 1: Better user management

- **Added a new page ("Register.html")**
  This Feature can be used for the user to self register in the previous version only admin has the privileges to add the user and now we have changed it and added self registration form

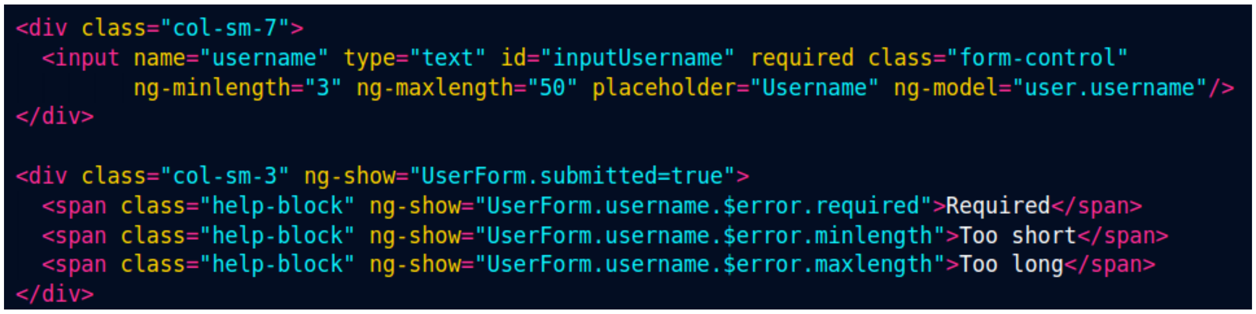
  The Registration Form UI is



  To make this Form we have created a new file register.html which contains the form html code along with attributes and we have also created a new appropriate Register.js controller where we will be doing a put request to the userresource.java file and we have also removed authentication in the register function of user because in the previous version the privileges of creating user has been only given to admin but now any user must be able to register.

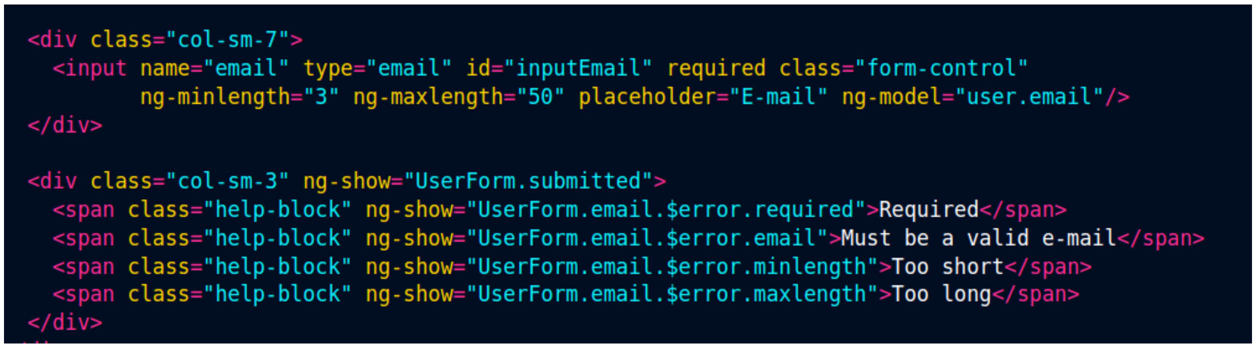
-The modified files are

register.html (Template File)



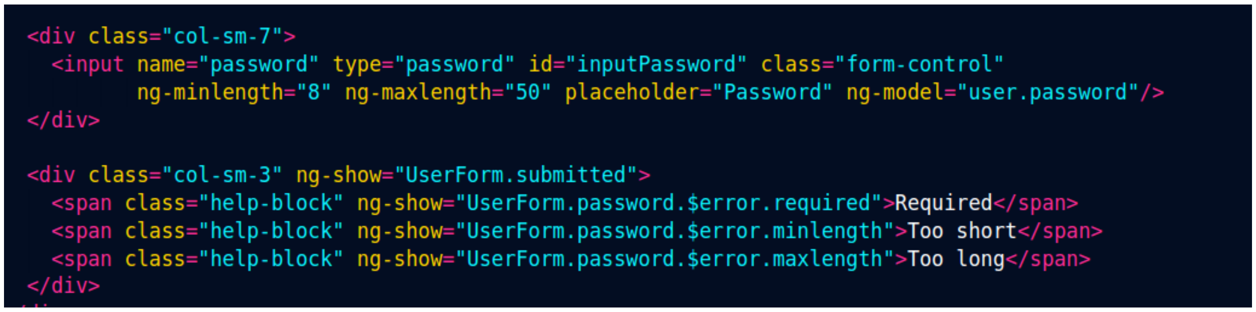
```html
<div class="col-sm-7">
  <input name="username" type="text" id="inputUsername" required class="form-control"
         ng-minlength="3" ng-maxlength="50" placeholder="Username" ng-model="user.username"/>
</div>

<div class="col-sm-3" ng-show="UserForm.submitted=true">
  <span class="help-block" ng-show="UserForm.username.$error.required">Required</span>
  <span class="help-block" ng-show="UserForm.username.$error.minlength">Too short</span>
  <span class="help-block" ng-show="UserForm.username.$error.maxlength">Too long</span>
</div>
```



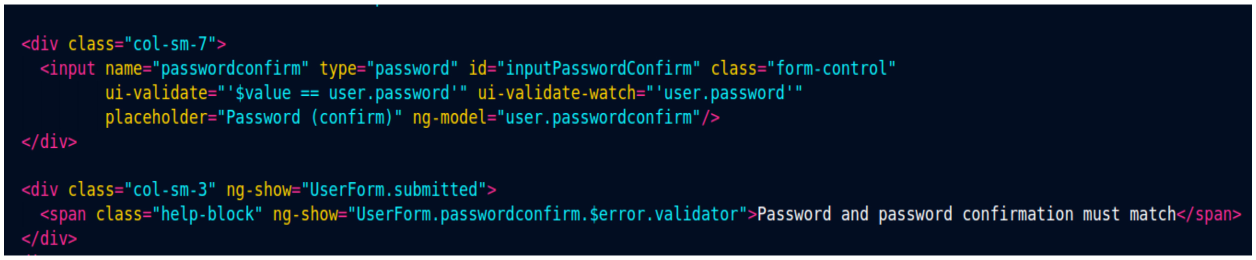
```html
<div class="col-sm-7">
  <input name="email" type="email" id="inputEmail" required class="form-control"
         ng-minlength="3" ng-maxlength="50" placeholder="E-mail" ng-model="user.email"/>
</div>

<div class="col-sm-3" ng-show="UserForm.submitted">
  <span class="help-block" ng-show="UserForm.email.$error.required">Required</span>
  <span class="help-block" ng-show="UserForm.email.$error.email">Must be a valid e-mail</span>
  <span class="help-block" ng-show="UserForm.email.$error.minlength">Too short</span>
  <span class="help-block" ng-show="UserForm.email.$error.maxlength">Too long</span>
</div>
```



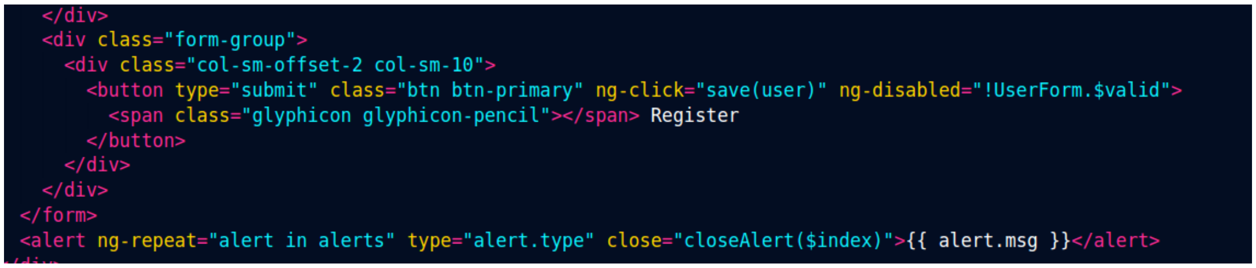
```html
<div class="col-sm-7">
  <input name="password" type="password" id="inputPassword" class="form-control"
         ng-minlength="8" ng-maxlength="50" placeholder="Password" ng-model="user.password"/>
</div>

<div class="col-sm-3" ng-show="UserForm.submitted">
  <span class="help-block" ng-show="UserForm.password.$error.required">Required</span>
  <span class="help-block" ng-show="UserForm.password.$error.minlength">Too short</span>
  <span class="help-block" ng-show="UserForm.password.$error.maxlength">Too long</span>
</div>
```



```html
<div class="col-sm-7">
  <input name="passwordconfirm" type="password" id="inputPasswordConfirm" class="form-control"
         ui-validate="'$value == user.password'" ui-validate-watch="'user.password'"
         placeholder="Password (confirm)" ng-model="user.passwordconfirm"/>
</div>

<div class="col-sm-3" ng-show="UserForm.submitted">
  <span class="help-block" ng-show="UserForm.passwordconfirm.$error.validator">Password and password confirmation must match</span>
</div>
```



```
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-primary" ng-click="save(user)" ng-disabled="!UserForm.$valid">
        <span class="glyphicon glyphicon-pencil"></span> Register
      </button>
    </div>
  </div>
</form>
<alert ng-repeat="alert in alerts" type="alert.type" close="closeAlert($index)">{{ alert.msg }}</alert>
```

Register.js (Controller)

```
'use strict';

/**
* Settings user edition page controller.
*/
angular.module('music').controller('Register', function($scope, $dialog, $state, $stateParams,
Restangular) {
 /**
  * Saves the current user.
  */
 $scope.save = function(user) {
   console.log(user);
   var promise = null;

   promise = Restangular
   .one('user')
   .put(user);

   promise.then(function() {
     $state.transitionTo('login');
   });
 };
});
```

App.js

```
.state('register', {
        url: '/register',
        views: {
```

```
      'page': {
        templateUrl: 'partial/register.html',
        controller: 'Register'
      }
    }
  })
```

UserResource.java

This authentication condition has been removed because every user must be able to register in the application.

Index.html

```
<script src="app/controller/Register.js" type="text/javascript"></script>
```

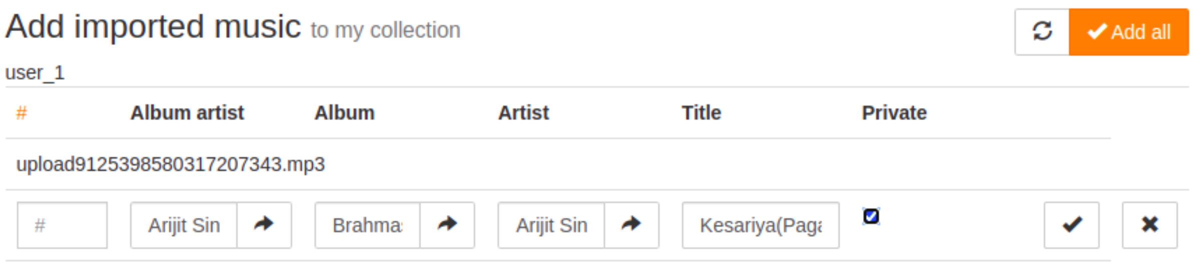
This line has been added in index.html inorder to import the Register.js controller

<u>Feature 2: Better library management</u>

As part of the task, we enable the user to have the ability to choose whether or not they wish their saved music to be publicly visible to other users.

<u>OPTION FOR USER TO ADD MUSIC(TRACK) AS PUBLIC OR PRIVATE:</u>
The user now has the option to make the trakcs private by clicking the checkbox in the below image. A basic checkbox has been added as part of the user interface to serve this purpose. If the box is checked, then the song added will not be visible to any user apart from the primary user themself.

This can be illustrated in the below example.
The below track has been set to private.

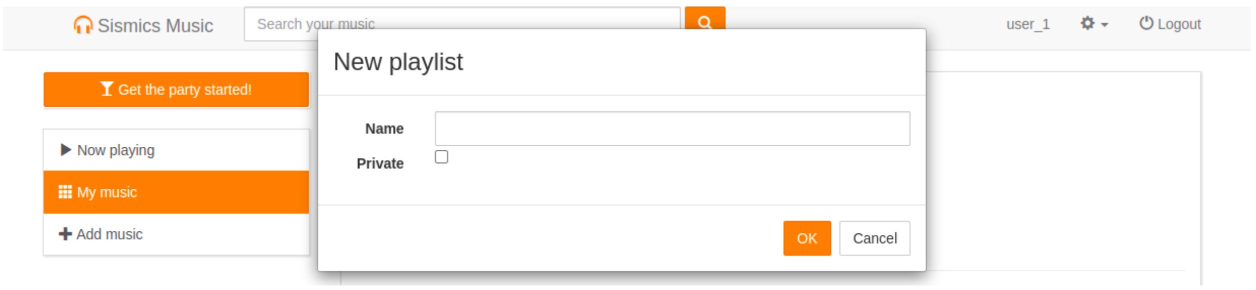

Add imported music to my collection

Add all

user_1

#	Album artist	Album	Artist	Title	Private
upload9125398580317207343.mp3					
#	Arijit Sin	Brahma	Arijit Sin	Kesariya(Paga	✓

OPTION FOR USER TO ADD PLAYLIST AS PUBLIC OR PRIVATE:



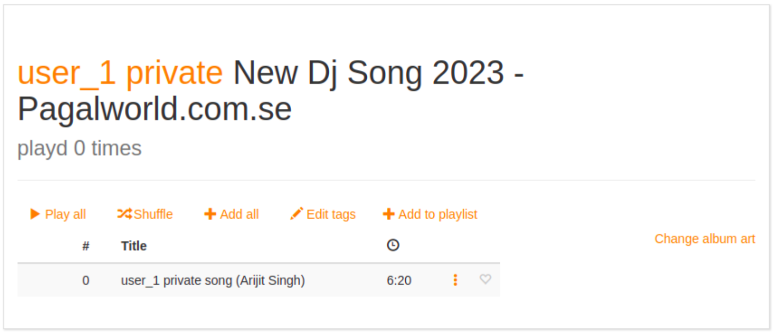
New playlist

Name

Private ☐

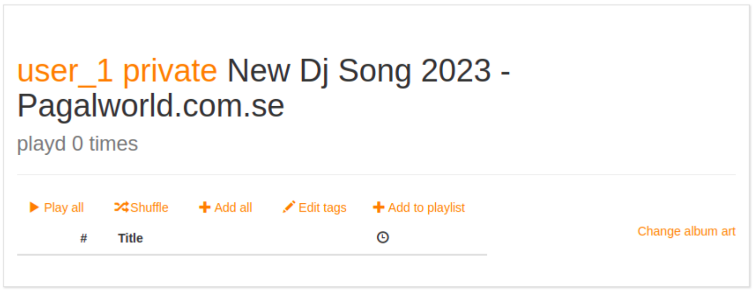
OK Cancel

- When a song is made private by user_1 then it is only present in tracks of user_1 as you can see in image 1, but you can't see this private track in user_2 in second image.


user_1 private New Dj Song 2023 - Pagalworld.com.se

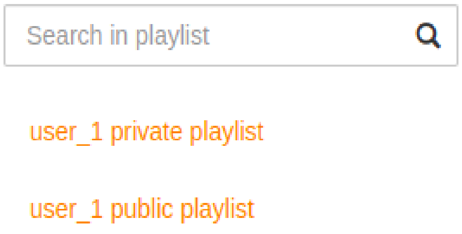
playd 0 times

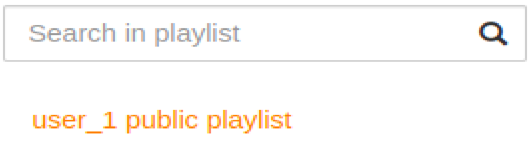
#	Title	⏱
0	user_1 private song (Arijit Singh)	6:20


user_1 private New Dj Song 2023 - Pagalworld.com.se

playd 0 times

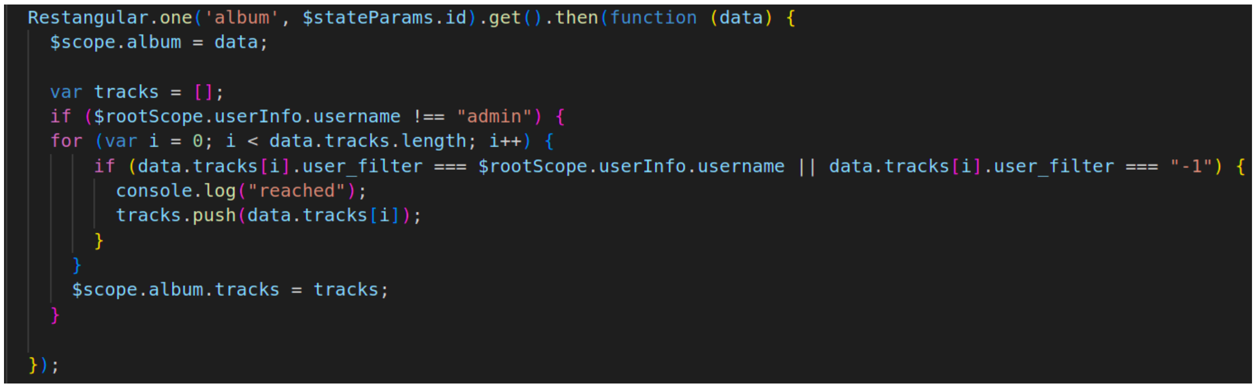
- The first playlists are created by user_1 one is private and another is public and in the second image you can see playlists for user_2 - only the public playlist created by user_1 and not the private playlist created by user_1.





- For admin we gave him access to see every track and playlist created by all the users.

The actual filter process is being applied over here. As you can see, we are creating a variable which we store in the backend database, in that variable it has the username if it is private or -1 if it is public. So we filter private/public accordingly in the following function.

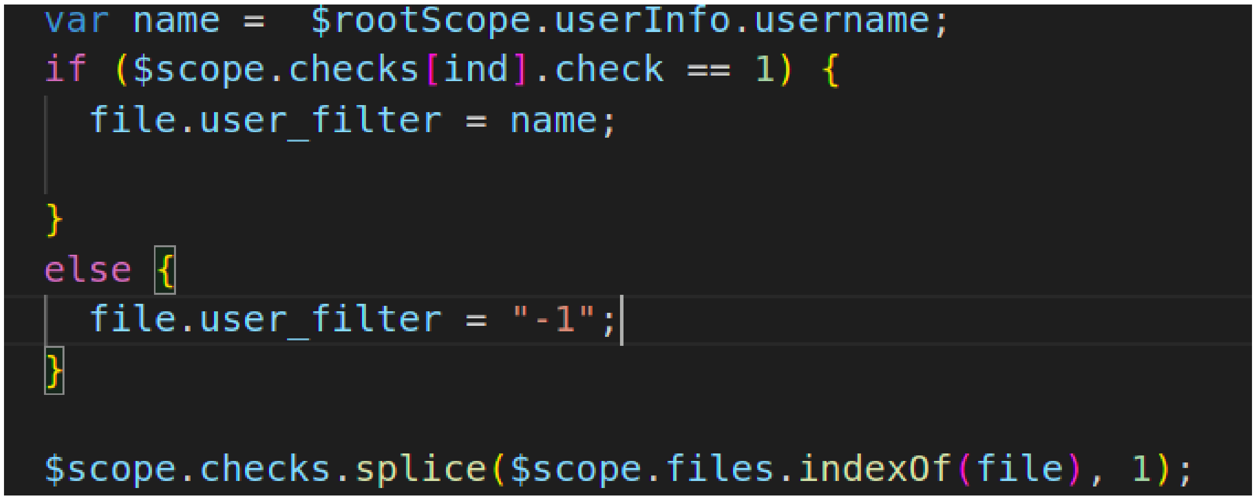


```
Restangular.one('album', $stateParams.id).get().then(function (data) {
  $scope.album = data;

  var tracks = [];
  if ($rootScope.userInfo.username !== "admin") {
  for (var i = 0; i < data.tracks.length; i++) {
    if (data.tracks[i].user_filter === $rootScope.userInfo.username || data.tracks[i].user_filter === "-1") {
      console.log("reached");
      tracks.push(data.tracks[i]);
    }
  }
   $scope.album.tracks = tracks;
  }

});
```

As you can see here the variable user_filter is getting creating here for the first time and sent to backend database.

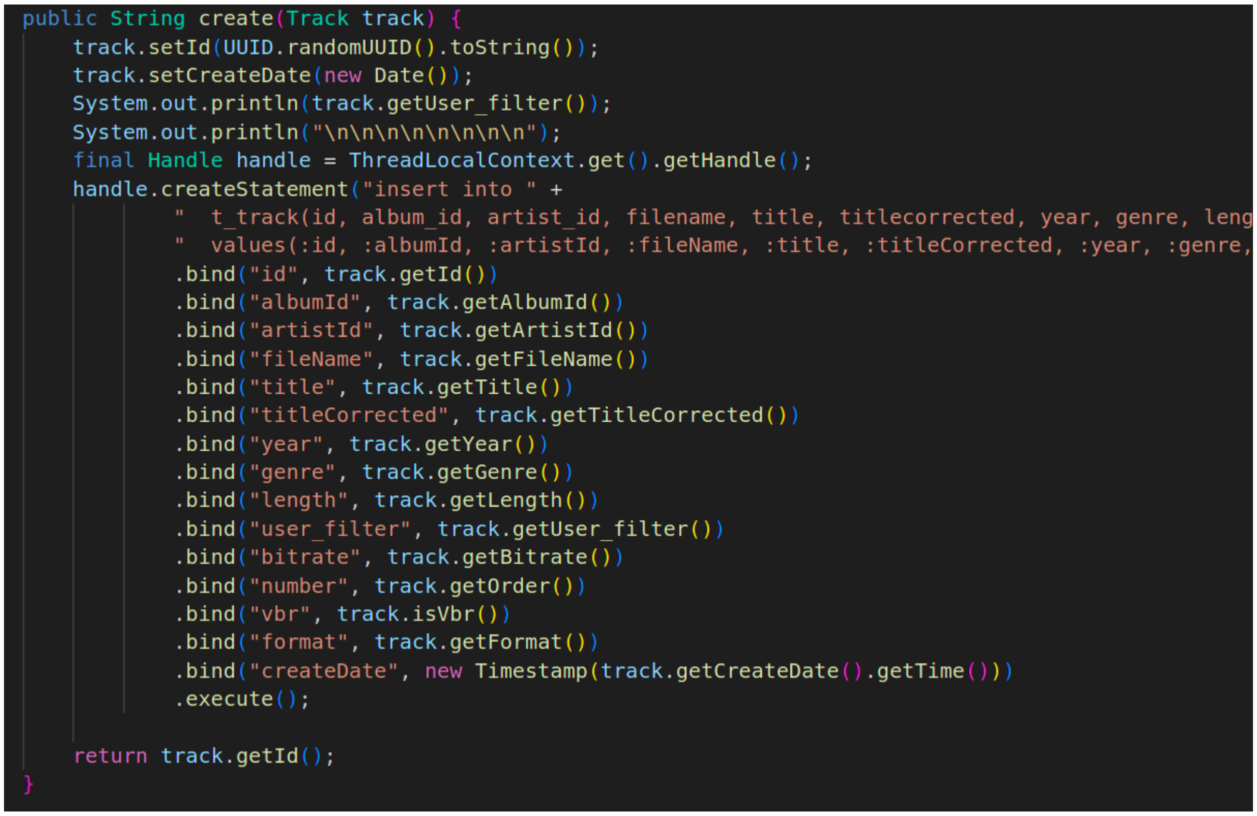


```
var name =  $rootScope.userInfo.username;
if ($scope.checks[ind].check == 1) {
  file.user_filter = name;

}
else {
  file.user_filter = "-1";
}

$scope.checks.splice($scope.files.indexOf(file), 1);
```

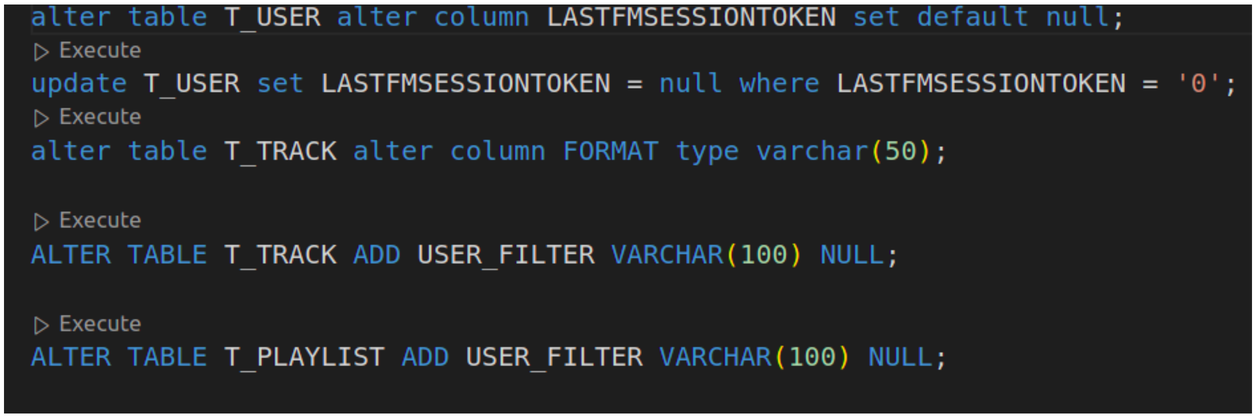
This is the Track Dao file, in which it is finally inserting all the data of track to the database, in here you can see variable user_filter also getting inserted.



```
public String create(Track track) {
    track.setId(UUID.randomUUID().toString());
    track.setCreateDate(new Date());
    System.out.println(track.getUser_filter());
    System.out.println("\n\n\n\n\n\n\n\n");
    final Handle handle = ThreadLocalContext.get().getHandle();
    handle.createStatement("insert into " +
            "  t_track(id, album_id, artist_id, filename, title, titlecorrected, year, genre, leng
            "  values(:id, :albumId, :artistId, :fileName, :title, :titleCorrected, :year, :genre,
            .bind("id", track.getId())
            .bind("albumId", track.getAlbumId())
            .bind("artistId", track.getArtistId())
            .bind("fileName", track.getFileName())
            .bind("title", track.getTitle())
            .bind("titleCorrected", track.getTitleCorrected())
            .bind("year", track.getYear())
            .bind("genre", track.getGenre())
            .bind("length", track.getLength())
            .bind("user_filter", track.getUser_filter())
            .bind("bitrate", track.getBitrate())
            .bind("number", track.getOrder())
            .bind("vbr", track.isVbr())
            .bind("format", track.getFormat())
            .bind("createDate", new Timestamp(track.getCreateDate().getTime()))
            .execute();

    return track.getId();
}
```

Here the database is creating extra USER_FILTER column in the tables.

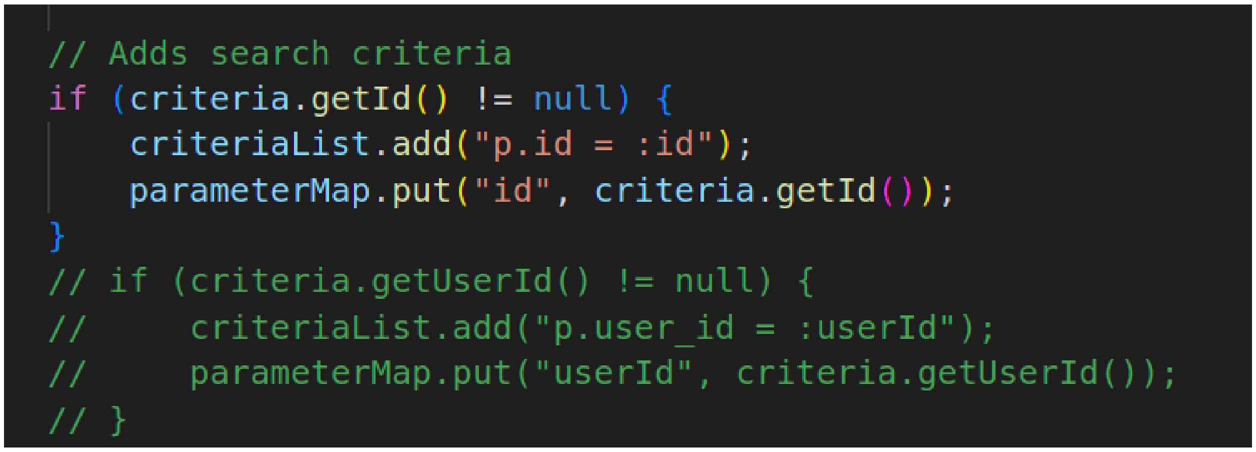


```
alter table T_USER alter column LASTFMSESSIONTOKEN set default null;
▷ Execute
update T_USER set LASTFMSESSIONTOKEN = null where LASTFMSESSIONTOKEN = '0';
▷ Execute
alter table T_TRACK alter column FORMAT type varchar(50);

▷ Execute
ALTER TABLE T_TRACK ADD USER_FILTER VARCHAR(100) NULL;

▷ Execute
ALTER TABLE T_PLAYLIST ADD USER_FILTER VARCHAR(100) NULL;
```

For the playlists as well the code is the same, as you are trying to make it private as well. But all the playlists are private initially to make them private there was some initial filter, which we removed and it is the following commented code.



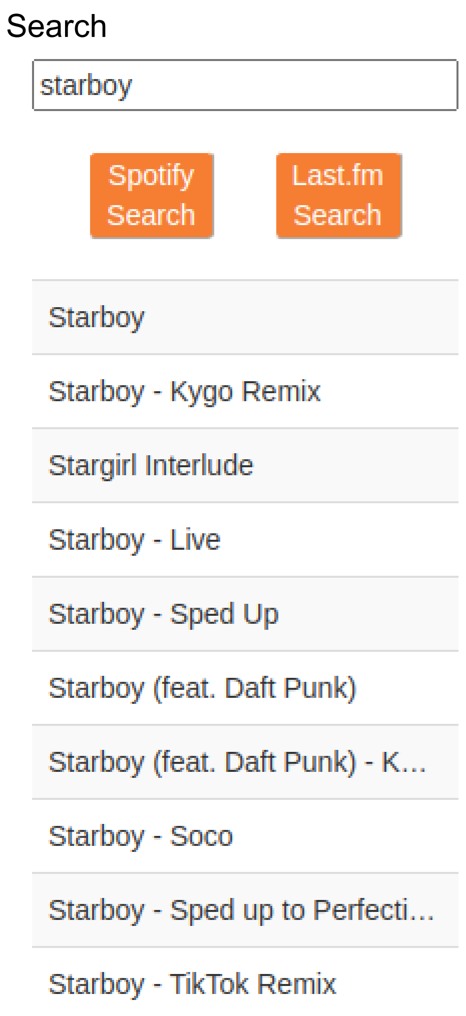
```
// Adds search criteria
if (criteria.getId() != null) {
    criteriaList.add("p.id = :id");
    parameterMap.put("id", criteria.getId());
}
// if (criteria.getUserId() != null) {
//     criteriaList.add("p.user_id = :userId");
//     parameterMap.put("userId", criteria.getUserId());
// }
```

Feature 3: Online Integration

To make it easier to see the results we first added radio buttons and search bars in the frontend.

For Song search, the query given by the user is recorded in the search bar and this query is used to make api calls to the choice of service (Spotify/Last.fm) selected by the user.

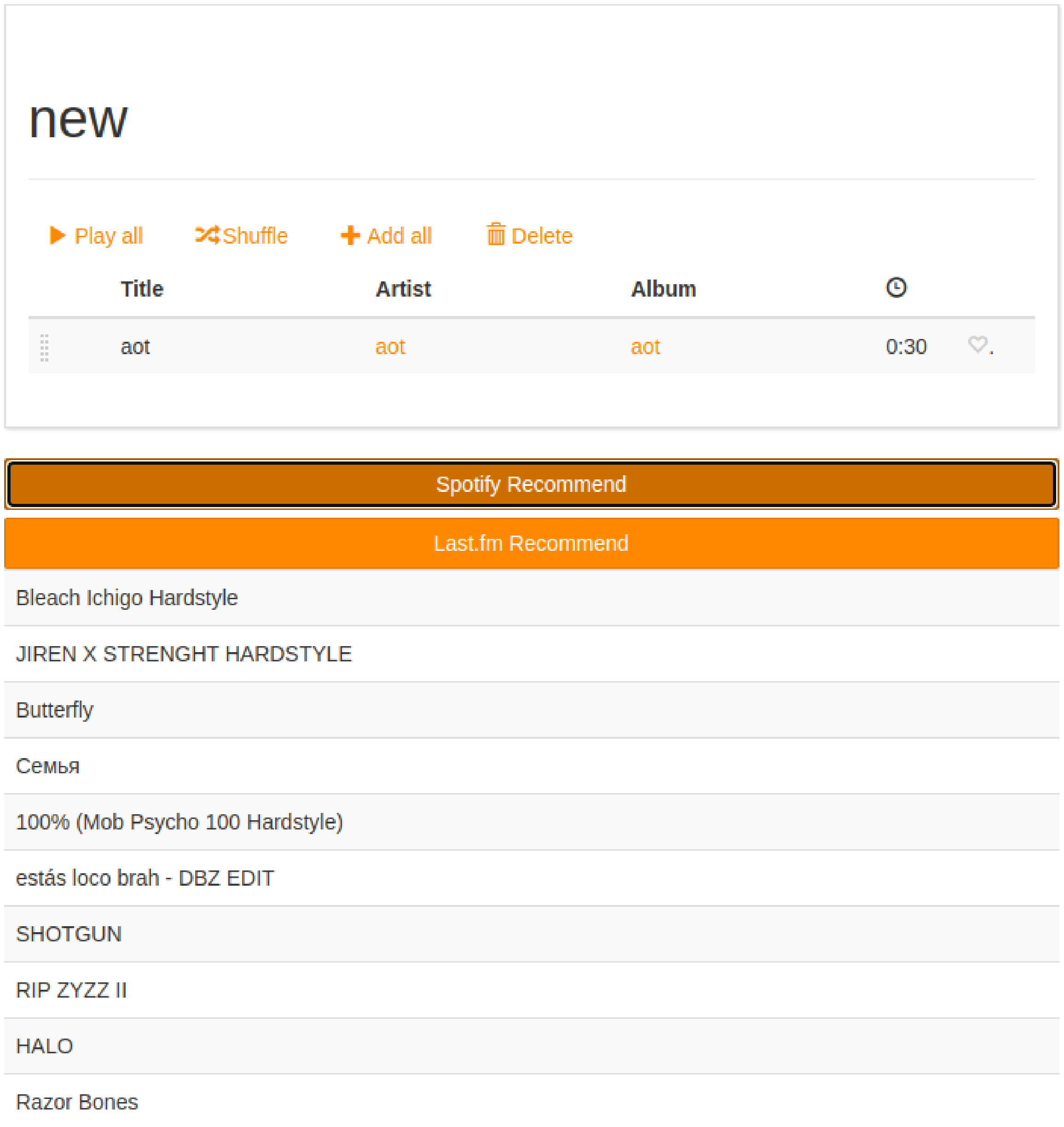
For Recommendations, two buttons are provided one for spotify and one for last.fm, all the songs ids from respective services are retrieved and then they are used to make api calls for recommendations from the chosen service.
UI


Search

starboy

Starboy
Starboy - Kygo Remix
Stargirl Interlude
Starboy - Live
Starboy - Sped Up
Starboy (feat. Daft Punk)
Starboy (feat. Daft Punk) - K…
Starboy - Soco
Starboy - Sped up to Perfecti…
Starboy - TikTok Remix

Recommendations


new

▶ Play all · Shuffle · + Add all · Delete

	Title	Artist	Album	🕒	
	aot	aot	aot	0:30	♡.

Spotify Recommend

Last.fm Recommend

Bleach Ichigo Hardstyle

JIREN X STRENGHT HARDSTYLE

Butterfly

Семья

100% (Mob Psycho 100 Hardstyle)

estás loco brah - DBZ EDIT

SHOTGUN

RIP ZYZZ II

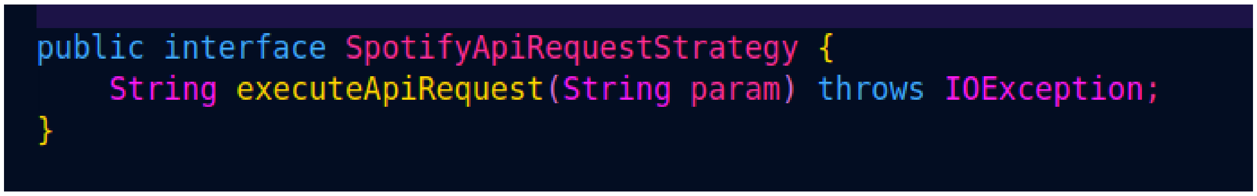
HALO

Razor Bones

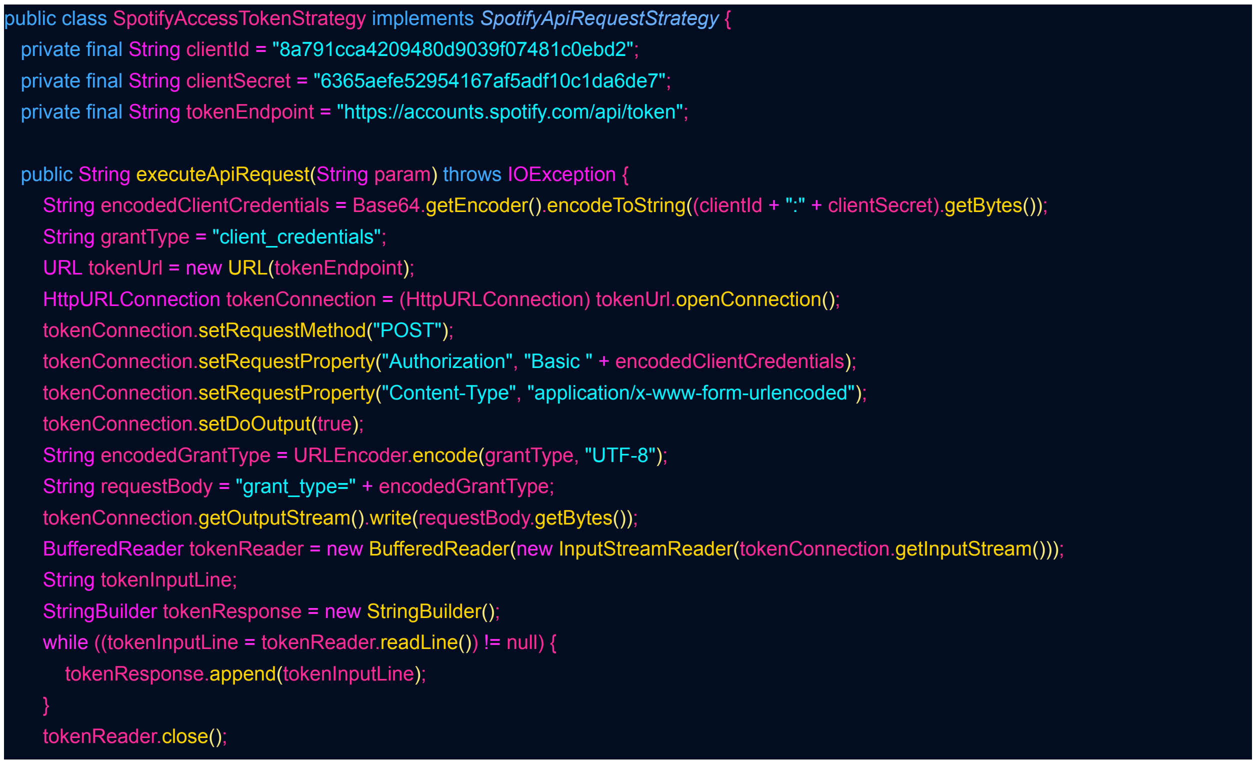
Design Patterns implemented:

1) **Strategy Pattern**
   Added a new folder spotify(location: se-project-1-12-master/music-core/src/main/java/com/sismics/music/core/service/spotify)

- It contains files for executing api requests to provide spotify service
- Used the strategy pattern to implement it where the strategies are SpotifyAccessTokenStrategy, SpotifyRecommendationsStrategy, SpotifySearchStrategy, all these concrete strategies make api requests to spotify, hence an interface SpotifyApiRequestStrategy is used
- SpotifyAccessTokenStrategy makes api calls to spotify to get an access token which will be used to make search and recommendation api calls
- SpotifySearchStrategy makes api calls to spotify with a query containing song name given by the user and returns the output as a string
- SpotifyRecommendationsStrategy makes api calls to spotify with a query containing song names from the playlist and returns the output as a string



```java
public interface SpotifyApiRequestStrategy {
    String executeApiRequest(String param) throws IOException;
}
```



```java
public class SpotifyAccessTokenStrategy implements SpotifyApiRequestStrategy {
  private final String clientId = "8a791cca4209480d9039f07481c0ebd2";
  private final String clientSecret = "6365aefe52954167af5adf10c1da6de7";
  private final String tokenEndpoint = "https://accounts.spotify.com/api/token";

  public String executeApiRequest(String param) throws IOException {
    String encodedClientCredentials = Base64.getEncoder().encodeToString((clientId + ":" + clientSecret).getBytes());
    String grantType = "client_credentials";
    URL tokenUrl = new URL(tokenEndpoint);
    HttpURLConnection tokenConnection = (HttpURLConnection) tokenUrl.openConnection();
    tokenConnection.setRequestMethod("POST");
    tokenConnection.setRequestProperty("Authorization", "Basic " + encodedClientCredentials);
    tokenConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    tokenConnection.setDoOutput(true);
    String encodedGrantType = URLEncoder.encode(grantType, "UTF-8");
    String requestBody = "grant_type=" + encodedGrantType;
    tokenConnection.getOutputStream().write(requestBody.getBytes());
    BufferedReader tokenReader = new BufferedReader(new InputStreamReader(tokenConnection.getInputStream()));
    String tokenInputLine;
    StringBuilder tokenResponse = new StringBuilder();
    while ((tokenInputLine = tokenReader.readLine()) != null) {
      tokenResponse.append(tokenInputLine);
    }
    tokenReader.close();
```

```
        String token = tokenResponse.toString();
        JSONObject json = new JSONObject(token);
        String accessToken = json.getString("access_token");
        return accessToken;
    }
}
```

```
public class SpotifyRecommendationsStrategy implements SpotifyApiRequestStrategy {
    private final String recommendationsEndpoint = "https://api.spotify.com/v1/recommendations";

    public String executeApiRequest(String seed_tracks) throws IOException {
        String AccessToken = new SpotifyAccessTokenStrategy().executeApiRequest("");
        URL recommendationsUrl = new URL(recommendationsEndpoint + "?seed_tracks=" + seed_tracks);
        HttpURLConnection recommendationsConnection = (HttpURLConnection) recommendationsUrl.openConnection();
        recommendationsConnection.setRequestMethod("GET");
        recommendationsConnection.setRequestProperty("Authorization", "Bearer " + AccessToken);

        BufferedReader recommendationsReader = new BufferedReader(new
InputStreamReader(recommendationsConnection.getInputStream()));
        String recommendationsInputLine;
        StringBuilder recommendationsResponse = new StringBuilder();
        while ((recommendationsInputLine = recommendationsReader.readLine()) != null) {
            recommendationsResponse.append(recommendationsInputLine);
        }

        recommendationsReader.close();

        String res = recommendationsResponse.toString();
        return res;
    }
}
```

```
public class SpotifySearchStrategy implements SpotifyApiRequestStrategy {
    private final String searchEndpoint = "https://api.spotify.com/v1/search?type=track&q=";

    public String executeApiRequest(String query) throws IOException {
        String AccessToken = new SpotifyAccessTokenStrategy().executeApiRequest("");
        String encodedQuery = URLEncoder.encode(query, "UTF-8");
        URL searchUrl = new URL(searchEndpoint + encodedQuery);
        HttpURLConnection searchConnection = (HttpURLConnection) searchUrl.openConnection();
        searchConnection.setRequestMethod("GET");
        searchConnection.setRequestProperty("Authorization", "Bearer " + AccessToken);
        BufferedReader searchReader = new BufferedReader(new InputStreamReader(searchConnection.getInputStream()));
        String searchInputLine;
        StringBuilder searchResponse = new StringBuilder();
        while ((searchInputLine = searchReader.readLine()) != null) {
            searchResponse.append(searchInputLine);
        }
        searchReader.close();
    String res = searchResponse.toString();
    return res;
  }
}
```

2) **Command Pattern**
In UserResource.java (/home/venkat/Desktop/se-project-1-12-master/music-web/src/main/java/com/sismics/music/rest/resource/UserResource.java), the actual calls to backend are made and the data needs to be extracted from the string returned from the backend, this process is needs to abstracted for better code clarity hence the command pattern is used to call Search and Recommend commands

- Spotify
  -SpotifyServiceCommand is command interface used to implement SpotifySearchCommand, SpotifyRecommendationsCommand
  -SpotifyServiceInvoker is used to invoke these commands

```
public interface SpotifyServiceCommand {
  public Response execute() throws IOException;
}
```

```
public class SpotifyRecommendationsCommand implements SpotifyServiceCommand {
  private String seed_tracks;
  private SpotifyApiRequestStrategy spotifyService;

  public SpotifyRecommendationsCommand(String seed_tracks, SpotifyApiRequestStrategy spotifyService) {
    this.seed_tracks = seed_tracks;
    this.spotifyService = spotifyService;
  }

  @Override
  public Response execute() throws IOException {

    String response = spotifyService.executeApiRequest(seed_tracks);

    JSONObject resJson = new JSONObject(response);

    JSONArray tracks = resJson.getJSONArray("tracks");
```

```
      JsonObjectBuilder recommendedJson = Json.createObjectBuilder();
      JsonArrayBuilder recommendedNames = Json.createArrayBuilder();

      for(Object each:tracks)
      {
          JSONObject track = (JSONObject) each;

recommendedNames.add(Json.createObjectBuilder().add("name",track.getString("name")).add("id",track.getString("id")));
      }

      recommendedJson.add("tracks",recommendedNames);

      return Response.ok(recommendedJson.build()).build();
  }

}
```

```
public class SpotifySearchCommand implements SpotifyServiceCommand {
  private String song;
  private SpotifyApiRequestStrategy spotifyService;

  public SpotifySearchCommand(String song, SpotifyApiRequestStrategy spotifyService) {
      this.song = song;
      this.spotifyService = spotifyService;
  }

  @Override
  public Response execute() throws IOException {
      // String searchResponse = spotifyService.search(song);

      // SpotifyApiRequestStrategy spotifyService = new SpotifySearchStrategy();

      String searchResponse = spotifyService.executeApiRequest(song);

      JSONObject trackjson = new JSONObject(searchResponse);

      JSONArray tracks = trackjson.getJSONObject("tracks").getJSONArray("items");

      JsonObjectBuilder searchJson = Json.createObjectBuilder();
```

```
        JsonArrayBuilder searchArray = Json.createArrayBuilder();

        for(Object each:tracks)
        {
            JSONObject track = (JSONObject) each;

searchArray.add(Json.createObjectBuilder().add("id",track.getString("id")).add("name",track.getStrin
g("name")));

        }

        searchJson.add("tracks",searchArray);

        return Response.ok(searchJson.build()).build();
    }
}
```

```
public class SpotifyServiceInvoker {
    private SpotifyServiceCommand spotifyServiceCommand;

    public void setSpotifyServiceCommand(SpotifyServiceCommand spotifyServiceCommand) {
        this.spotifyServiceCommand = spotifyServiceCommand;
    }

    public Response invoke() throws IOException {
        return spotifyServiceCommand.execute();
    }
}
```

- Last.fm
  -LastFmServiceCommand is command interface used to implement LastFmSearchCommand, LastFmRecommendationsCommand
  -LastFmServiceInvoker is used to invoke these commands

```
public interface LastFmServiceCommand {
    public Response execute() throws IOException;
}
```

```
public class LastFmSearchCommand implements LastFmServiceCommand {
  private String song;
  private LastFmService lastFmService;

  public LastFmSearchCommand(String song, LastFmService lastFmService) {
      this.song = song;
      this.lastFmService = lastFmService;
  }

  @Override
  public Response execute() throws IOException {
      String searchResponse = lastFmService.search(song);

      // String searchResponse = LastFmService.search(song);

      JSONObject trackjson = new JSONObject(searchResponse);

      JSONArray tracks =
trackjson.getJSONObject("results").getJSONObject("trackmatches").getJSONArray("track");

      JsonObjectBuilder searchJson = Json.createObjectBuilder();
      JsonArrayBuilder searchArray = Json.createArrayBuilder();

      for(Object each:tracks)
      {
          JSONObject track = (JSONObject) each;

searchArray.add(Json.createObjectBuilder().add("id",track.getString("mbid")).add("name",track.getSt
ring("name")).add("mbid",track.getString("mbid")));

      }

      searchJson.add("tracks",searchArray);
      // return renderJson(searchJson);

      return Response.ok(searchJson.build()).build();
  }
}
```

```java
public class LastFmRecommendationsCommand implements LastFmServiceCommand {
  private String seed_mbids;
  private LastFmService lastFmService;

  public LastFmRecommendationsCommand(String seed_mbids, LastFmService lastFmService) {
      this.seed_mbids = seed_mbids;
      this.lastFmService = lastFmService;
  }

  @Override
  public Response execute() throws IOException {
      String searchResponse = LastFmService.recommend(seed_mbids);

      System.out.println("searchResponse: " + searchResponse);

      JSONObject trackjson = new JSONObject(searchResponse);

      JSONArray tracks = trackjson.getJSONObject("similartracks").getJSONArray("track");

      JsonObjectBuilder recommendJson = Json.createObjectBuilder();
      JsonArrayBuilder recommendArray = Json.createArrayBuilder();

      for(Object each:tracks)
      {
          JSONObject track = (JSONObject) each;
          recommendArray.add(Json.createObjectBuilder().add("name",track.getString("name")));
      }

      recommendJson.add("tracks",recommendArray);
      // return renderJson(recommendJson);
      return Response.ok(recommendJson.build()).build();
  }
}
```

```java
public class LastFmServiceInvoker {
  private LastFmServiceCommand lastFmServiceCommand;

  public void setLastFmServiceCommand(LastFmServiceCommand lastFmServiceCommand) {
      this.lastFmServiceCommand = lastFmServiceCommand;
  }

  public Response invoke() throws IOException {
      return lastFmServiceCommand.execute();
  }
}
```

UserResource.java

```
@GET
    @Path("lastfmSearch")
    public Response lastFmSearch(@QueryParam("song") String song) throws IOException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        LastFmServiceInvoker invoker = new LastFmServiceInvoker();
        LastFmService lastFmService = new LastFmService();

        LastFmServiceCommand searchCommand = new LastFmSearchCommand(song,
lastFmService);
        invoker.setLastFmServiceCommand(searchCommand);
        Response response = invoker.invoke();

        return response;
    }

    @GET
    @Path("lastfmRecommendations")
    public Response lastFmRecommendations(@QueryParam("seed_mbids") String seed_mbids)
throws IOException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        LastFmServiceInvoker invoker = new LastFmServiceInvoker();
        LastFmService lastFmService = new LastFmService();

```

```
      LastFmServiceCommand recommendationsCommand = new
LastFmRecommendationsCommand(seed_mbids, lastFmService);
      invoker.setLastFmServiceCommand(recommendationsCommand);
      Response response = invoker.invoke();

      return response;

  }

  @GET
  @Path("spotifySearch")
  public Response search(@QueryParam("song") String song) throws IOException {


      SpotifyServiceInvoker invoker = new SpotifyServiceInvoker();
      SpotifyApiRequestStrategy SpotifySearch = new SpotifySearchStrategy();

      SpotifyServiceCommand searchCommand = new SpotifySearchCommand(song,
SpotifySearch);

      invoker.setSpotifyServiceCommand(searchCommand);

      Response response = invoker.invoke();

      return response;

  }


  @GET
  @Path("spotifyRecommendations")
  public Response recommend(@QueryParam("seed_tracks") String seed_tracks) throws
IOException {

      SpotifyServiceInvoker invoker = new SpotifyServiceInvoker();
      SpotifyApiRequestStrategy SpotifyRecommend = new SpotifyRecommendationsStrategy();

      SpotifyServiceCommand recommendationsCommand = new
SpotifyRecommendationsCommand(seed_tracks, SpotifyRecommend);
```

```
    invoker.setSpotifyServiceCommand(recommendationsCommand);

    Response response = invoker.invoke();

    return response;

}
```