SE report 3

Team Members:

Chandu Chegu: 202220162

Nemalikanti V M Dheeraj: 2022201022

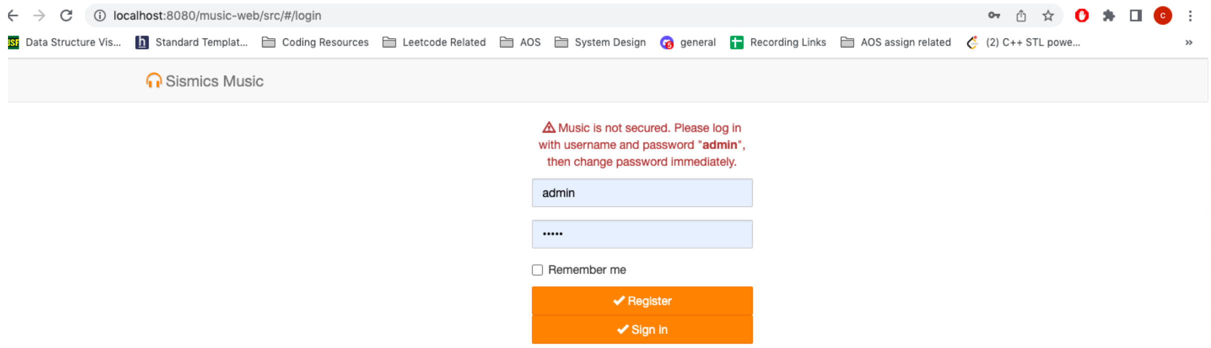
Krishna Chaitanya Dama: 2021202005

Sai Sumit: 2022202004

Prem : 2022201036

Feature 1: Better User Management

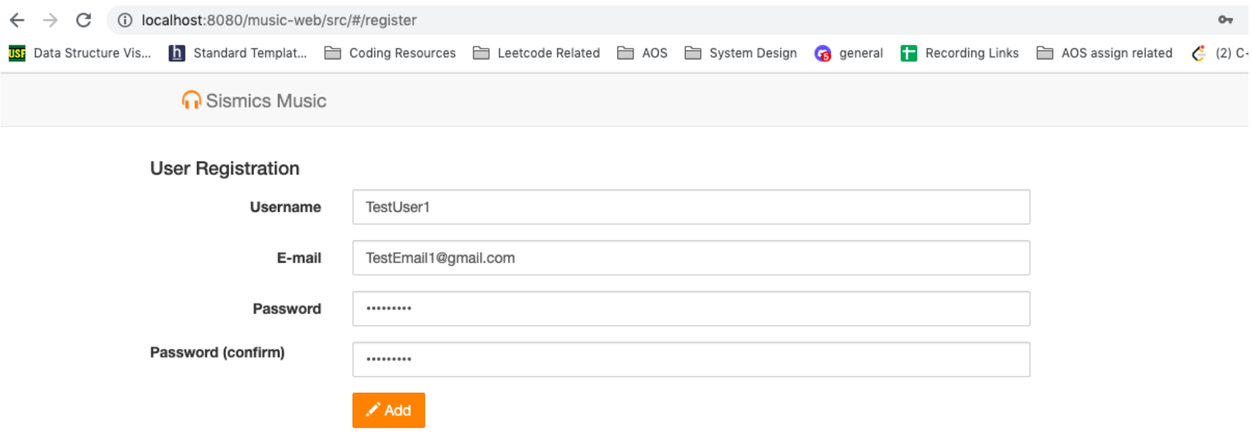
We created **Register** option for new users directly in the login page instead of login via admin and manual creation of user.



⚠ Music is not secured. Please log in with username and password "**admin**", then change password immediately.

admin

•••••



User Registration

Username: TestUser1

E-mail: TestEmail1@gmail.com

Password: •••••••••

Password (confirm): •••••••••

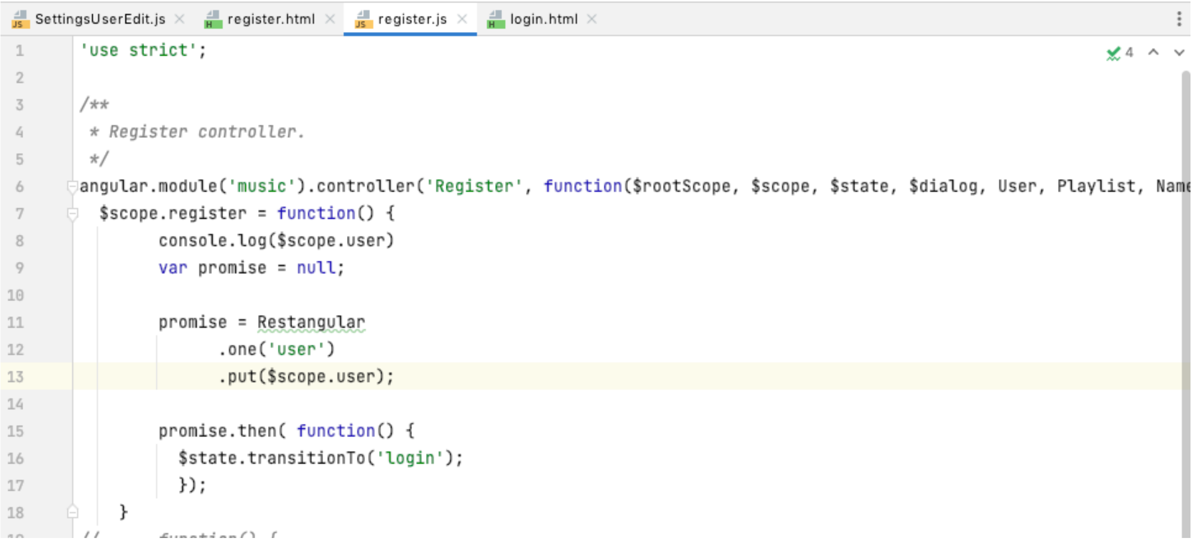
Flow : Once user lands in login page, if it is a new user, click on register button, and will be directed to User Registration page where he will fill his details and when he clicks on add, it is recorded in database.

Changes done in html file :

- **login.html** : added a button and on click, redirected to /register path
- **register.html** : Created new html file - register.html and used existing code in settings.user.html file with slight modifications. When user clicks on add, invoked register.js controller.
- **index.html** : declared register.js controller path

Changes done in controller files :

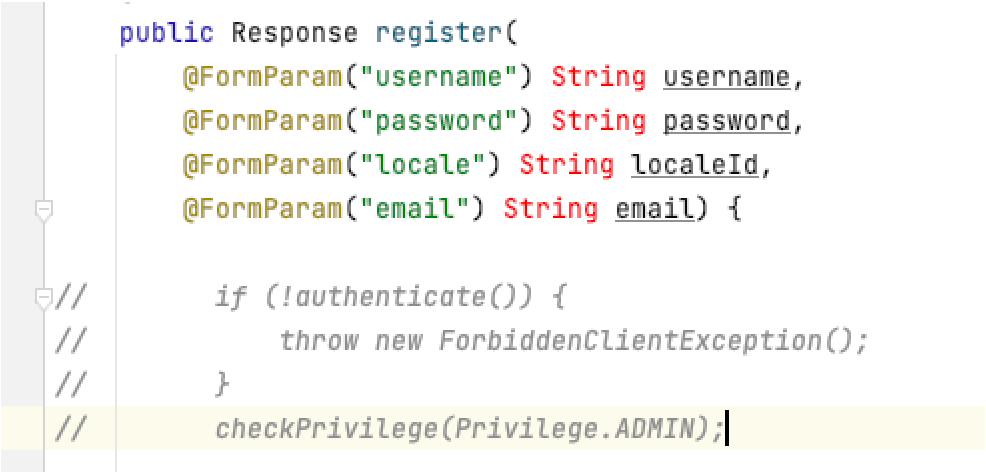
- **register.js :** Used RestAngular to talk with backend and once the user is created, redirected to login page, so that he can login.



```
'use strict';

/**
 * Register controller.
 */
angular.module('music').controller('Register', function($rootScope, $scope, $state, $dialog, User, Playlist, Name
  $scope.register = function() {
        console.log($scope.user)
        var promise = null;

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

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

Changes done in Java files :

- **UserResource.java** : To make the new user register, we need to remove admin privilege check during register.

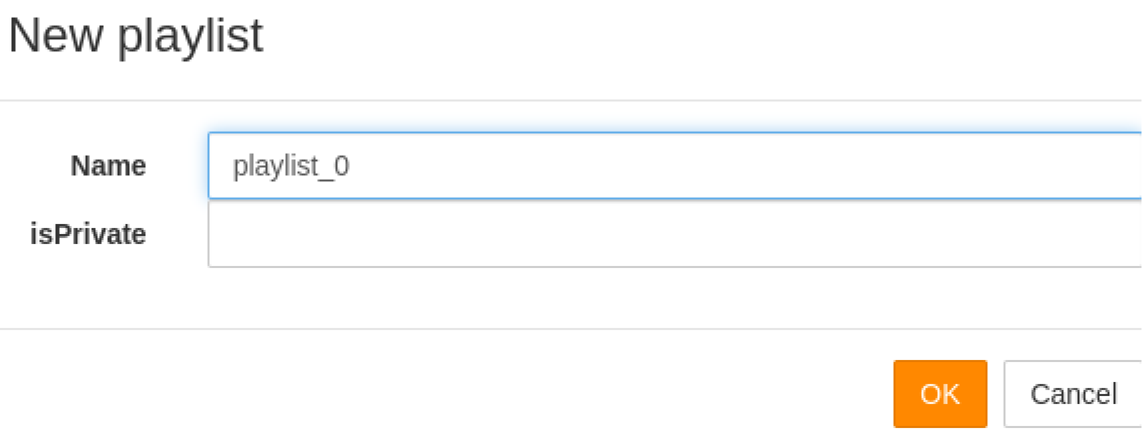


```
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) {

//          if (!authenticate()) {
//              throw new ForbiddenClientException();
//          }
//          checkPrivilege(Privilege.ADMIN);
```

Feature 2: Making public playlists and imported music private

In this task we implemented 2 features

Making public playlists: Once a playlist is created only the specific user who created the playlist can see it. We added a new feature called public playlists such that when the playlist is created we give an option to the user to select the type of playlist he wants to create. The user should mention the type of playlist he wants to create by mentioning 1 if user wants to create a private playlist or 0 if wants to create a public playlist.



New playlist

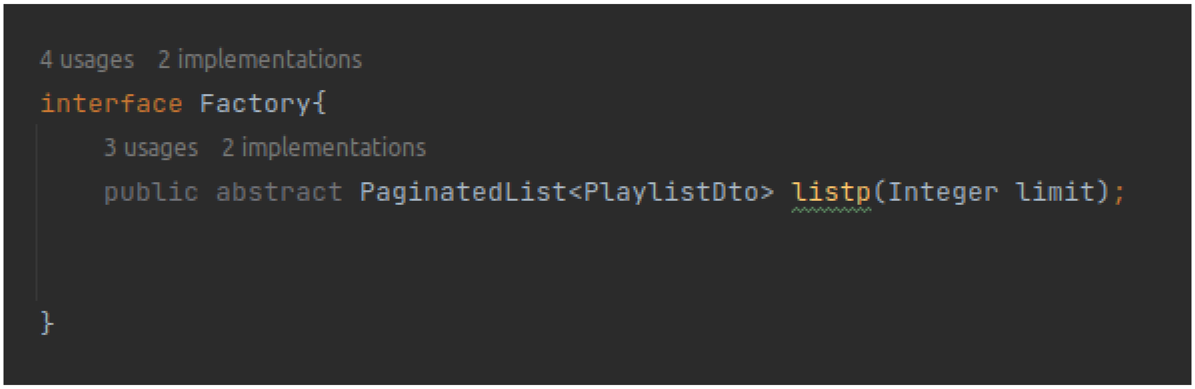
Name: playlist_0

isPrivate:

Making music private: When music is imported it is visible to all the users . So now we have given an option to the user whether to keep it public or make it private.

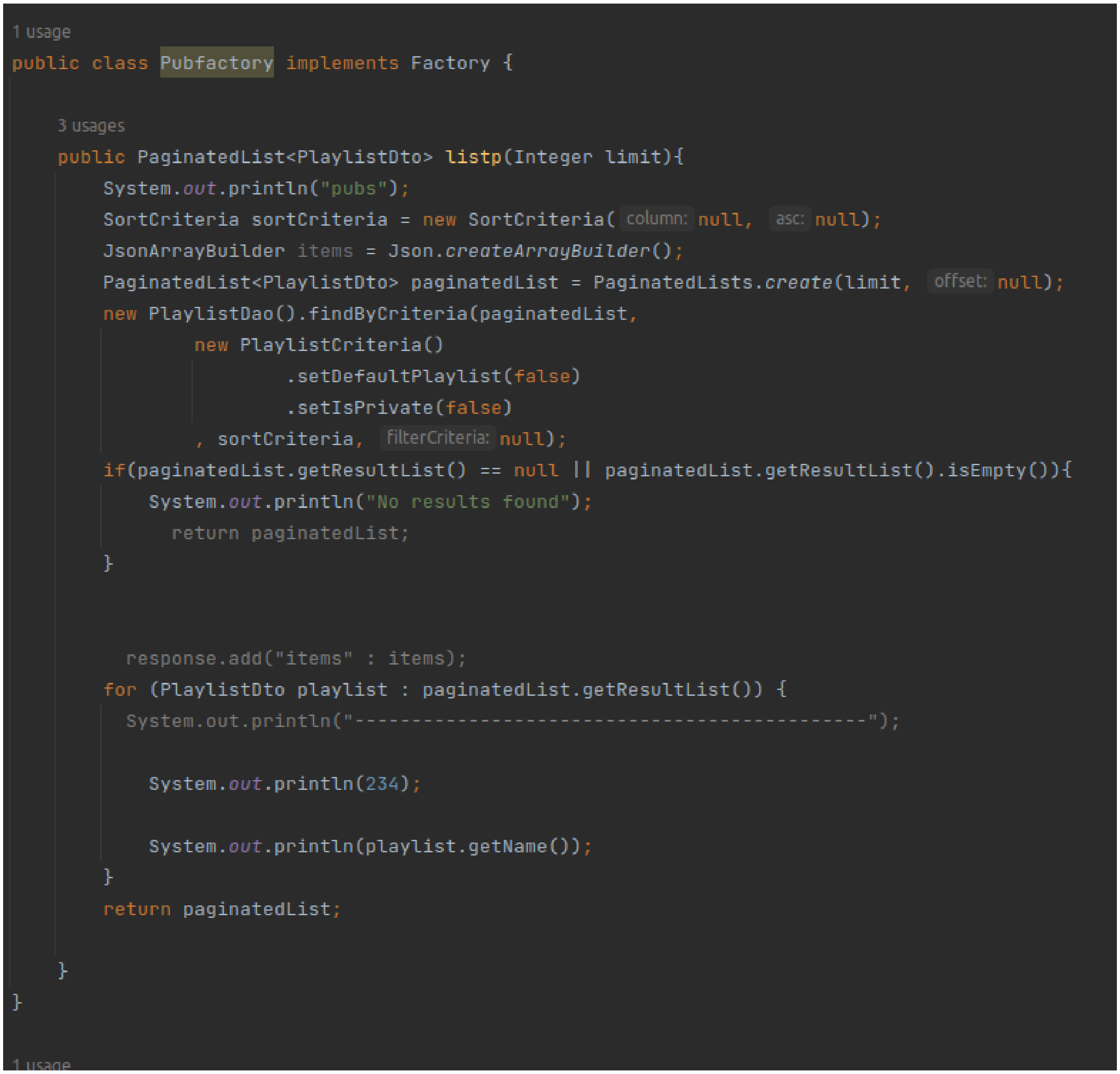
Design patterns used:

We used factory pattern.We created a factory interface.Two classes implements the interface.The factory interface contains the method **listp** which is implemented by the base classes.

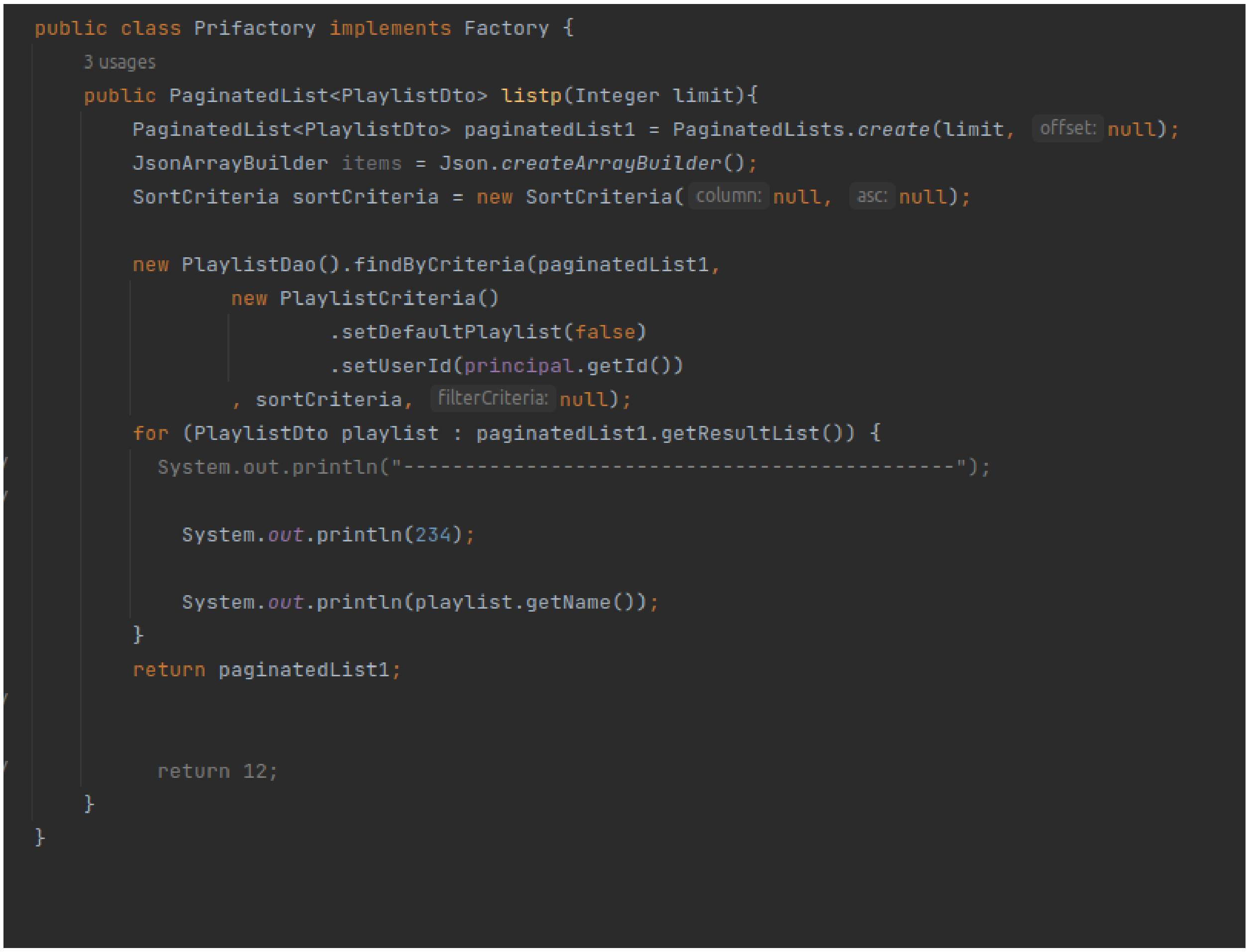


```
4 usages  2 implementations
interface Factory{
    3 usages  2 implementations
    public abstract PaginatedList<PlaylistDto> listp(Integer limit);


}
```

Each classes that implement the factory interface are Pubfactory and Prifactory. Pubfactory is used to get all the playlists that are public and prifactory is used to get all the playlists of the specific user.

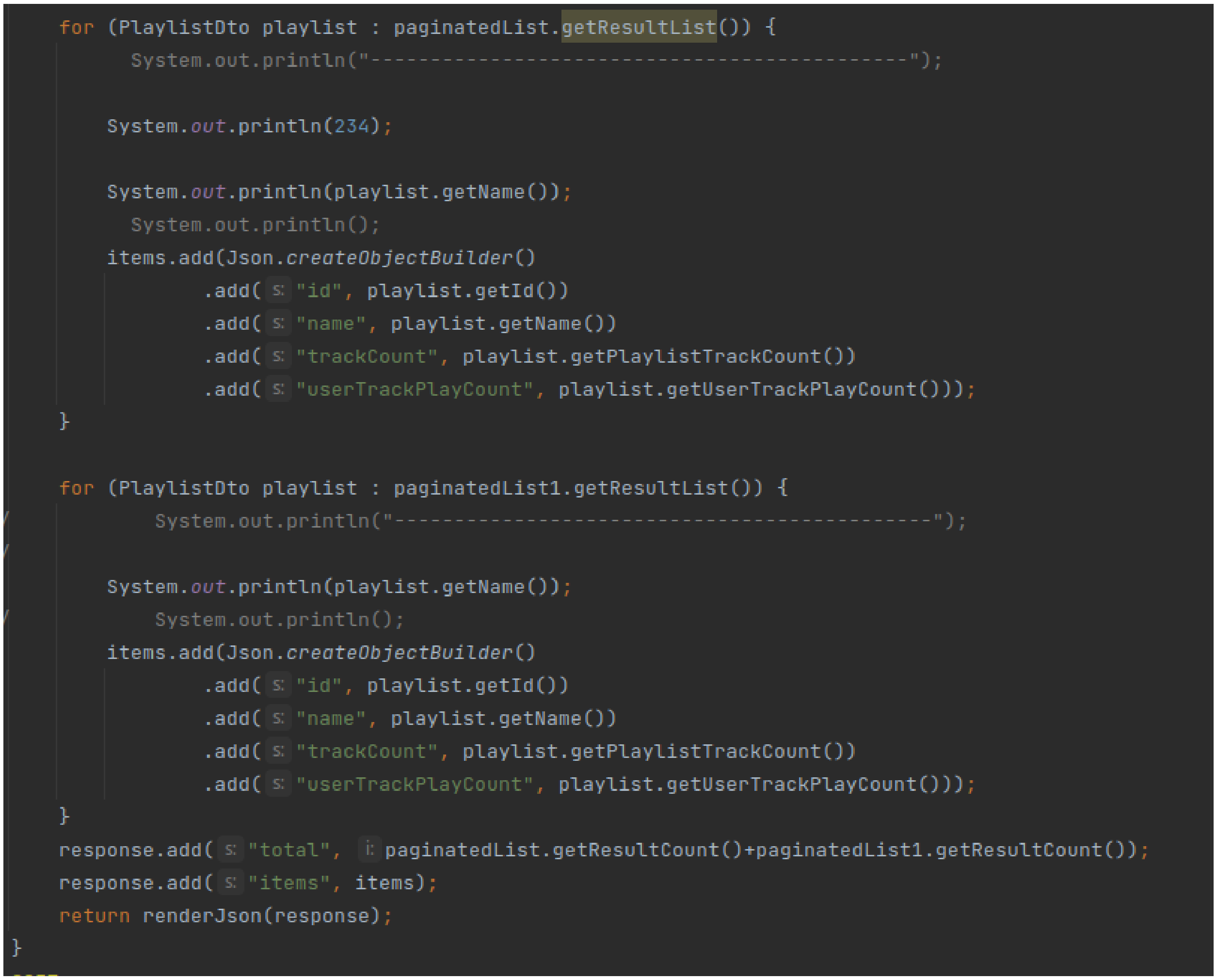


```
1 usage
public class Pubfactory implements Factory {

    3 usages
    public PaginatedList<PlaylistDto> listp(Integer limit){
        System.out.println("pubs");
        SortCriteria sortCriteria = new SortCriteria( column: null,  asc: null);
        JsonArrayBuilder items = Json.createArrayBuilder();
        PaginatedList<PlaylistDto> paginatedList = PaginatedLists.create(limit,  offset: null);
        new PlaylistDao().findByCriteria(paginatedList,
                new PlaylistCriteria()
                        .setDefaultPlaylist(false)
                        .setIsPrivate(false)
                , sortCriteria,  filterCriteria: null);
        if(paginatedList.getResultList() == null || paginatedList.getResultList().isEmpty()){
            System.out.println("No results found");
              return paginatedList;
        }


          response.add("items" : items);
        for (PlaylistDto playlist : paginatedList.getResultList()) {
          System.out.println("---------------------------------------------");

            System.out.println(234);

            System.out.println(playlist.getName());
        }
        return paginatedList;

    }
}

1 usage
```



```
public class Prifactory implements Factory {
    3 usages
    public PaginatedList<PlaylistDto> listp(Integer limit){
        PaginatedList<PlaylistDto> paginatedList1 = PaginatedLists.create(limit,  offset: null);
        JsonArrayBuilder items = Json.createArrayBuilder();
        SortCriteria sortCriteria = new SortCriteria( column: null,  asc: null);

        new PlaylistDao().findByCriteria(paginatedList1,
                new PlaylistCriteria()
                        .setDefaultPlaylist(false)
                        .setUserId(principal.getId())
                , sortCriteria,  filterCriteria: null);
        for (PlaylistDto playlist : paginatedList1.getResultList()) {
          System.out.println("---------------------------------------------");

            System.out.println(234);

            System.out.println(playlist.getName());
        }
        return paginatedList1;


          return 12;
    }
}
```

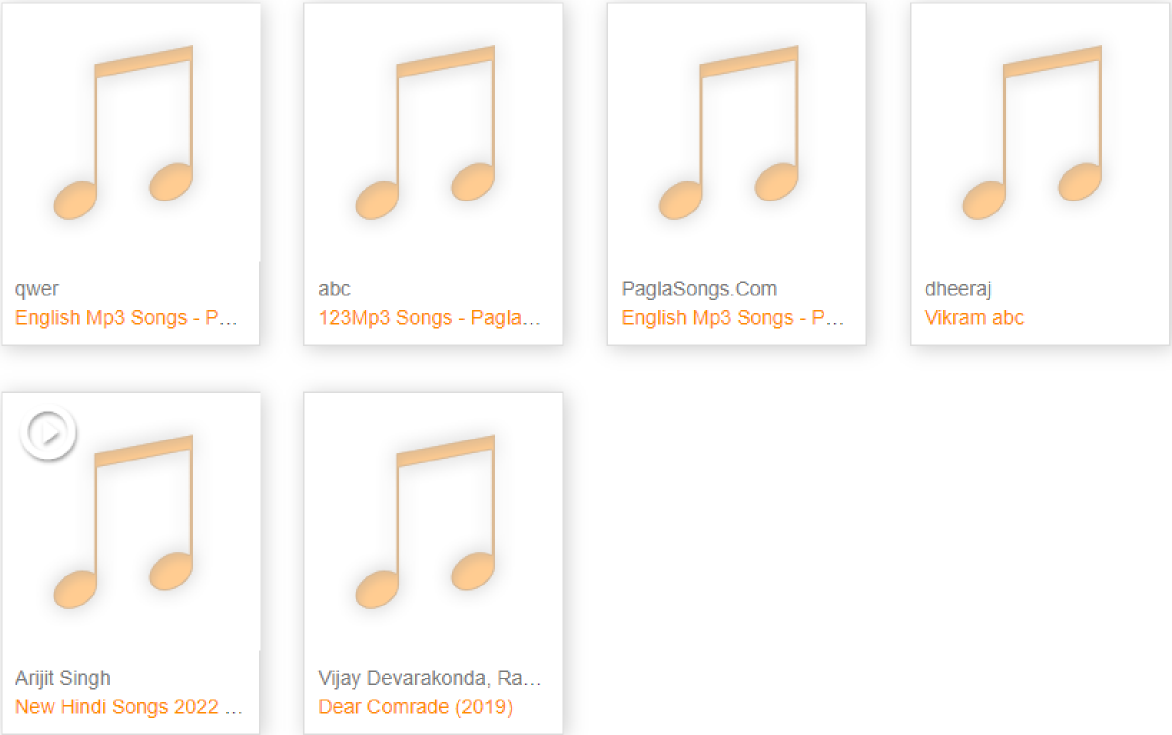
Below is the factory class



```
    for (PlaylistDto playlist : paginatedList.getResultList()) {
          System.out.println("---------------------------------------------");

        System.out.println(234);

        System.out.println(playlist.getName());
          System.out.println();
        items.add(Json.createObjectBuilder()
                .add( s: "id", playlist.getId())
                .add( s: "name", playlist.getName())
                .add( s: "trackCount", playlist.getPlaylistTrackCount())
                .add( s: "userTrackPlayCount", playlist.getUserTrackPlayCount()));
    }

    for (PlaylistDto playlist : paginatedList1.getResultList()) {
            System.out.println("---------------------------------------------");

        System.out.println(playlist.getName());
            System.out.println();
        items.add(Json.createObjectBuilder()
                .add( s: "id", playlist.getId())
                .add( s: "name", playlist.getName())
                .add( s: "trackCount", playlist.getPlaylistTrackCount())
                .add( s: "userTrackPlayCount", playlist.getUserTrackPlayCount()));
    }
    response.add( s: "total",  i: paginatedList.getResultCount()+paginatedList1.getResultCount());
    response.add( s: "items", items);
    return renderJson(response);
}
```

In this way using factory pattern to create objects that are used to get required playlists.

Feature 2: Making music visible only to the user



qwer
English Mp3 Songs - P…

abc
123Mp3 Songs - Pagla…

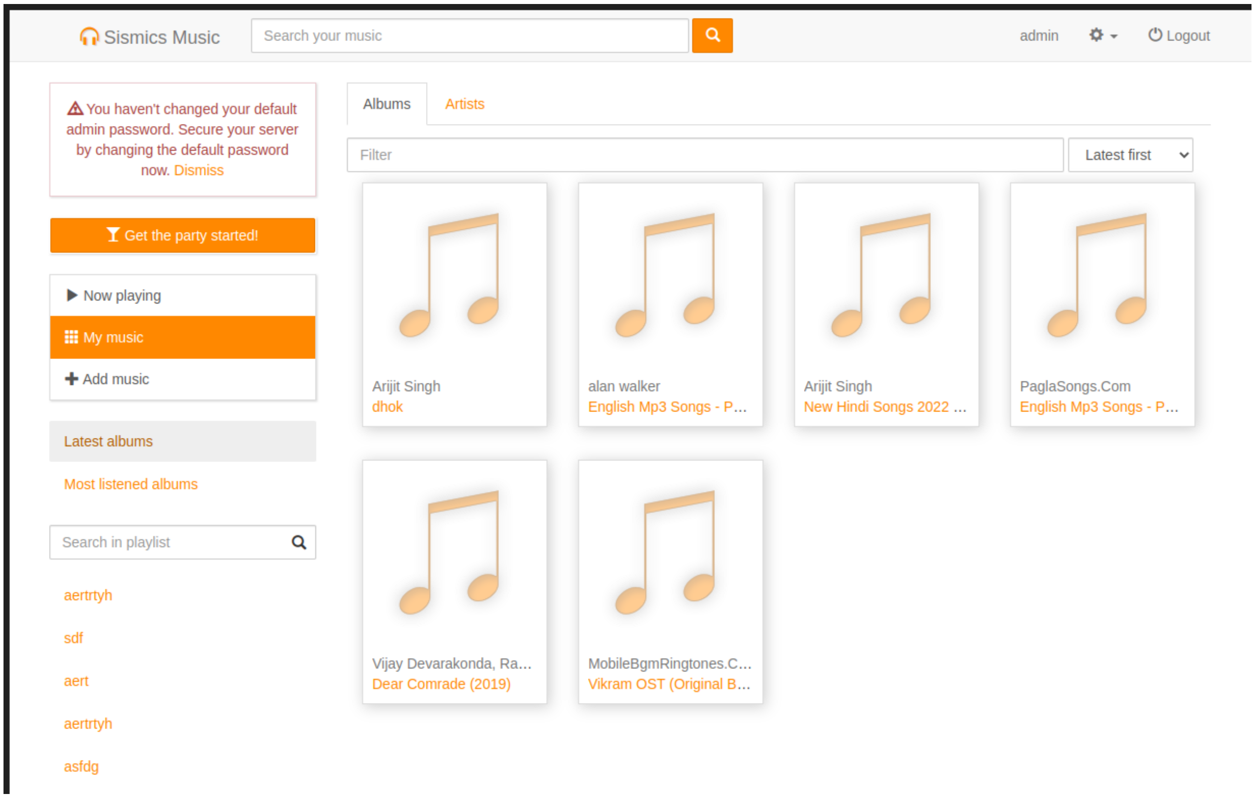
PaglaSongs.Com
English Mp3 Songs - P…

dheeraj
Vikram abc

Arijit Singh
New Hindi Songs 2022 …

Vijay Devarakonda, Ra…
Dear Comrade (2019)

After changes:



You haven't changed your default admin password. Secure your server by changing the default password now. Dismiss

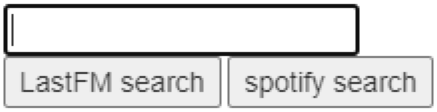
Feature 3: Online Integration

In this task we implemented 2 features:

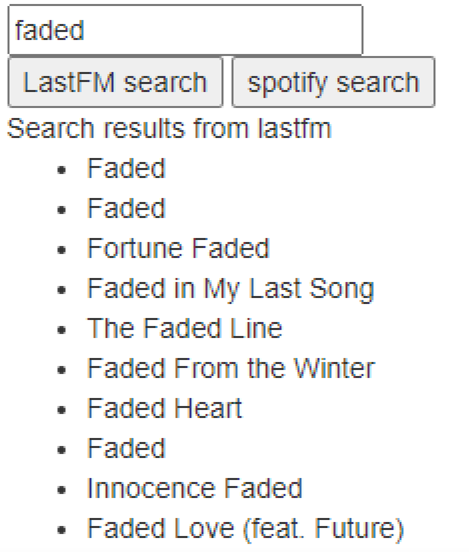
Search for songs: We used LastFM and Spotify APIs to allow users to search for songs in Music

Song Recommendations: Music allows us to create playlists. LastFM and Spotify APIs enable users to get recommendations similar to provided songs. We used the APIs mentioned above and implemented two buttons. Upon clicking them we get recommendations based on our current playlist.

Updated Frontend:



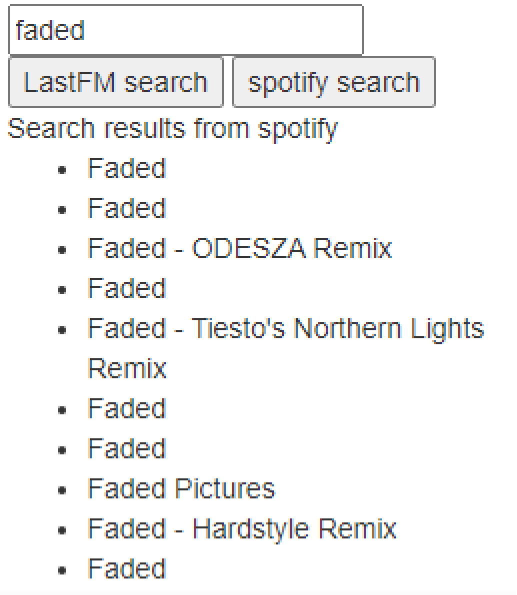
Upon entering a song and clicking on LastFM search or Spotify search, we get our songs as shown below.



faded

Search results from lastfm

- Faded
- Faded
- Fortune Faded
- Faded in My Last Song
- The Faded Line
- Faded From the Winter
- Faded Heart
- Faded
- Innocence Faded
- Faded Love (feat. Future)

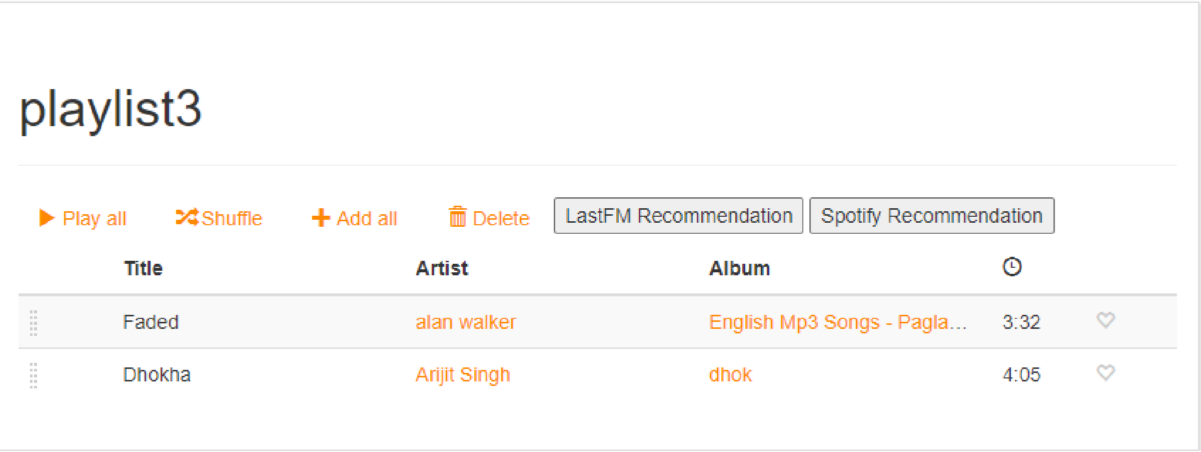


faded

Search results from spotify

- Faded
- Faded
- Faded - ODESZA Remix
- Faded
- Faded - Tiesto's Northern Lights Remix
- Faded
- Faded
- Faded Pictures
- Faded - Hardstyle Remix
- Faded

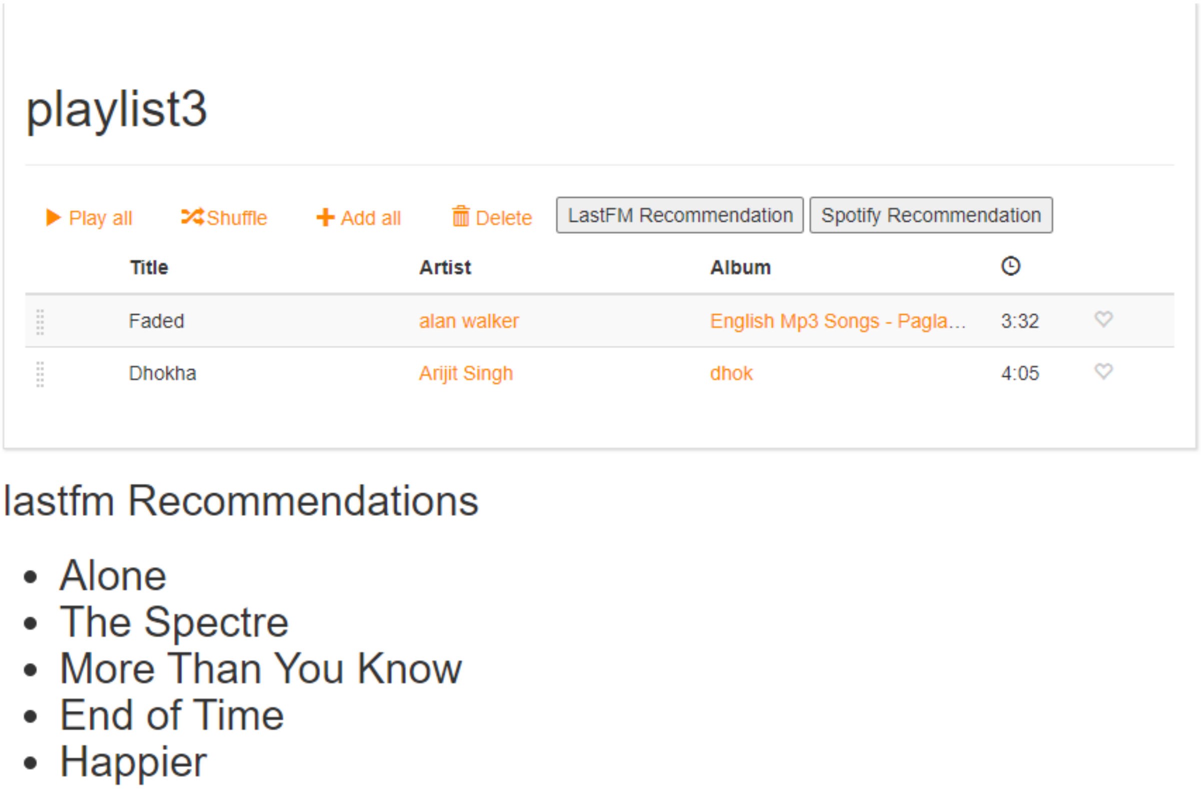
We also created two new buttons in the playlist:



playlist3

Title	Artist	Album	🕓	
Faded	alan walker	English Mp3 Songs - Pagla...	3:32	♡
Dhokha	Arijit Singh	dhok	4:05	♡

So, upon clicking LastFM Recommendation button we get the recommendations as shown below



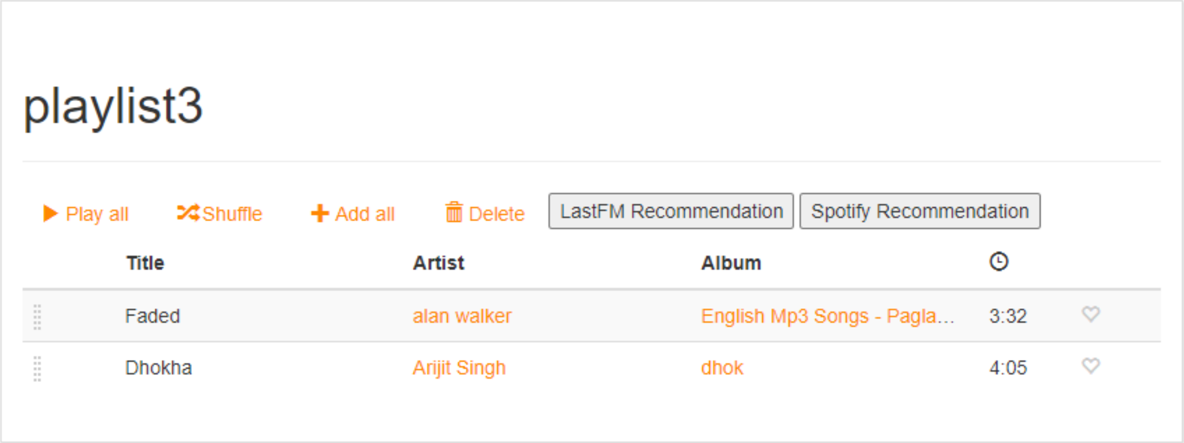
playlist3

Title	Artist	Album	🕒
Faded	alan walker	English Mp3 Songs - Pagla…	3:32
Dhokha	Arijit Singh	dhok	4:05

lastfm Recommendations

- Alone
- The Spectre
- More Than You Know
- End of Time
- Happier

Similarly upon clicking Spotify Recommendation button we get the recommendations as shown below



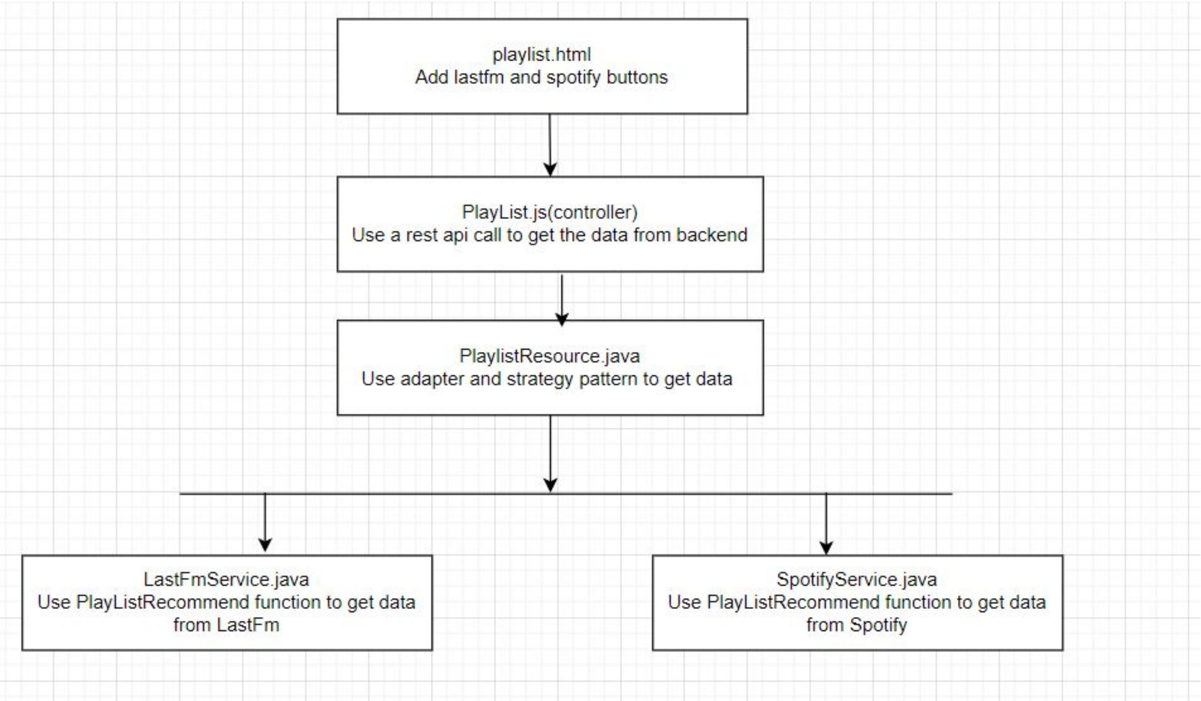
Title	Artist	Album	🕒	
Faded	alan walker	English Mp3 Songs - Pagla…	3:32	♡
Dhokha	Arijit Singh	dhok	4:05	♡

spotify Recommendations

- Running Out Of Roses
- All The Time
- STAY (with Justin Bieber)
- Lonely Together - (feat. Rita Ora) Alan Walker Remix
- There's Nothing Holdin' Me Back
- Locha-E-Ulfat
- Looking For Love
- Qismat
- Hosanna
- Arijit Singh Mashup (By DJ Paroma)

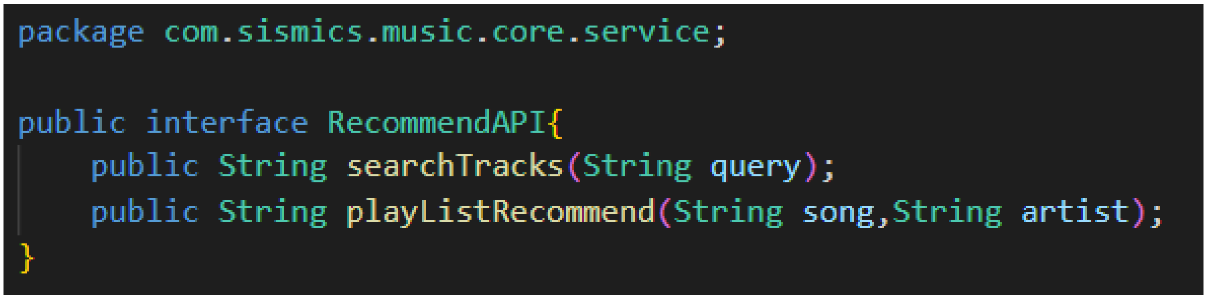
Design Patterns Used:

This is the flow in which we make changes for Recommendation function


playlist.html
Add lastfm and spotify buttons
PlayList.js(controller)
Use a rest api call to get the data from backend
PlaylistResource.java
Use adapter and strategy pattern to get data
LastFmService.java
Use PlayListRecommend function to get data from LastFm
SpotifyService.java
Use PlayListRecommend function to get data from Spotify

We used strategy pattern for implementing services i.e LastFM service .

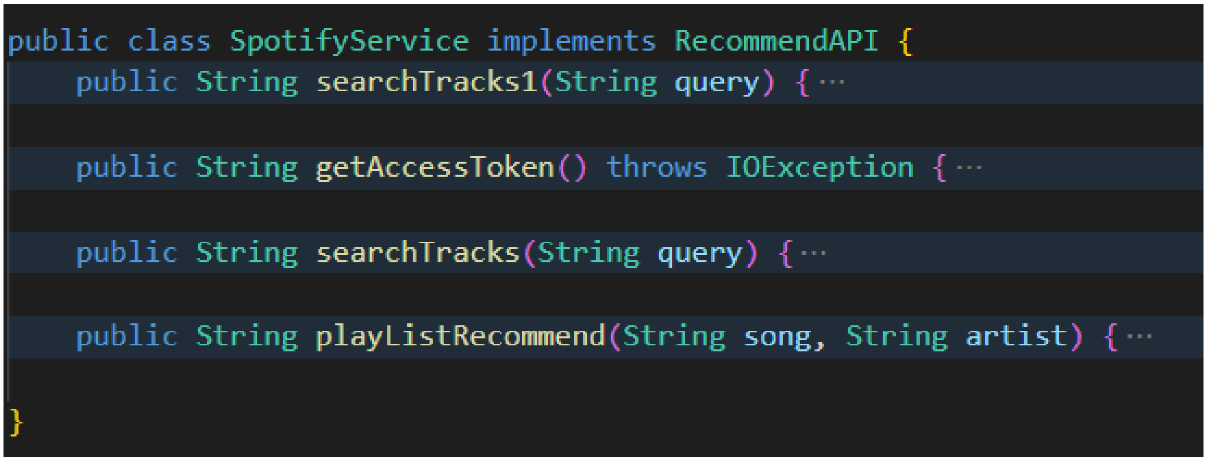
We created an interface RecommendAPI.



```
package com.sismics.music.core.service;

public interface RecommendAPI{
    public String searchTracks(String query);
    public String playListRecommend(String song,String artist);
}
```

The above interface has two methods searchTracks and playListRecommend.

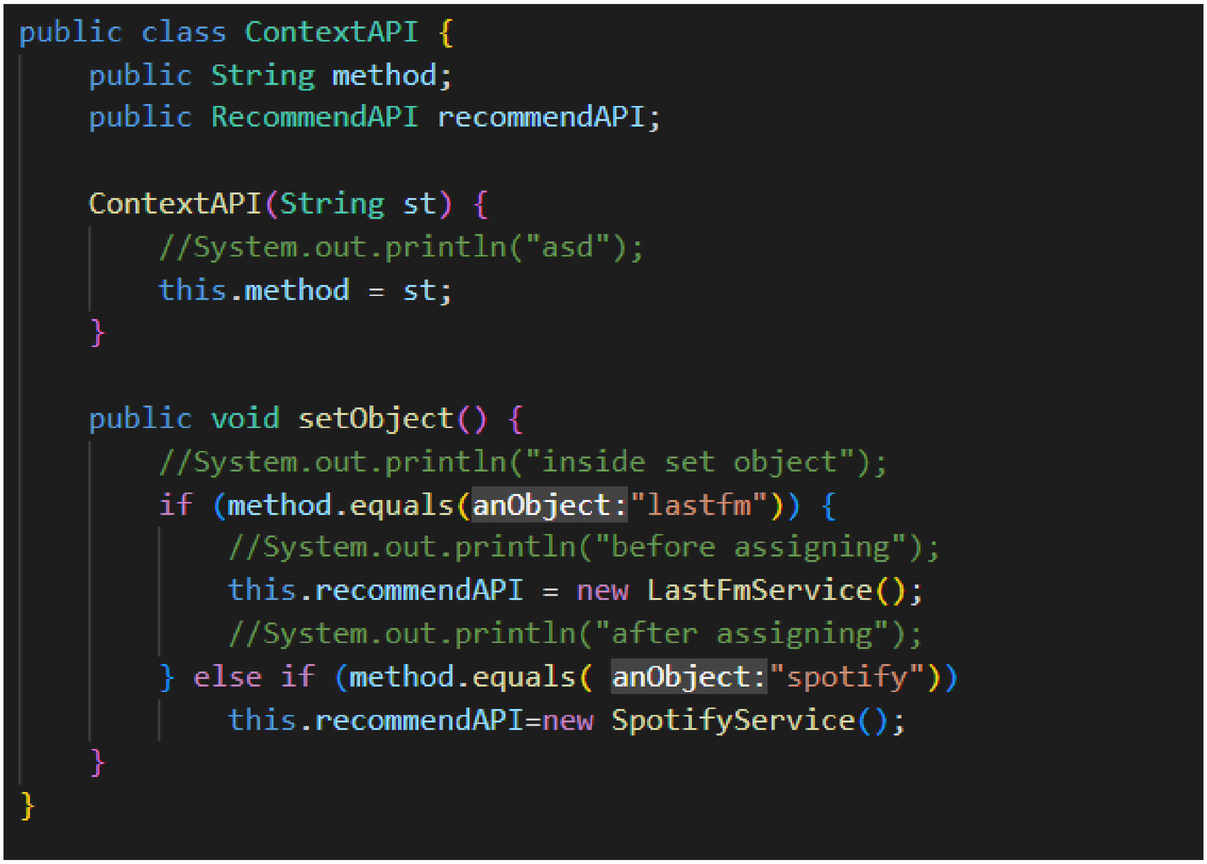
Also we created an additional class for Spotify named SpotifyService. This class implements the RecommendAPI interface



```
public class SpotifyService implements RecommendAPI {
    public String searchTracks1(String query) {···

    public String getAccessToken() throws IOException {···

    public String searchTracks(String query) {···

    public String playListRecommend(String song, String artist) {···

}
```

Similarly LastFM service also implements the RecommendAPI interface.

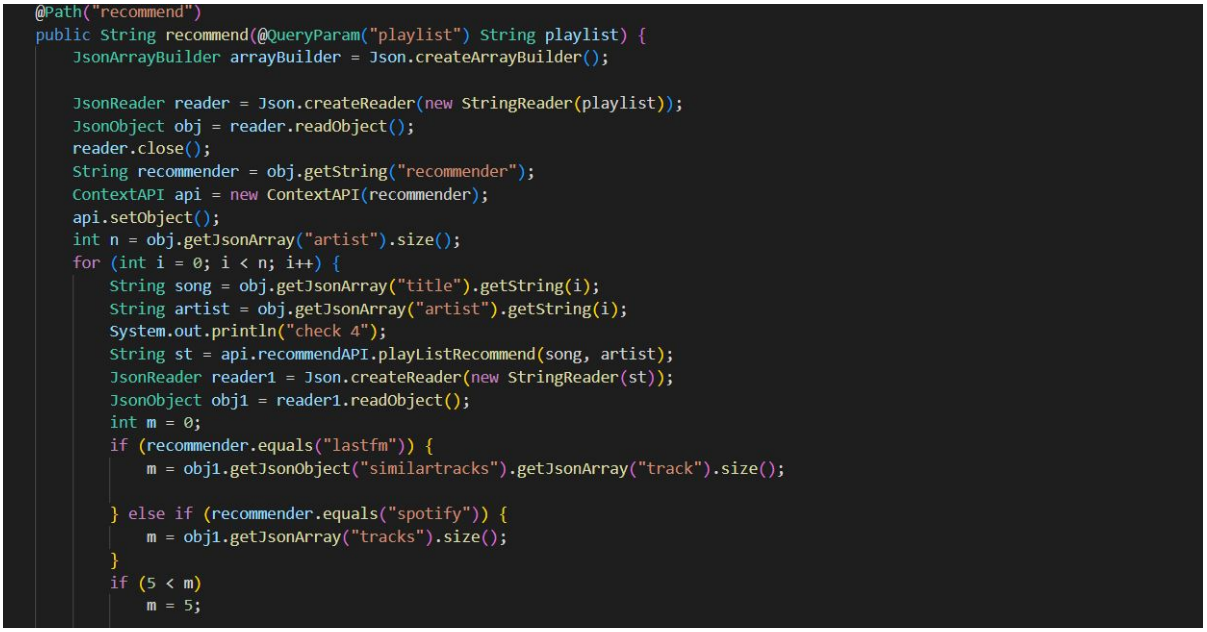
We now create a context class named ContextAPI



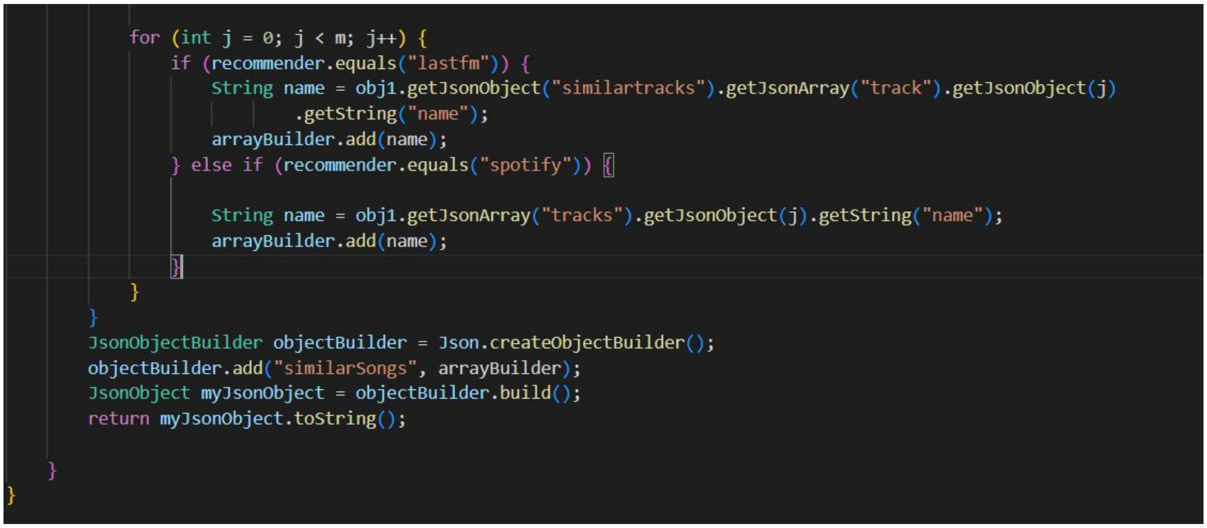
```
public class ContextAPI {
    public String method;
    public RecommendAPI recommendAPI;

    ContextAPI(String st) {
        //System.out.println("asd");
        this.method = st;
    }

    public void setObject() {
        //System.out.println("inside set object");
        if (method.equals(anObject:"lastfm")) {
            //System.out.println("before assigning");
            this.recommendAPI = new LastFmService();
            //System.out.println("after assigning");
        } else if (method.equals( anObject:"spotify"))
            this.recommendAPI=new SpotifyService();
    }
}
```

The above class has a method named setObject which sets the RecommendAPI based on the string passed in the constructor (lastfm or Spotify).

So, when the backend receives data, it RecommendAPI based on the service i.e spotify or lastfm.



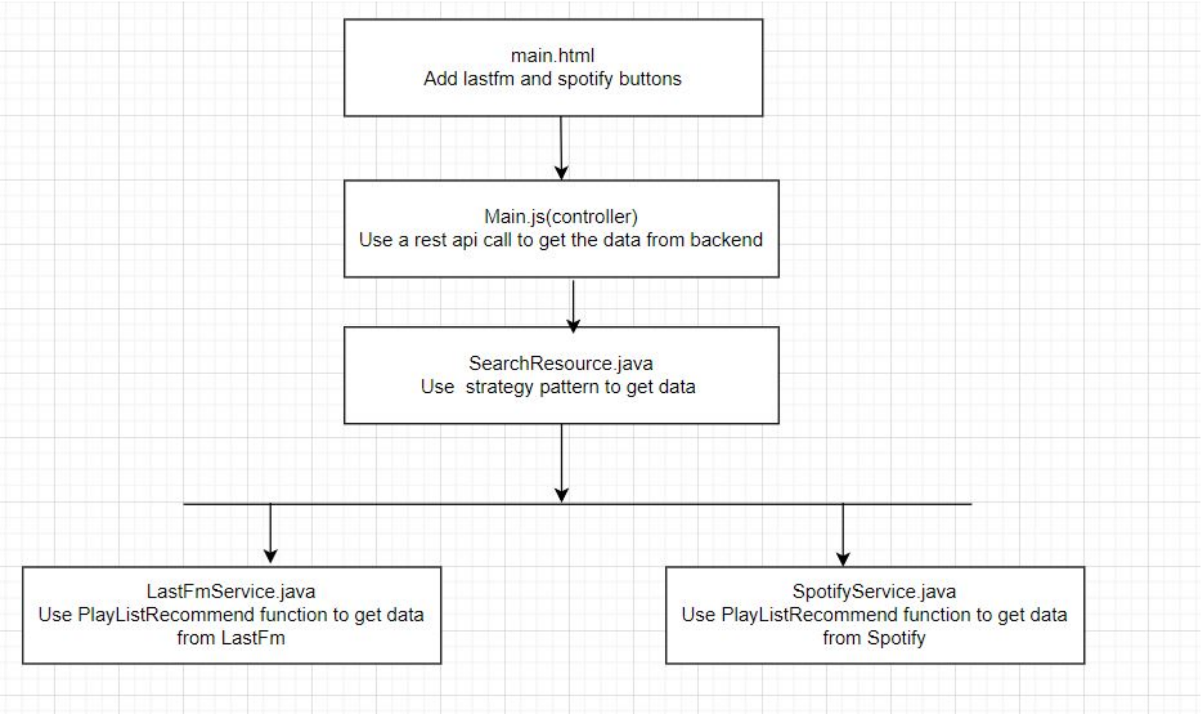
```
    @Path("recommend")
    public String recommend(@QueryParam("playlist") String playlist) {
        JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

        JsonReader reader = Json.createReader(new StringReader(playlist));
        JsonObject obj = reader.readObject();
        reader.close();
        String recommender = obj.getString("recommender");
        ContextAPI api = new ContextAPI(recommender);
        api.setObject();
        int n = obj.getJsonArray("artist").size();
        for (int i = 0; i < n; i++) {
            String song = obj.getJsonArray("title").getString(i);
            String artist = obj.getJsonArray("artist").getString(i);
            System.out.println("check 4");
            String st = api.recommendAPI.playListRecommend(song, artist);
            JsonReader reader1 = Json.createReader(new StringReader(st));
            JsonObject obj1 = reader1.readObject();
            int m = 0;
            if (recommender.equals("lastfm")) {
                m = obj1.getJsonObject("similartracks").getJsonArray("track").size();

            } else if (recommender.equals("spotify")) {
                m = obj1.getJsonArray("tracks").size();
            }
            if (5 < m)
                m = 5;
```



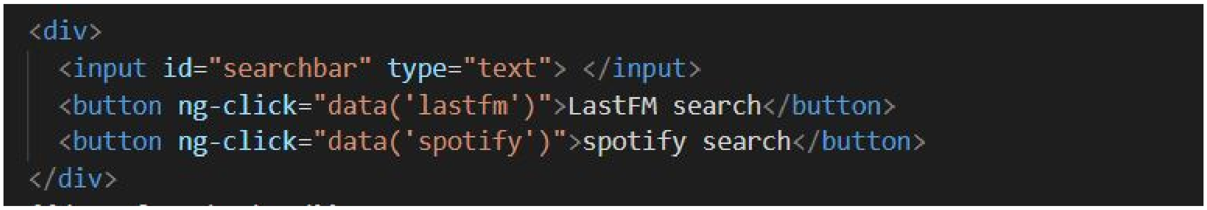
```
            for (int j = 0; j < m; j++) {
                if (recommender.equals("lastfm")) {
                    String name = obj1.getJsonObject("similartracks").getJsonArray("track").getJsonObject(j)
                            .getString("name");
                    arrayBuilder.add(name);
                } else if (recommender.equals("spotify")) {

                    String name = obj1.getJsonArray("tracks").getJsonObject(j).getString("name");
                    arrayBuilder.add(name);
                }
            }
        }
        JsonObjectBuilder objectBuilder = Json.createObjectBuilder();
        objectBuilder.add("similarSongs", arrayBuilder);
        JsonObject myJsonObject = objectBuilder.build();
        return myJsonObject.toString();

    }
}
```

Also when searching for a song using lasfm API or spotify api.

This is the flow in which we make changes for Search function

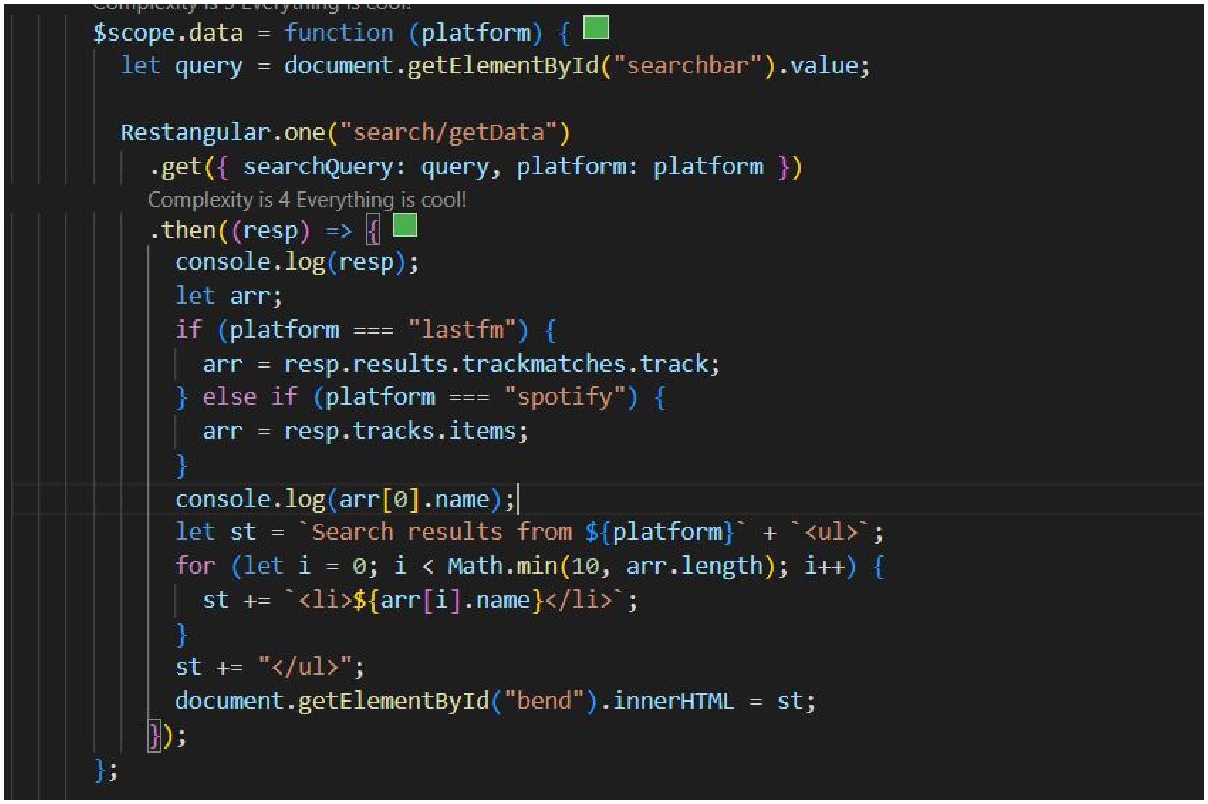

main.html
Add lastfm and spotify buttons
Main.js(controller)
Use a rest api call to get the data from backend
SearchResource.java
Use strategy pattern to get data
LastFmService.java
Use PlayListRecommend function to get data from LastFm
SpotifyService.java
Use PlayListRecommend function to get data from Spotify

main.html



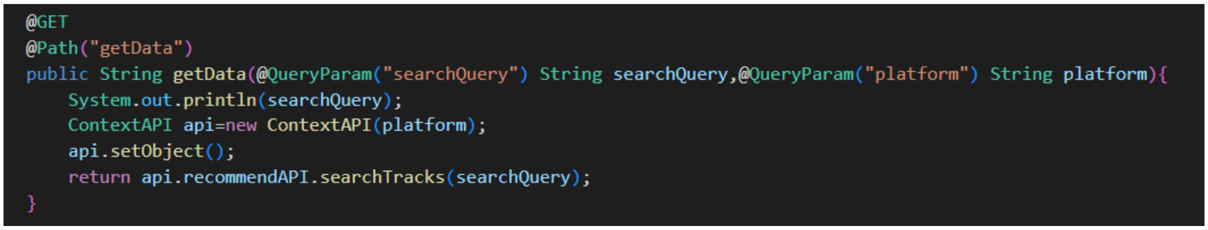
```
<div>
  <input id="searchbar" type="text"> </input>
  <button ng-click="data('lastfm')">LastFM search</button>
  <button ng-click="data('spotify')">spotify search</button>
</div>
```

main.js



```
Complexity is 3 Everything is cool!
$scope.data = function (platform) {
  let query = document.getElementById("searchbar").value;

  Restangular.one("search/getData")
    .get({ searchQuery: query, platform: platform })
    Complexity is 4 Everything is cool!
    .then((resp) => {
      console.log(resp);
      let arr;
      if (platform === "lastfm") {
        arr = resp.results.trackmatches.track;
      } else if (platform === "spotify") {
        arr = resp.tracks.items;
      }
      console.log(arr[0].name);
      let st = `Search results from ${platform}` + `<ul>`;
      for (let i = 0; i < Math.min(10, arr.length); i++) {
        st += `<li>${arr[i].name}</li>`;
      }
      st += "</ul>";
      document.getElementById("bend").innerHTML = st;
    });
};
```

SearchResource.java

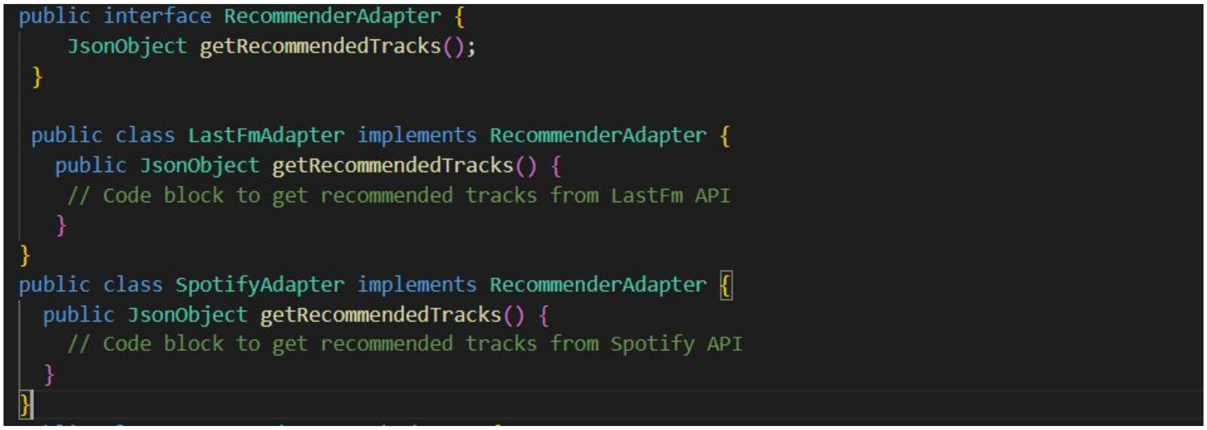


```
@GET
@Path("getData")
public String getData(@QueryParam("searchQuery") String searchQuery,@QueryParam("platform") String platform){
    System.out.println(searchQuery);
    ContextAPI api=new ContextAPI(platform);
    api.setObject();
    return api.recommendAPI.searchTracks(searchQuery);
}
```

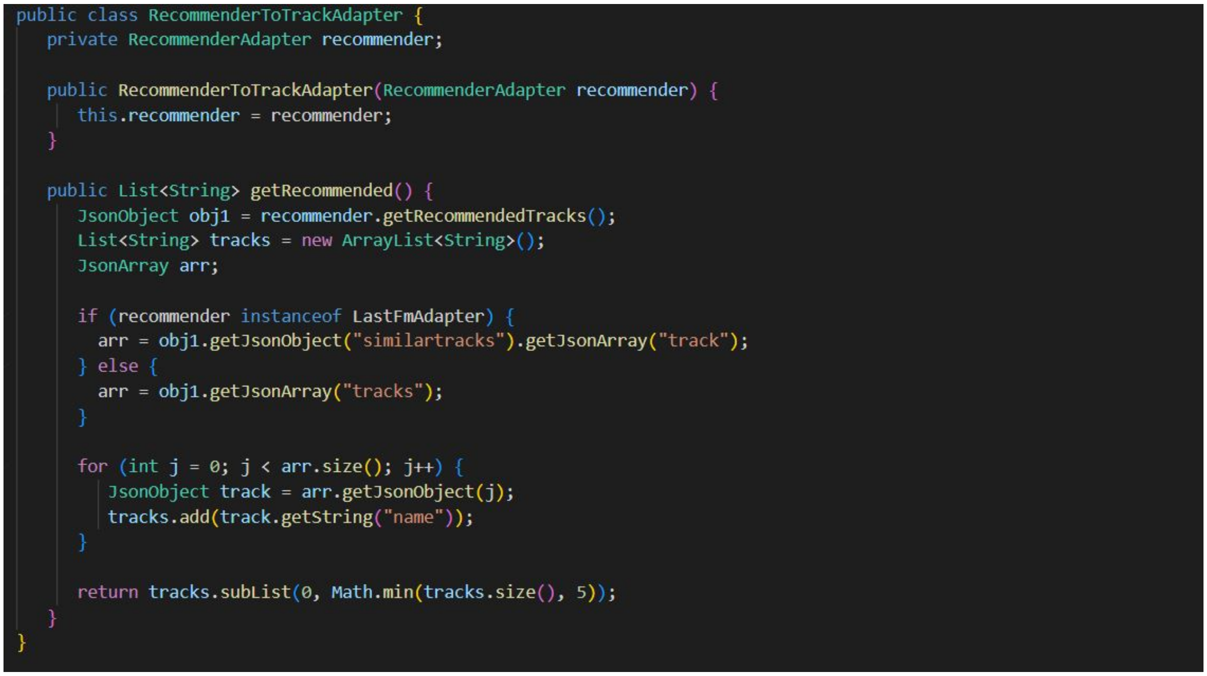
Adapter pattern: Our team has encountered a challenge with the data received from the LastFm and Spotify APIs. Specifically, the data we receive from one API differs from the other, which has made it difficult for us to consolidate the data in a meaningful and efficient way.

To address this issue, we have decided to create a custom class that will serve as an intermediary between the two APIs. This class will be responsible for converting the data we receive from both APIs into a single, uniform format that can be processed easily by our frontend system. Essentially, this will allow us to render the data in a consistent and user-friendly manner, without having to worry about which recommender it came from.

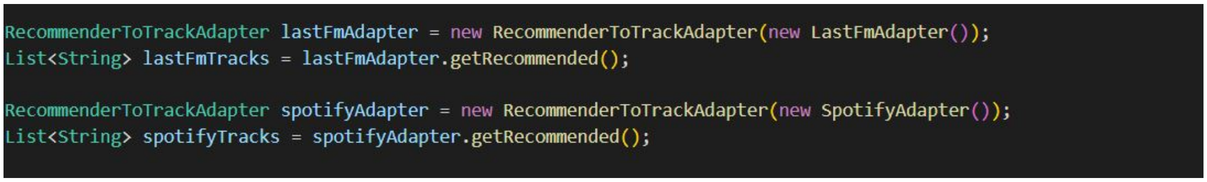
By developing such a class, we aim to make our recommendation system more robust and reliable. Additionally, it will enable us to better leverage the respective strengths of the LastFm and Spotify APIs, without having to worry about differences or incompatibilities between them. Overall, we are confident that this approach will ultimately lead to a more effective and user-friendly recommendation system for our customers.



```
public interface RecommenderAdapter {
    JsonObject getRecommendedTracks();
 }

 public class LastFmAdapter implements RecommenderAdapter {
   public JsonObject getRecommendedTracks() {
    // Code block to get recommended tracks from LastFm API
   }
}
public class SpotifyAdapter implements RecommenderAdapter {
  public JsonObject getRecommendedTracks() {
    // Code block to get recommended tracks from Spotify API
  }
}
```



```
public class RecommenderToTrackAdapter {
   private RecommenderAdapter recommender;

   public RecommenderToTrackAdapter(RecommenderAdapter recommender) {
      this.recommender = recommender;
   }

   public List<String> getRecommended() {
      JsonObject obj1 = recommender.getRecommendedTracks();
      List<String> tracks = new ArrayList<String>();
      JsonArray arr;

      if (recommender instanceof LastFmAdapter) {
        arr = obj1.getJsonObject("similartracks").getJsonArray("track");
      } else {
        arr = obj1.getJsonArray("tracks");
      }

      for (int j = 0; j < arr.size(); j++) {
         JsonObject track = arr.getJsonObject(j);
         tracks.add(track.getString("name"));
      }

      return tracks.subList(0, Math.min(tracks.size(), 5));
   }
}
```



```
RecommenderToTrackAdapter lastFmAdapter = new RecommenderToTrackAdapter(new LastFmAdapter());
List<String> lastFmTracks = lastFmAdapter.getRecommended();

RecommenderToTrackAdapter spotifyAdapter = new RecommenderToTrackAdapter(new SpotifyAdapter());
List<String> spotifyTracks = spotifyAdapter.getRecommended();
```