SISMICS MUSIC PROJECT -2

SOFTWARE ENGINEERING

TEAM 22:
Dishant Sharma - 2022202019
Harshit Kashyap - 2022201050
Shubham Deshmukh - 2022201076
Udrasht Pal - 2022201020
Utkarsh Pathak - 2022201018

TASK 1 - USER MANAGEMENT

LIMITATIONS:

Earlier the User Management System of the Sismic Music was handled solely by the administrator, manually which includes addition, deletion, changing of credentials of the users, etc.

IMPROVEMENT:

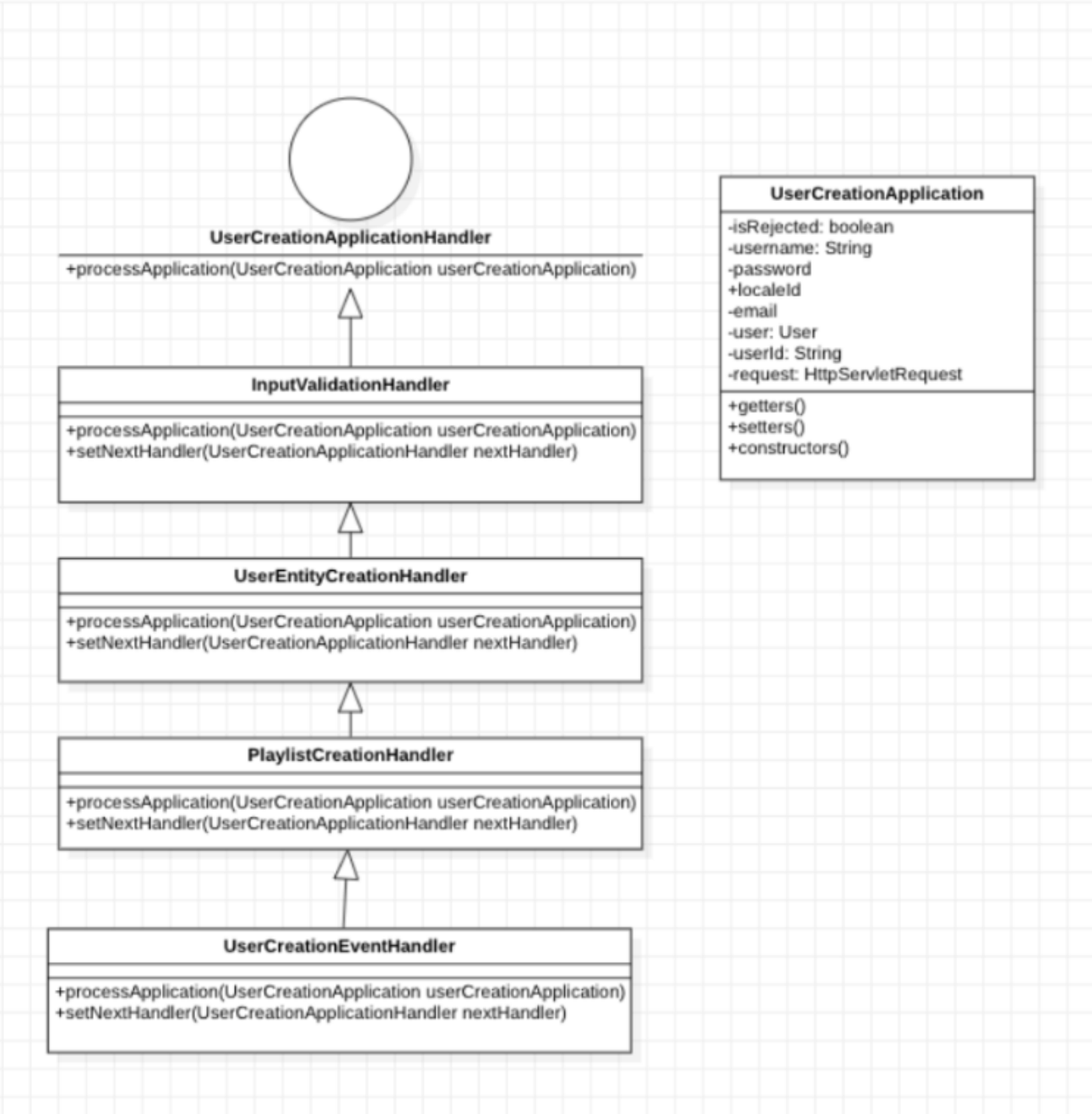
To enhance user management we implemented self-Registration of Users. It allows individual users to register and create accounts for themselves directly from the login page. This feature removes the dependency of users on the Admin to create their account and give access to the System.

DESIGN PATTERN - CHAIN OF RESPONSIBILITY:

The Chain of Responsibility is a behavioral design pattern that allows you to pass requests along a chain of handlers, each of which handles the request or passes it on to the next handler in the chain. It decouples the sender of a request from its receivers, allowing multiple objects to handle the request.

In this pattern, each handler has a reference to the next handler in the chain. When a request is received, the current handler determines whether it can handle it. If it can, it handles the request, and the processing ends. If it can't, it passes the request on to the next handler in the chain. We can link the handlers in any order but all the requests would follow the pattern as intended.

UML DIAGRAM OF IMPLEMENTATION



IMPLEMENTATION LOGIC

To create a user there are multiple steps involved. The steps are :

1. Validation of Input (such as username, email, and password)
2. Creation of User Object and setting the attributes
3. Creation of default playlist
4. Triggering an event and saving the object in the database.

As we can see that there are multiple actions(checks and functionality) involved and they are followed by one after the another.

DIRECTORY STRUCTURE

com.sismics.music.usercreation
->InputValidationHandler.java
->PlaylistCreationHandler.java
->UserCreationApplication.java
->UserCreationApplicationHandler.java
->UserCreationEventHandler.java
->UserEntityCreationHandler.java

IMPLEMENTATION DESCRIPTION

We created multiple handlers for performing each of the tasks:

InputValidationHandler.java

This handler checks whether the form submitted by the user for account creation is valid or not.

UserEntityCreationHandler.java

This creates a user object, sets the attributes and checks in the database whether the user exists in the database or not, and then sends the application to the next handler(i.e., PlaylistCreationHandler.java)

PlaylistCreationHandler.java

This handler creates a playlist object and creates a default playlist for the user and stores the playlist in the database.

UserCreationEventHandler.java

This handler creates an instance of AppContext and triggers the creation of a user.

CODE SNIPPET

InputValidationHandler

```
ublic class InputValidationHandler implements UserCreationApplicationHandler{

 private UserCreationApplicationHandler nextHandler;

 public void setNextHandler(UserCreationApplicationHandler nextHandler) {
 this.nextHandler = nextHandler;
 }
 @Override
 public void processApplication(UserCreationApplication
userCreationApplication) {

 String username = userCreationApplication.getUsername();
 String password = userCreationApplication.getPassword();
 String localeId = userCreationApplication.getLocaleId();
 String email = userCreationApplication.getEmail();

 username = Validation.length(username, "username", 3, 50);
 Validation.alphanumeric(username, "username");
 password = Validation.length(password, "password", 8, 50);
 email = Validation.length(email, "email", 3, 50);
 Validation.email(email, "email");

 userCreationApplication.setUsername(username);
 userCreationApplication.setPassword(password);
 userCreationApplication.setLocaleId(localeId);
 userCreationApplication.setEmail(email);

 if(nextHandler != null) {
 nextHandler.processApplication(userCreationApplication);
 }
 }

}
```

CODE SNIPPET

PlaylistCreationHandler.java

```
public class PlaylistCreationHandler implements UserCreationApplicationHandler
{

 private UserCreationApplicationHandler nextHandler;
 public void setNextHandler(UserCreationApplicationHandler nextHandler) {
 this.nextHandler = nextHandler;
 }
 @Override
 public void processApplication(UserCreationApplication
userCreationApplication) {

 Playlist playlist = new Playlist();
 playlist.setUserId(userCreationApplication.getUserId());
 Playlist.createPlaylist(playlist);

 if(nextHandler != null) {
 nextHandler.processApplication(userCreationApplication);
 }

 }

}
```

CODE SNIPPET

UserCreationEventHandler..java

```
public class UserCreationEventHandler implements UserCreationApplicationHandler
{

 @Override
 public void processApplication(UserCreationApplication
userCreationApplication) {

  UserCreatedEvent userCreatedEvent = new UserCreatedEvent();
  userCreatedEvent.setUser(userCreationApplication.getUser());
  AppContext.getInstance().getAsyncEventBus().post(userCreatedEvent);
 }
}
```

CODE SNIPPET

UserEntityCreationHandler..java

```
UserCreationApplicationHandler {

 private UserCreationApplicationHandler nextHandler;
 public void setNextHandler(UserCreationApplicationHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

 @Override
 public void processApplication(UserCreationApplication
userCreationApplication) {

  User user = userCreationApplication.getUser();

  user.setRoleId(Constants.DEFAULT_USER_ROLE);
        user.setUsername(username);
        user.setPassword(password);
        user.setEmail(email);
        user.setCreateDate(new Date());

        if (localeId == null) {
            // Set the locale from the HTTP headers
            localeId =
LocaleUtil.getLocaleIdFromAcceptLanguage(request.getHeader("Accept-Language"));
        }
        user.setLocaleId(localeId);

        // Create the user
        UserDao userDao = new UserDao();

        try {
         userCreationApplication.setUserId(userDao.create(user));
        } catch (Exception e) {
            if ("AlreadyExistingUsername".equals(e.getMessage())) {
                throw new ServerException("AlreadyExistingUsername", "Login
already used", e);
            } else {
                throw new ServerException("UnknownError", "Unknown Server
Error", e);
            }
        }

        if(nextHandler != null) {
   nextHandler.processApplication(userCreationApplication);
  }
 }

}
```

CODE SNIPPET

UserCreationApplication..java

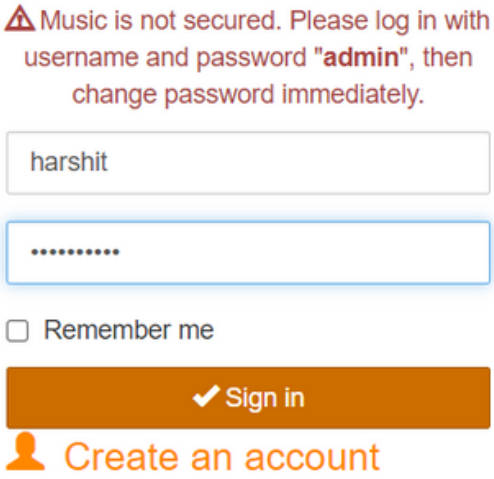
```
public class UserCreationApplication {
 private boolean isRejected;
 private String username;
 private String password;
 private String localeId;
 private String email;
 private User user;
 private String userId;

getters()
setters()
constructors()
```

UserCreationApplicationHandler.java

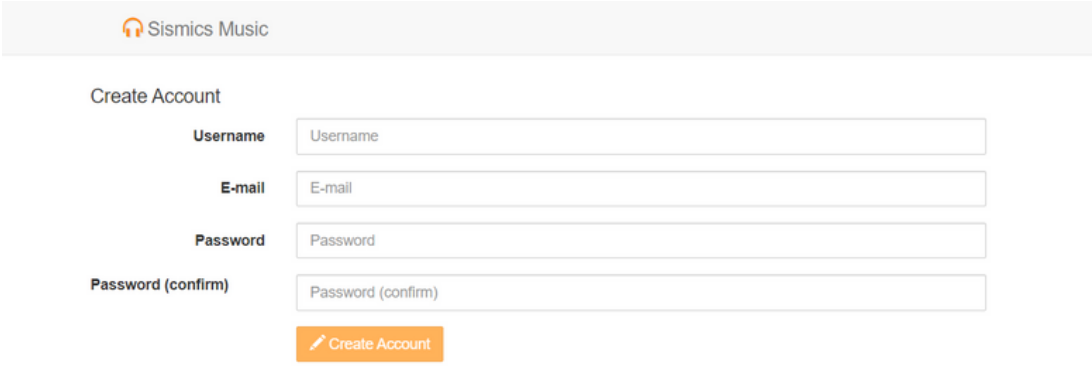
```
package com.sismics.music.usercreation;

public interface UserCreationApplicationHandler {
 void processApplication(UserCreationApplication userCreationApplication);
}


```

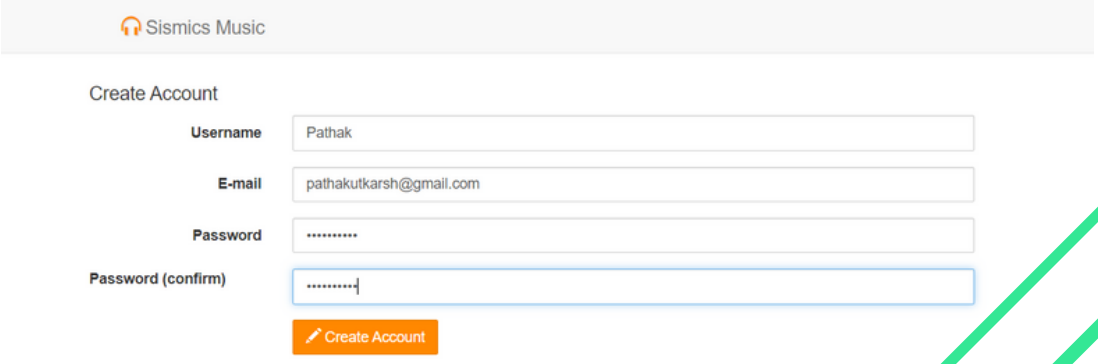
UI SCREENSHOTS

Account Creation Tab on Login Page



Account Creation Form





TASK 2 - LIBRARY MANAGEMENT

INRODUCTION

The users have the privilege to add songs to the system, like, play them. The users can create playlist , add music to playlist and update update playlist.
Users can create multiple playlists according to their choice and they can even upload multiple music tracks.
The user can view playlists and play songs in the playlists. Music Library allows to like the songs , view the play count. When the songs are being played it shows the music visualizer making the interface more appealing for the users.

LIMITATIONS

By default all the songs uploaded by the users are public i.e. all the users could view and play the songs uploaded by other and themselves. But this does not seem to be correct because it does keep privacy of the user with respect to songs.

By Default all the playlists created by the users are private i.e. only the creator of playlist can only view the playlist.

TASK 2.1 - PRIVATE MUSIC

LIMITATIONS

By default all the songs uploaded by the users are public i.e. all the users could view and play the songs uploaded by other and themselves. But this does not seem to be correct because it does keep privacy of the user with respect to songs.

IMPROVEMENT:

So as to resolve the limitation of songs being public is resolved by making the songs private for the user who has uploaded the them.

Now the songs will be only available for the original users.

IMPLEMENTATION LOGIC

For making the music private we added one column user_id in the album, track, and artist tables. This column stores the user_id of the owner(uploader). This column is utilized when we load the albums and artists on the main landing page. Therefore the user who uploads the music can see only their uploads making themusic provate

IMPLEMENTATION DESCRIPTION

We created multiple handlers for performing each of the tasks:

Album.java and Artist.java
Created an attribute userId: String in the above classes.

AlbumDtoMapper.java and ArtistDtoMapper.java
Added the constraint of user id in both the DTO classes.

AlbumMapper.java and ArtistMapper.java
Added the column of user_id and created the constructor according to the new attribute

AlbumCriteria.java and ArtistCriteria.java
Created a getter and setter function for the new attribute user_id

CollectionService.java
Initializing the user id of the album, artist and track when a user uploads the track

AlbumDao.java and ArtistDao.java
Added the check of user_id in the where clause while fetching the albums or artists based on the user_id

AlbumResource.java and ArtistResource.java
Added a criteria for the search query which will be passed to the DAO layer for query building and fetching the result.

CODE SNIPPET

AlbumResource

```
1 @Path("/album")
2 public class AlbumResource extends BaseResource {
3     /**
4      * Logger.
5      */
6     private static final Logger log =
  LoggerFactory.getLogger(AlbumResource.class);
7     @GET
8     public Response list(
9             @QueryParam("limit") Integer limit,
10            @QueryParam("offset") Integer offset,
11            @QueryParam("sort_column") Integer sortColumn,
12            @QueryParam("asc") Boolean asc,
13            @QueryParam("search") String search) {
14        if (!authenticate()) {
15            throw new ForbiddenClientException();
16        }
17        System.out.println(principal.getId()+" - "+principal.getName());
18        AlbumDao albumDao = new AlbumDao();
19        PaginatedList<AlbumDto> paginatedList = PaginatedLists.create(limit,
  offset);
20        SortCriteria sortCriteria = new SortCriteria(sortColumn, asc);
21        System.out.println(principal.getId()+" - "+ principal.getName());
22        AlbumCriteria albumCriteria = new AlbumCriteria()
23                .setNameLike(search)
24                .setUserId(principal.getId());
25        albumDao.findByCriteria(paginatedList, albumCriteria, sortCriteria, null);
26
27        JsonObjectBuilder response = Json.createObjectBuilder();
28        JsonArrayBuilder items = Json.createArrayBuilder();
29        System.out.println("Album Resource GET list");
30        //building the JSON
31
32
33        return renderJson(response);
34    }
35
```

CODE SNIPPET

ArtistResource

```
1  @Path("/artist")
2  public class ArtistResource extends BaseResource {
3
4      @GET
5      @Path("{id: [a-z0-9\\-]+}")
6      public Response get(
7              @PathParam("id") String id) {
8          if (!authenticate()) {
9              throw new ForbiddenClientException();
10         }
11
12         //setting other crterias
13
14         // Artist's albums
15         AlbumDao albumDao = new AlbumDao();
16         List<AlbumDto> albumList = albumDao.findByCriteria(new AlbumCriteria()
17                 .setUserId(principal.getId())
18                 .setArtistId(artist.getId()));
19
20        //building JSON
21         response.add("albums", albums);
22
23         // Artist's tracks
24         TrackDao trackDao = new TrackDao();
25         List<TrackDto> trackList = trackDao.findByCriteria(new TrackCriteria()
26                 .setUserId(principal.getId())
27                 .setArtistId(artist.getId()));
28
29         JsonArrayBuilder tracks = Json.createArrayBuilder();
30         //building json
31         response.add("tracks", tracks);
32
33         return renderJson(response);
34     }
35
```

CODE SNIPPETS

AlbumDtoMapper

```
1 public class AlbumDtoMapper implements ResultSetMapper<AlbumDto> {
2     @Override
3     public AlbumDto map(int index, ResultSet r, StatementContext ctx) throws
  SQLException {
4         AlbumDto dto = new AlbumDto();
5         dto.setId(r.getString("id"));
6         dto.setName(r.getString("c0"));
7         dto.setUserId(r.getString("user_id"));
8         dto.setAlbumArt(r.getString("albumArt"));
9         dto.setArtistId(r.getString("artistId"));
10        dto.setArtistName(r.getString("artistName"));
11        dto.setUpdateDate(r.getTimestamp("c1"));
12        dto.setUserPlayCount(r.getLong("c2"));
13        return dto;
14    }
15 }
16
```

ArtistDtoMapper

```
1
2 public class ArtistDtoMapper implements ResultSetMapper<ArtistDto> {
3     @Override
4     public ArtistDto map(int index, ResultSet r, StatementContext ctx) throws
  SQLException {
5         ArtistDto dto = new ArtistDto();
6         dto.setId(r.getString("id"));
7         dto.setName(r.getString("c0"));
8         dto.setUserId(r.getString("user_id"));
9         return dto;
10    }
11 }
12
```

CODE SNIPPETS

AlbumMapper

```java
public class AlbumMapper extends BaseResultSetMapper<Album> {
    public String[] getColumns() {
        return new String[] {
                "id",
                "directory_id",
                "artist_id",
                "name",
                "albumart",
                "updatedate",
                "createdate",
                "deletedate",
                "location",
                "user_id",};
    }

    @Override
    public Album map(int index, ResultSet r, StatementContext ctx) throws SQLException {
        final String[] columns = getColumns();
        int column = 0;
        return new Album(
                r.getString(columns[column++]),
                r.getString(columns[column++]),
                r.getString(columns[column++]),
                r.getString(columns[column++]),
                r.getString(columns[column++]),
                r.getTimestamp(columns[column++]),
                r.getTimestamp(columns[column++]),
                r.getTimestamp(columns[column++]),
                r.getString(columns[column++]),
                r.getString(columns[column]));
    }
}
```

ArtistMapper

```java
public class ArtistMapper extends BaseResultSetMapper<Artist> {
    public String[] getColumns() {
        return new String[] {
                "id",
                "name",
                "namecorrected",
                "createdate",
                "deletedate",
                "user_id"};
    }
    @Override
    public Artist map(int index, ResultSet r, StatementContext ctx) throws SQLException {
        final String[] columns = getColumns();
        int column = 0;
        return new Artist(
            r.getString(columns[column++]),
            r.getString(columns[column++]),
            r.getString(columns[column++]),
            r.getTimestamp(columns[column++]),
            r.getTimestamp(columns[column++]),
            r.getString(columns[column])
            );
    }
}
```

CODE SNIPPETS

AlbumDao

```
1
2 public class AlbumDao extends BaseDao<AlbumDto, AlbumCriteria> {
3     @Override
4     public QueryParam getQueryParam(AlbumCriteria criteria, FilterCriteria
  filterCriteria) {
5
6         StringBuilder sb = new StringBuilder("select a.id as id, a.user_id as
  user_id, a.name as c0, a.albumart as albumArt, a.artist_id as artistId, ar.name as
  artistName, a.updatedate as c1, ");
7
8         //other criterias
9         criteriaList.add("a.user_id =:userId");
10        parameterMap.put("userId", criteria.getUserId());
11        //other criterias
12
13        return new QueryParam(sb.toString(), criteriaList, parameterMap, null,
  filterCriteria, Lists.newArrayList("a.id"), new AlbumDtoMapper());
14    }
15
16    public String create(Album album) {
17        handle.createStatement("insert into " +
18                " t_album(id, user_id, directory_id, artist_id, name, albumart,
  createdate, updatedate, location)" +
19                " values(:id, :userId, :directoryId, :artistId, :name, :albumArt,
  :createDate, :updateDate, :location)")
20                .bind("id", album.getId())
21                .bind("userId", album.getUserId())
22                .bind("directoryId", album.getDirectoryId())
23                .bind("artistId", album.getArtistId())
24                .bind("name", album.getName())
25                .bind("albumArt", album.getAlbumArt())
26                .bind("updateDate", new
  Timestamp(album.getUpdateDate().getTime()))
27                .bind("createDate", new
  Timestamp(album.getCreateDate().getTime()))
28                .bind("location", album.getLocation())
29                .execute();
30
31        return album.getId();
32    }
33
34
```

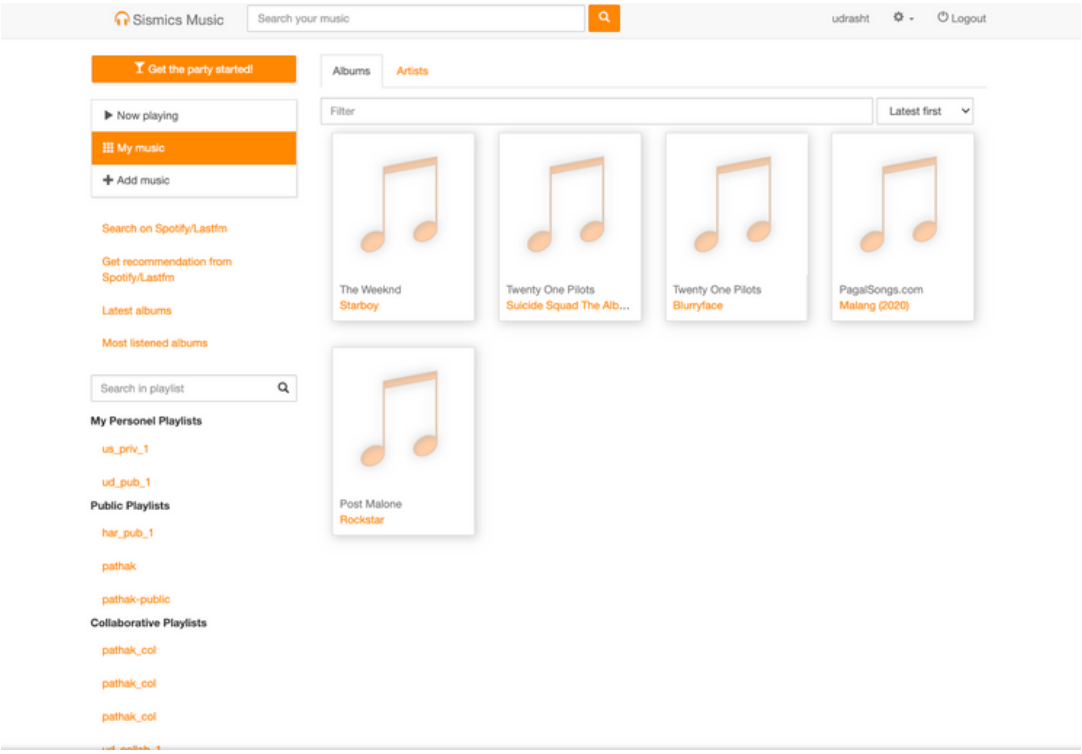
CODE SNIPPETS

ArtistDao

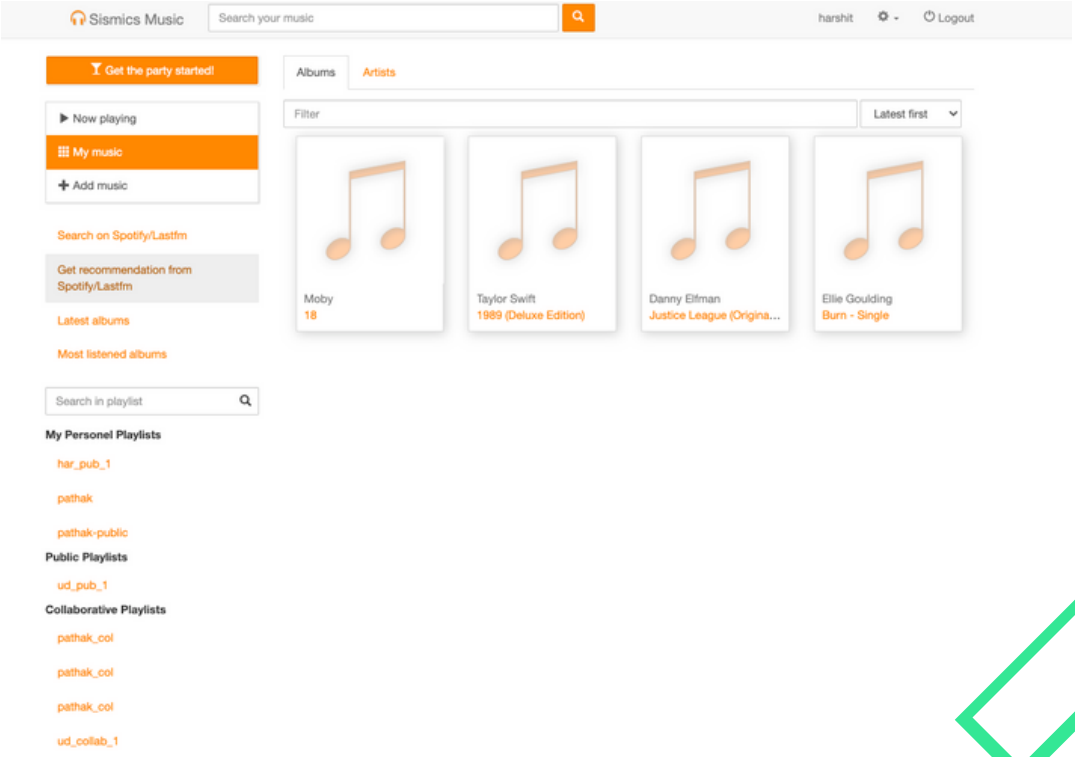
```
1
2 public class ArtistDao extends BaseDao<ArtistDto, ArtistCriteria> {
3     @Override
4     protected QueryParam getQueryParam(ArtistCriteria criteria, FilterCriteria
  filterCriteria) {
5         StringBuilder sb = new StringBuilder("select a.id as id, a.user_id as
  user_id, a.name as c0 ");
6         System.out.println("adding where clause for user id");
7         criteriaList.add("a.user_id =:userId");
8         parameterMap.put("userId", criteria.getUserId());
9
10        //other criteria
11        return new QueryParam(sb.toString(), criteriaList, parameterMap, null,
  filterCriteria, new ArtistDtoMapper());
12    }
13
14    public String create(Artist artist) {
15      System.out.println("In ARTIST DAO : CREATING A NEW ARTIST");
16      System.out.println(artist.toString());
17        artist.setId(UUID.randomUUID().toString());
18        artist.setCreateDate(new Date());
19
20        final Handle handle = ThreadLocalContext.get().getHandle();
21        handle.createStatement("insert into " +
22                "  t_artist (id, user_id, name, namecorrected, createdate)" +
23                "  values(:id, :userId, :name, :nameCorrected, :createDate)")
24                .bind("id", artist.getId())
25                .bind("userId",artist.getUserId())
26                .bind("name", artist.getName())
27                .bind("nameCorrected", artist.getNameCorrected())
28                .bind("createDate", new
  Timestamp(artist.getCreateDate().getTime()))
29                .execute();
30
31        return artist.getId();
32    }
33
34
```

UI SCREENSHOTS

User 1 Albums



User 2 Albums



TASK 2.2 -PUBLIC PLAYLIST

LIMITATIONS

By Default all the playlists created by the users are private i.e. only the creator of playlist can only view the playlist.

IMPROVEMENT:

Library Management has improved which now allow the playlists to be public to all the users. While creating playlist the users can designate the playlist to be public or private.

DESIGN PATTERN - BUILDER + TEMPLATE

BUILDER

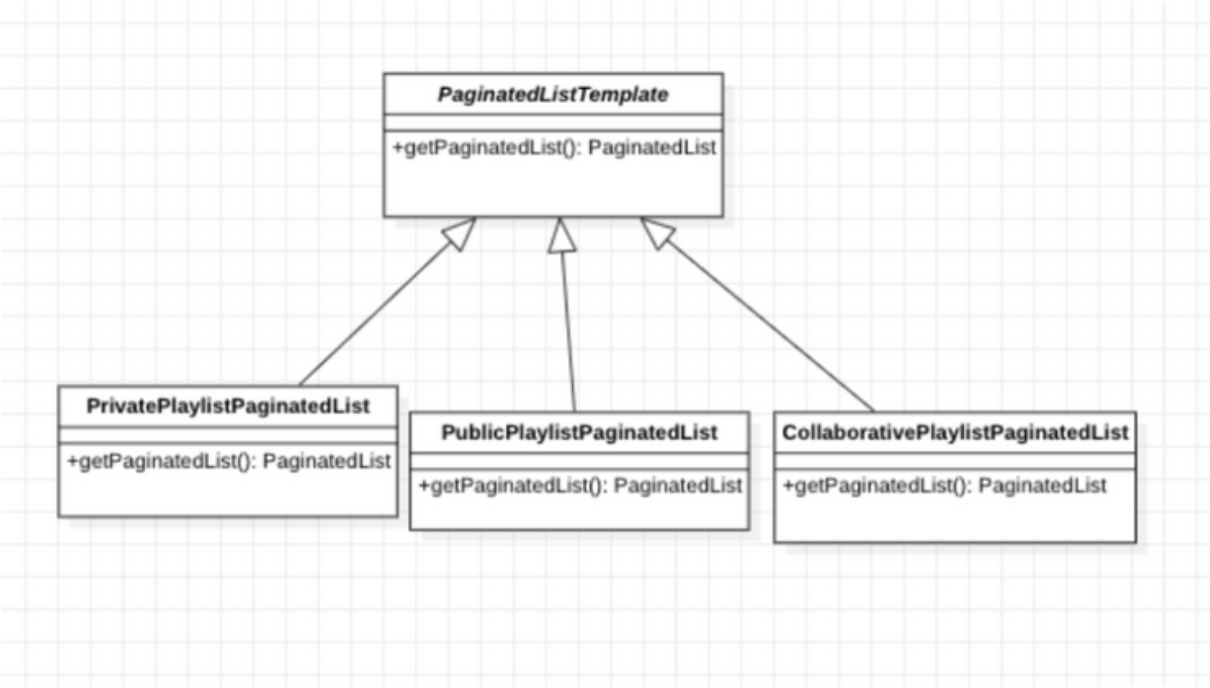
The builder design pattern is a creational design pattern that separates the construction of a complex object from its representation, allowing the same construction process to create different representations. It is useful while creating objects that have multiple parts, or while creating objects that require different initialization steps.
In this pattern, a builder class is responsible for creating the object, and a director class controls the order in which the builder's methods are called to create the object and you can reuse the same construction code when building various representations of products.

TEMPLATE

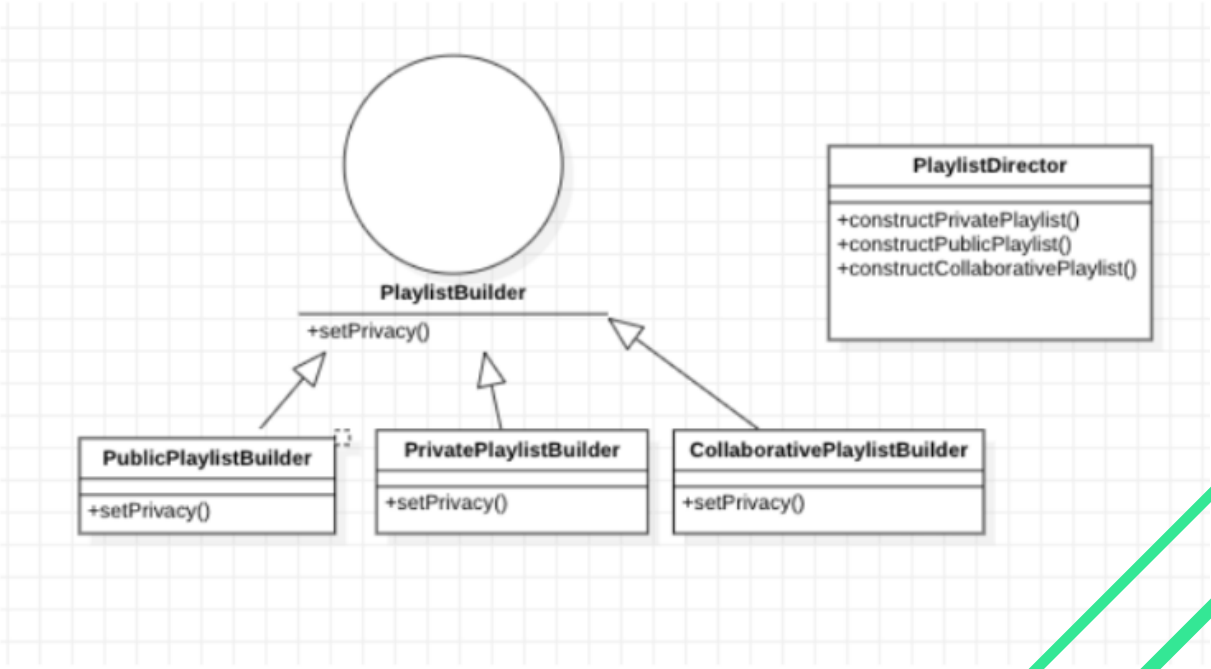
The Template Method is a behavioral design pattern that defines the skeleton of an algorithm in a superclass but allows subclasses to override specific steps of the algorithm without changing its structure.
In this pattern, the template method is defined in the base class and provides a series of steps that make up the algorithm. Each step may be either abstract or have a default implementation. Subclasses may then provide specific implementations of the abstract steps to create a complete implementation of the algorithm.
If we only want to use a few steps from the algorithm or we have several classes that contain almost identical algorithms with some minor differences then we can use this design pattern.

UML DIAGRAM OF IMPLEMENTATION

Template Method Design Pattern



Builder Design Pattern



IMPLEMENTATION LOGIC

For making the playlist public, we added one column of privacy in the t_playlist table. For this task, this field can have only public and private value. When a user creates a new playlist, the user is asked to either select a public or private playlist and the field are set accordingly in the database.

IMPLEMENTATION DESCRIPTION

Playlist.java
Created attribute privacy: String in the class

PlaylistDto.java
Created attribute privacy: String in the above classes.

PlaylistMapper.java
Added the column of privacy and created the constructor according to the new attribute

PlaylistCriteria.java
Created a getter and setter function for the new attribute privacy

PlaylistResource.java
Added criteria (public or private) for the search query which will be passed to the DAO layer for query building and fetching the result.

CODE SNIPPETS

PlaylistResource Template

```
@Path("/playlist")
public class PlaylistResource extends BaseResource {
  public static final String DEFAULt_playlist = "default";
  @GET
  public Response listPlaylist(
      @QueryParam("limit") Integer limit,
      @QueryParam("offset") Integer offset,
      @QueryParam("sort_column") Integer sortColumn,
      @QueryParam("asc") Boolean asc) {
    if (!authenticate()) {
      throw new ForbiddenClientException();
    }
    // Get the personel private playlists
    PaginatedListTemplate<PlaylistDto> privateTemplate = new PrivatePlaylistPaginatedList();
    PaginatedList<PlaylistDto> privatePlaylists = privateTemplate.getPaginatedList(limit, offset, sortColumn, asc, principal.getId());
    //bulding JSON

    // Getting public the playlists
    PaginatedListTemplate<PlaylistDto> publicTemplate = new PublicPlaylistPaginatedList();
    PaginatedList<PlaylistDto> publicPlaylists = publicTemplate.getPaginatedList(limit, offset, sortColumn, asc, principal.getId());
    //building JSON

    // Getting collaborative playlists
    PaginatedListTemplate<PlaylistDto> collaborativeTemplate = new CollaborativePlaylistPaginatedList();
        PaginatedList<PlaylistDto> collaborativePlaylists = collaborativeTemplate.getPaginatedList(limit, offset, sortColumn, asc, principal.getId());

    //building JSON
    return renderJson(response);
  }

```

PlaylistMapper

```

public class PlaylistMapper implements ResultSetMapper<PlaylistDto> {
    @Override
    public PlaylistDto map(int index, ResultSet r, StatementContext ctx) throws SQLException {
        PlaylistDto dto = new PlaylistDto();
        dto.setId(r.getString("id"));
        dto.setPrivacy(r.getString("privacy"));
        dto.setName(r.getString("c0"));
        dto.setUserId(r.getString("userId"));
        dto.setPlaylistTrackCount(r.getLong("c1"));
        dto.setUserTrackPlayCount(r.getLong("c2"));
        return dto;
    }
}

```

CODE SNIPPETS

PlaylistDao Builder

```java
@Path("/playlist")
public class PlaylistResource extends BaseResource {
  public static final String DEFAULt_playlist = "default";
  @PUT
  public Response createPlaylist(
      @FormParam("name") String name, @FormParam("privacy") String privacy) {
    if (!authenticate()) {
      throw new ForbiddenClientException();
    }

    Validation.required(name, "name");
    //usage of builder designer patter
    Playlist playlist = new Playlist();
    PlaylistDirector playlistDirector = new PlaylistDirector();
    if(privacy.equalsIgnoreCase("private")) {
      PrivatePlaylistBuilder privatePlaylistBuilder = new PrivatePlaylistBuilder();
      playlistDirector.constructPrivatePlaylist(privatePlaylistBuilder);
      playlist = privatePlaylistBuilder.getPlaylist();
    }else if(privacy.equalsIgnoreCase("public")){
      PublicPlaylistBuilder publicPlaylistBuilder = new PublicPlaylistBuilder();
      playlistDirector.constructPublicPlaylist(publicPlaylistBuilder);
      playlist = publicPlaylistBuilder.getPlaylist();
    }else if(privacy.equalsIgnoreCase("collaborative")) {
      CollaborativePlaylistBuilder collaborativePlaylistBuilder = new
CollaborativePlaylistBuilder();
      playlistDirector.constructCollaborativePlaylist(collaborativePlaylistBuilder);
      playlist = collaborativePlaylistBuilder.getPlaylist();
    }


    // Output the playlist
    return renderJson(Json.createObjectBuilder()
        .add("item", Json.createObjectBuilder()
            .add("id", playlist.getId())
            .add("name", playlist.getName())
            .add("trackCount", 0)
            .add("userTrackPlayCount", 0)
            .add("privacy", playlist.getPrivacy())
            .build()));
  }

```

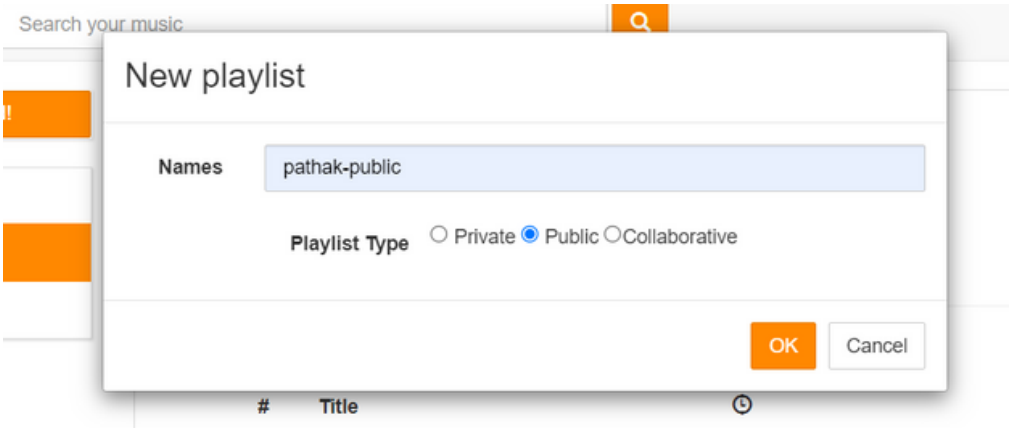
CODE SNIPPETS

PlaylistDao

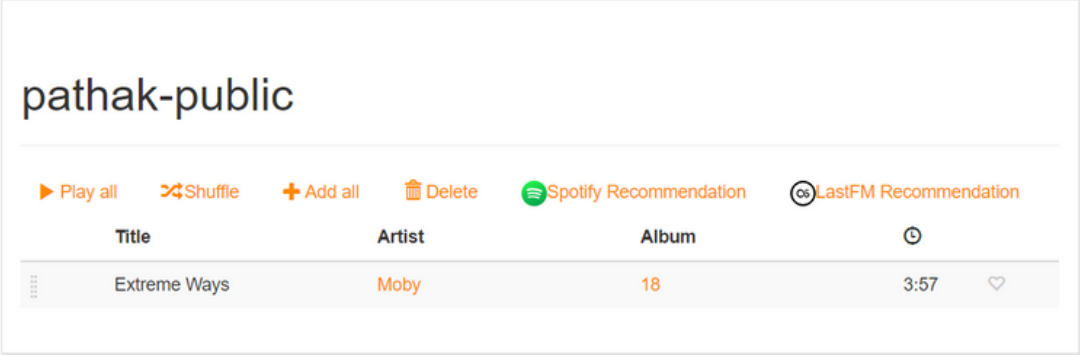
```java
public class PlaylistDao extends BaseDao<PlaylistDto, PlaylistCriteria> {
    @Override
    protected QueryParam getQueryParam(PlaylistCriteria criteria, FilterCriteria filterCriteria) {
        StringBuilder sb = new StringBuilder("select p.id as id, p.name as c0,")
                .append("  p.user_id as userId,")
                .append("  p.privacy as privacy,")
                .append("  count(pt.id) as c1,")
                .append("  sum(utr.playcount) as c2")
                .append("  from t_playlist p")
                .append("  left join t_playlist_track pt on(pt.playlist_id = p.id)")
                .append("  left join t_user_track utr on(utr.track_id = pt.track_id)");

        // 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());
        }
        //setting other criterias

        return new QueryParam(sb.toString(), criteriaList, parameterMap, null, filterCriteria, Lists.newArrayList("p.id"), new PlaylistMapper());
    }

    public String create(Playlist playlist) {

        final Handle handle = ThreadLocalContext.get().getHandle();
        handle.createStatement("insert into " +
                "  t_playlist(id, user_id, name, privacy)" +
                "  values(:id, :userId, :name, :privacy)")
                .bind("id", playlist.getId())
                .bind("userId", playlist.getUserId())
                .bind("name", playlist.getName())
                .bind("privacy", playlist.getPrivacy())
                .execute();

        return playlist.getId();
    }
```

UI SCREENSHOTS

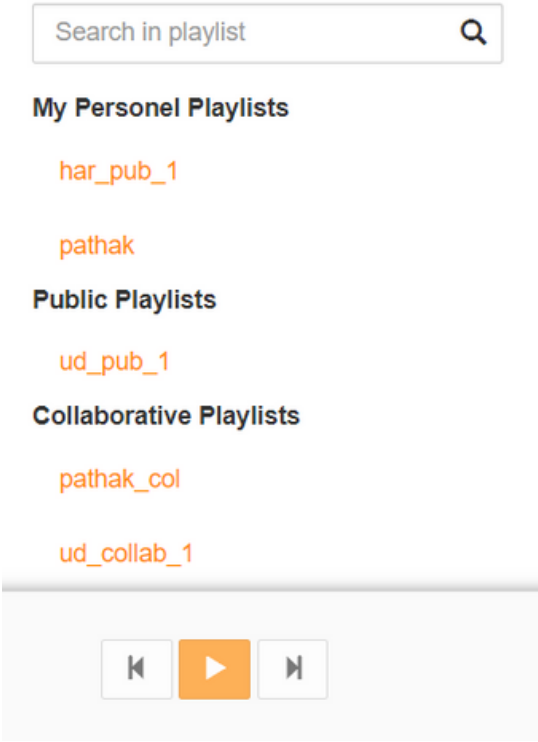
Playlist Creation Modal



Public Playlist Created



Listing all playlist



TASK 3 - ONLINE INTEGRATION

LIMITATIONS

Music system has online integration but it is only limited to only one service of lastFM where it gives only basic functionality for the users.

IMPROVEMENT

The Music system is integrated with the available lastFm and Spotify service for better library management.
The users have the privilege to search the songs and get recommendations from lastFM and spotify. While searching for the songs , the users can choose which service to use among the available two.

Music users to get recommendations from these services(can select any among the two) based on existing playlists.

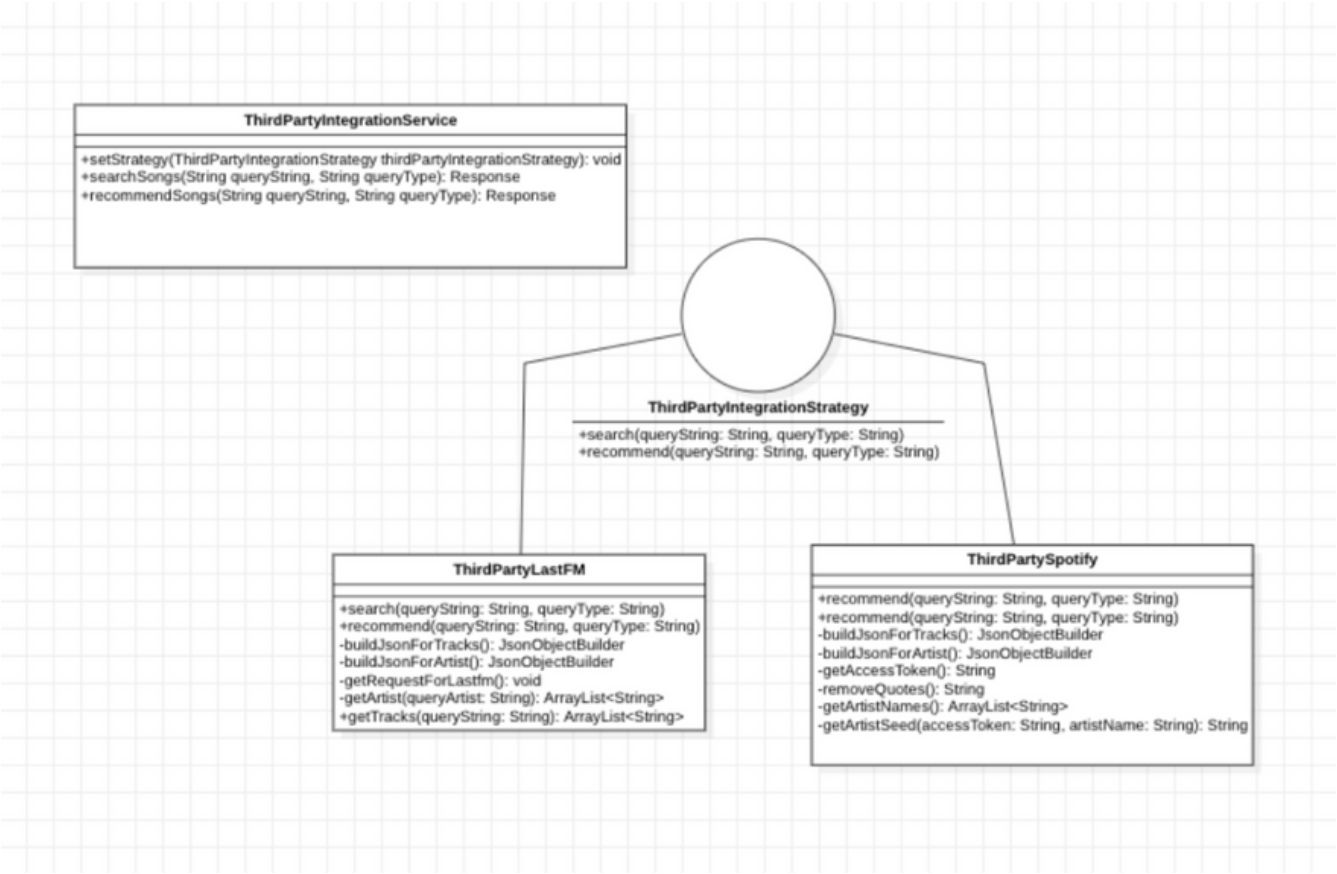
DESIGN PATTERN - STRATEGY

The Strategy pattern is a behavioral design pattern that enables you to define a family of algorithms, encapsulate each one as an object, and make them interchangeable. This pattern allows the algorithms to vary independently from the clients that use them, making it easy to add new algorithms or modify existing ones without changing the clients.

In this pattern, the strategy pattern defines a common interface for all the algorithms, allowing them to be used interchangeably. Each algorithm is encapsulated as a separate object that implements the common interface.

UML DIAGRAM OF IMPLEMENTATION

STRATEGY PATTERN



IMPLEMENTATION LOGIC

There are two different types of functionality search and recommendation using two different third-party services Spotify and LastFM. We created an interface ThirdPartyIntegrationStrategy which implemented two functions search and recommend. These functions take a query String and a query Type as an argument and either search or recommend based on the query. We created ThirdPartyIntegrationService class for setting the strategy and then returning the search or recommended songs bases on the function call.

DIRECTORY STRUCTURE

com.sismics.music.thirdpartyintegration
->ThirdPartyIntegrationStrategy.java
->ThirdPartyIntegrationStrategy.java
->ThirdPartyLastFM.java
->ThirdPartySpotify.java

IMPLEMENTATION DESCRIPTION

We created multiple handlers for performing each of the tasks:

ThirdPartyIntegrationService.java

This class has a function that sets the strategy based on the conditions. There are two other functions searchSongs() and recommendSongs() based on query string and query type.

ThirdPartyIntegrationStrategy.java

This is an interface that has two functions search and recommend.

ThirdPartyLastFM.java

This class has a search and recommend function implemented according to the LASTFM API. There are some helper functions for the search and recommend functions.

ThirdPartySpotify.java

This class has a search and recommend function implemented according to the SPOTIFY API. There are some helper functions for search and recommend functions.

CODE SNIPPETS

ThirdPartyIntegrationStrategy .java

```
public interface ThirdPartyIntegrationStrategy {
  public Response search(String queryString,String queryType) throws
IOException;
  public Response recommend(String queryString,String queryType)throws
IOException;
}
```

ThirdPartyIntegrationService

```
public class ThirdPartyIntegrationService
{
  private ThirdPartyIntegrationStrategy thirdPartyIntegrationStrategy;

  public void setStrategy(ThirdPartyIntegrationStrategy thirdPartyIntegrationStrategy) {
    this.thirdPartyIntegrationStrategy = thirdPartyIntegrationStrategy;
  }

  public Response searchSongs(String queryString,String queryType) throws IOException {
    return thirdPartyIntegrationStrategy.search(queryString,queryType);
  }

  public Response recommendSongs(String queryString,String queryType) throws IOException {
    return thirdPartyIntegrationStrategy.recommend(queryString,queryType);
  }
}
```

CODE SNIPPETS

SearchResource

```java

    @GET
    @Path("/search-third-party")
    public Response searchThirdParty(@QueryParam("thirdPartyType") String thirdPartyType,
        @QueryParam("queryType") String queryType,
        @QueryParam("queryString") String queryString
        ) throws IOException {
      ThirdPartyIntegrationService thirdpartyIntegrationService = new ThirdPartyIntegrationService();

      thirdpartyIntegrationService.setStrategy(new ThirdPartySpotify());
      if(thirdPartyType.equals("SPOTIFY")) {
        thirdpartyIntegrationService.setStrategy(new ThirdPartySpotify());
      }else if(thirdPartyType.equals("LASTFM")) {
        thirdpartyIntegrationService.setStrategy(new ThirdPartyLastFM());
      }
      Response response=thirdpartyIntegrationService.searchSongs(queryString,queryType);
        return response;
    }
    @GET
    @Path("/recommend-third-party")
    public Response recommendThirdParty(@QueryParam("thirdPartyType") String thirdPartyType,
        @QueryParam("queryType") String queryType,
        @QueryParam("queryString") String queryString) throws IOException {
      System.out.println(thirdPartyType);
      System.out.println(queryType);
      System.out.println(queryString);

      ThirdPartyIntegrationService thirdpartyIntegrationService = new ThirdPartyIntegrationService();
      if(thirdPartyType.equals("SPOTIFY")) {
        thirdpartyIntegrationService.setStrategy(new ThirdPartySpotify());
      }else if(thirdPartyType.equals("LASTFM")) {
        thirdpartyIntegrationService.setStrategy(new ThirdPartyLastFM());
      }
//      for last fm => queryString: artists and queryType: tracks
//      for spotify => queryString: artists and queryType : "seed_artists"
      Response response = thirdpartyIntegrationService.recommendSongs(queryString,queryType);
      return response;
    }

}
```

CODE SNIPPETS

ThirdPartySpotify search

```java
public class ThirdPartySpotify implements ThirdPartyIntegrationStrategy {

  @Override
  public Response search(String queryString,String queryType) throws IOException {
    //generate token
    //building API URI
    //add query params
    URL url = new URL(SEARCH_URI);
    //make a get request to search
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();

    conn.setRequestProperty("Authorization","Bearer "+accessToken);
    conn.setRequestProperty("Content-Type","application/json");
    conn.setRequestMethod("GET");
    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String output;
    StringBuffer response = new StringBuffer();
    while ((output = in.readLine()) != null) {
      response.append(output);
    }
    in.close();
    // printing result from response

    //building JSON

      return Response.ok(finalJsonObject.toString(),MediaType.APPLICATION_JSON).build();
  }


```

CODE SNIPPETS

ThirdPartySpotify recommend

```java
public class ThirdPartySpotify implements ThirdPartyIntegrationStrategy {
  @Override
  public Response recommend(String queryString,String queryType) throws IOException {
    //get access token
    //build the API URI

    ArrayList<String> artistNames = getArtistNames(queryString);
    String seedArtists="";
    for (int i=0 ;i<artistNames.size();i++) {
      String seedArtist = getArtistSeed(accessToken,artistNames.get(0));
      seedArtists = seedArtists + seedArtist+",";
    }
    seedArtists = seedArtists.substring(0, seedArtists.length() - 1);
    queryString=seedArtists;

    String RECOMMEND_URI = BASE_URI+"?"+queryType+"="+queryString+"&limit=25";


    URL url = new URL(RECOMMEND_URI);

        HttpURLConnection connection = (HttpURLConnection) url.openConnection();

        connection.setRequestMethod("GET");
        connection.setRequestProperty("Authorization","Bearer "+" "+accessToken);
        connection.setRequestProperty("Accept", "application/json");
        connection.setRequestProperty("Connection", "keep-alive");

        // Send the request and get response
        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();

        //building JSON
      return Response.ok(finalJsonObject.toString(),MediaType.APPLICATION_JSON).build();
  }
```

CODE SNIPPETS

ThirdPartyLastFM recommend

```java
public class ThirdPartyLastFM implements ThirdPartyIntegrationStrategy {
  @Override
  public Response recommend(String queryTracks,String queryArtists)throws IOException {
    ArrayList<String> artists = getArtist(queryArtists);
    ArrayList<String> tracks = getTracks(queryTracks);
    JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

    for (int i =0 ;i<artists.size();i++) {
      getRequestForLastfm(arrayBuilder,artists.get(i),tracks.get(i));

    }

    JsonObjectBuilder finalBuilder = Json.createObjectBuilder();
      finalBuilder.add("tracks", arrayBuilder);
      JsonObject finalJsonObject = finalBuilder.build();

      return Response.ok(finalJsonObject.toString(),MediaType.APPLICATION_JSON).build();

  }

  public static void getRequestForLastfm(JsonArrayBuilder arrayBuilder,String artist,String track) throws IOException {
    //defining API KEY and URI

    URL url = new URL(RECOMMEND_URI);

    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

    connection.setRequestProperty("Content-Type","application/json");
    connection.setRequestMethod("GET");

    BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    String output;

    StringBuffer response = new StringBuffer();
    while ((output = in.readLine()) != null) {
      response.append(output);
    }

    in.close();

    //building JSON
  }
```

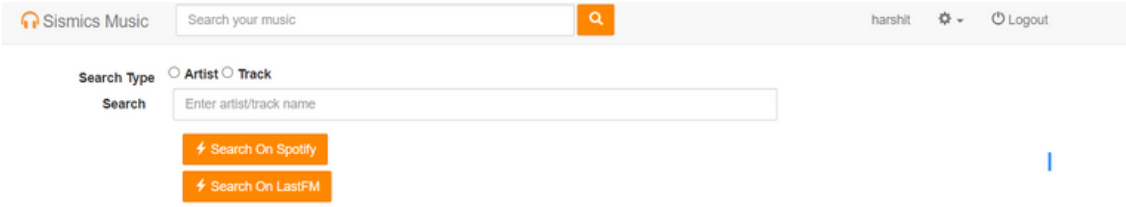
CODE SNIPPETS

ThirdPartyLastFM search

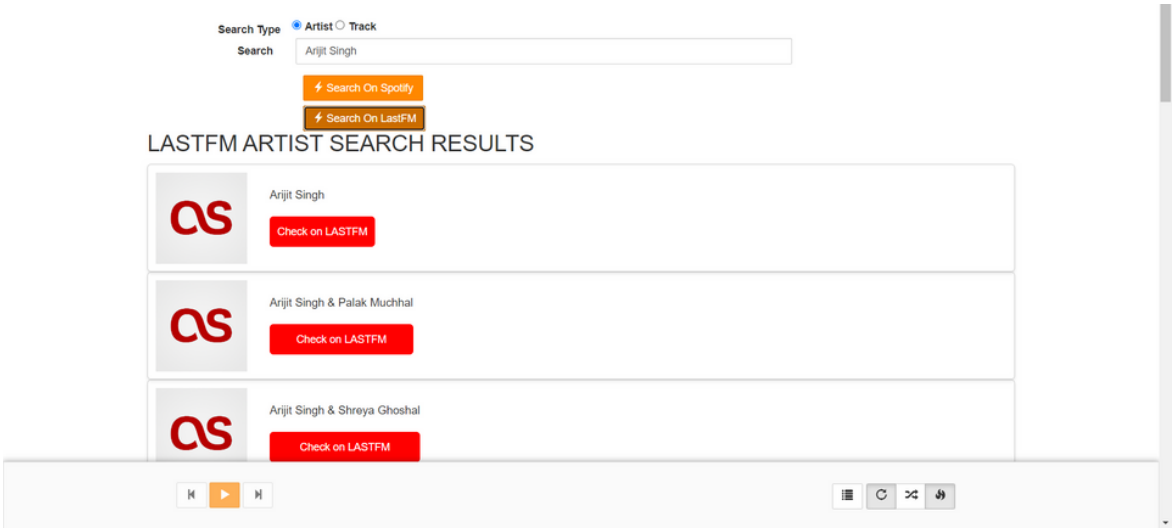
```
public class ThirdPartyLastFM implements ThirdPartyIntegrationStrategy {

  @Override
  public Response search(String queryString,String queryType) throws IOException{
    //building the URI from API KEY & paramaters

    URL url = new URL(SEARCH_URL);

    //make a get request to search
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();

    conn.setRequestProperty("Content-Type","application/json");
    conn.setRequestMethod("GET");

    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String output;

    StringBuffer response = new StringBuffer();
    while ((output = in.readLine()) != null) {
      response.append(output);
    }

    in.close();
    // printing result from response

    //building JSON

      return Response.ok(finalJsonObject.toString(),MediaType.APPLICATION_JSON).build();

  }
```

UI SCREENSHOTS

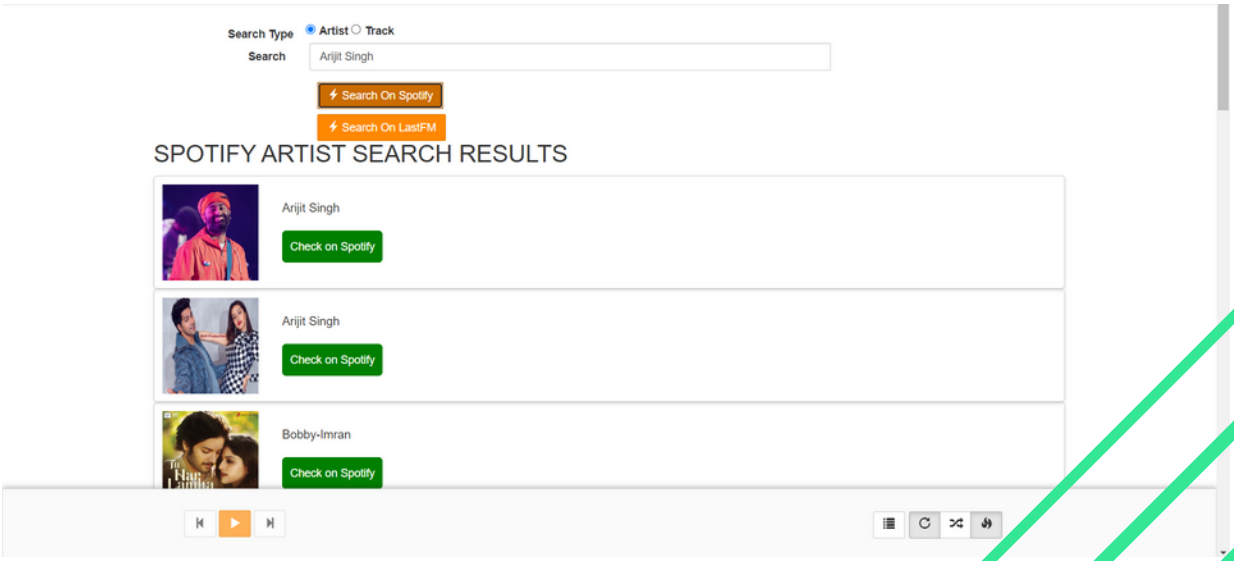
Track/Artist Search Page



LASTFM Artist Search Results

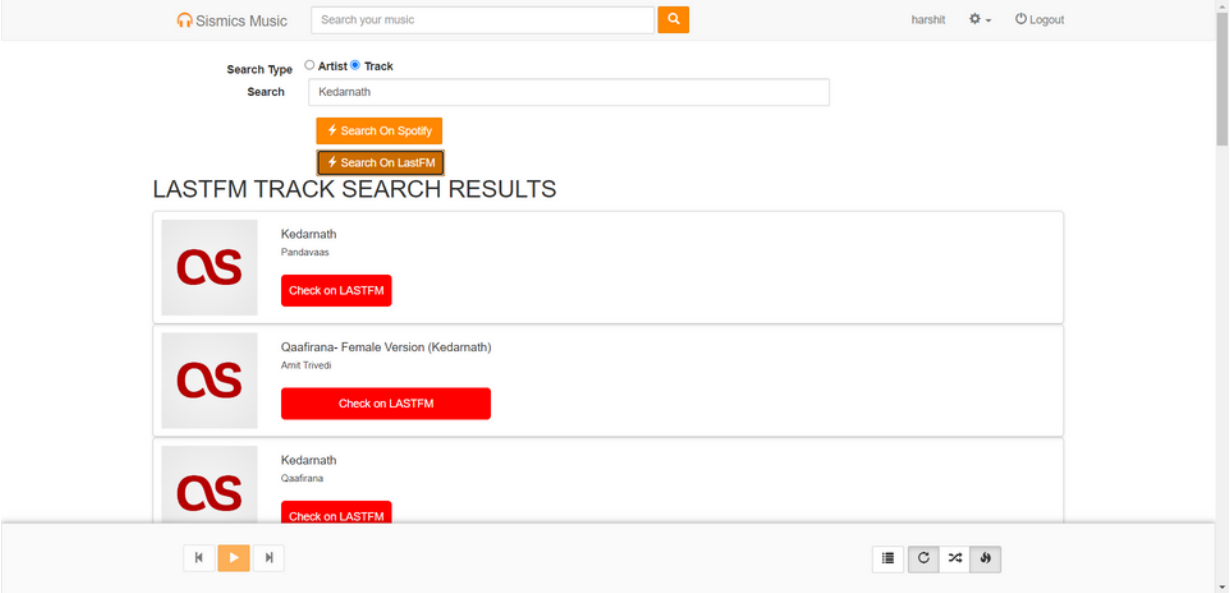


SPOTIFY Artist Search Results

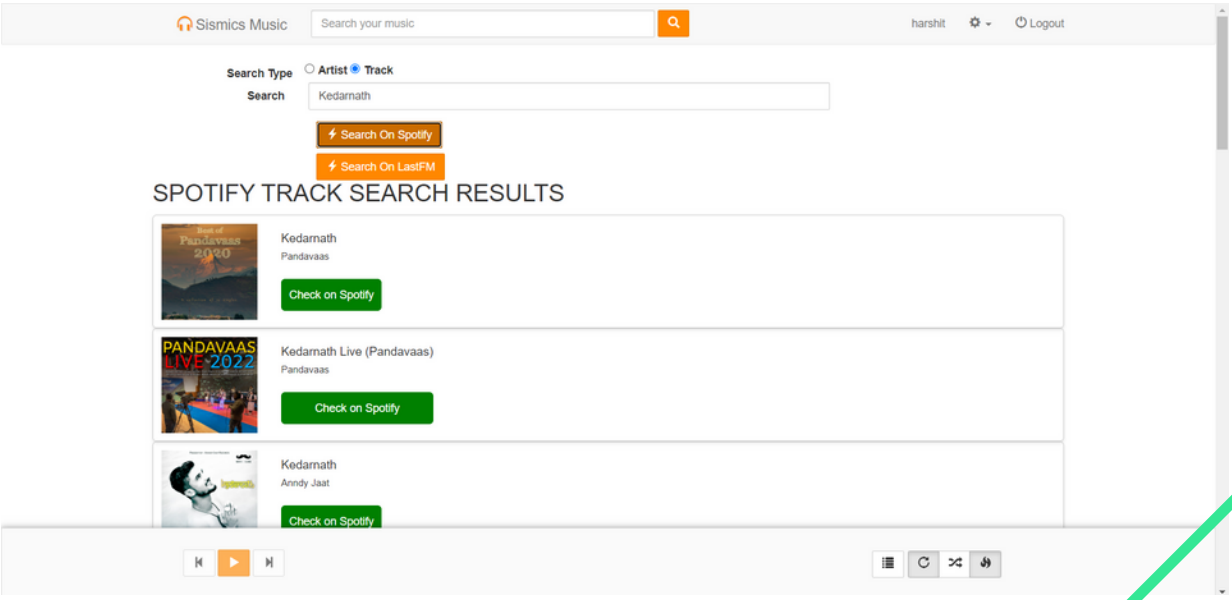


UI SCREENSHOTS

LASTFM Track Search Results

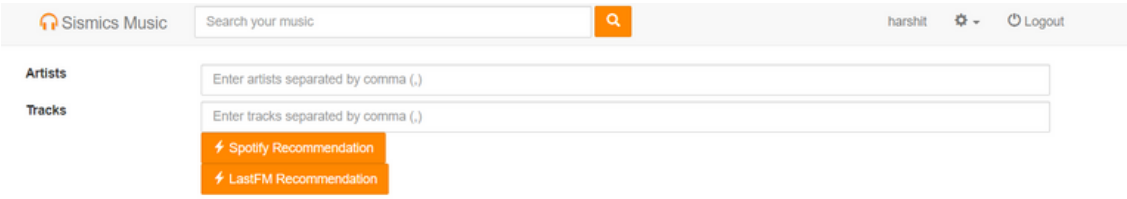


SPOTIFY Track Search Results

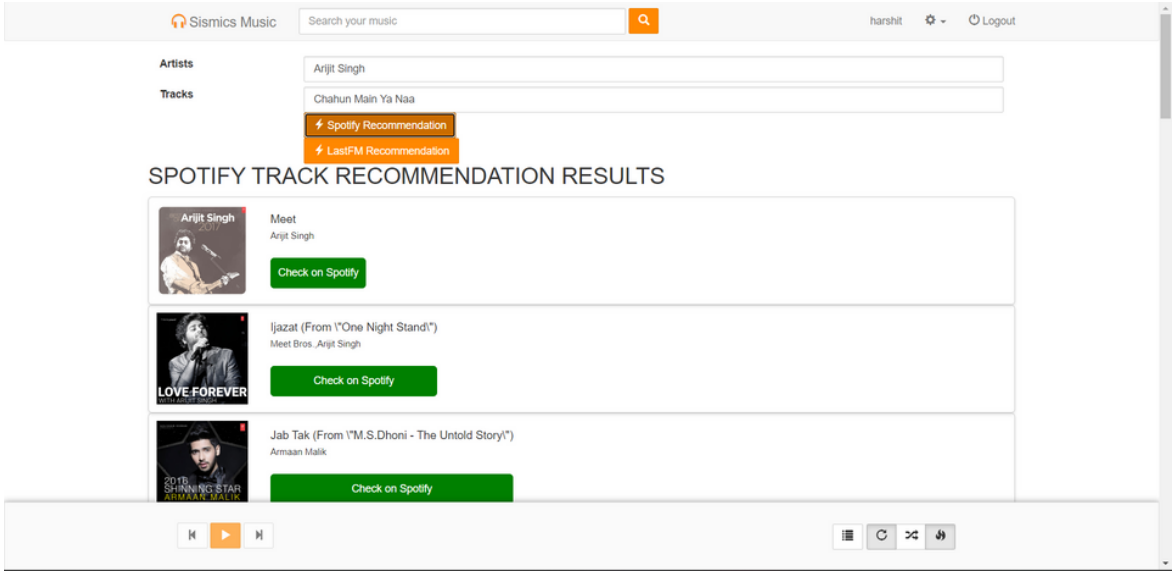


UI SCREENSHOTS

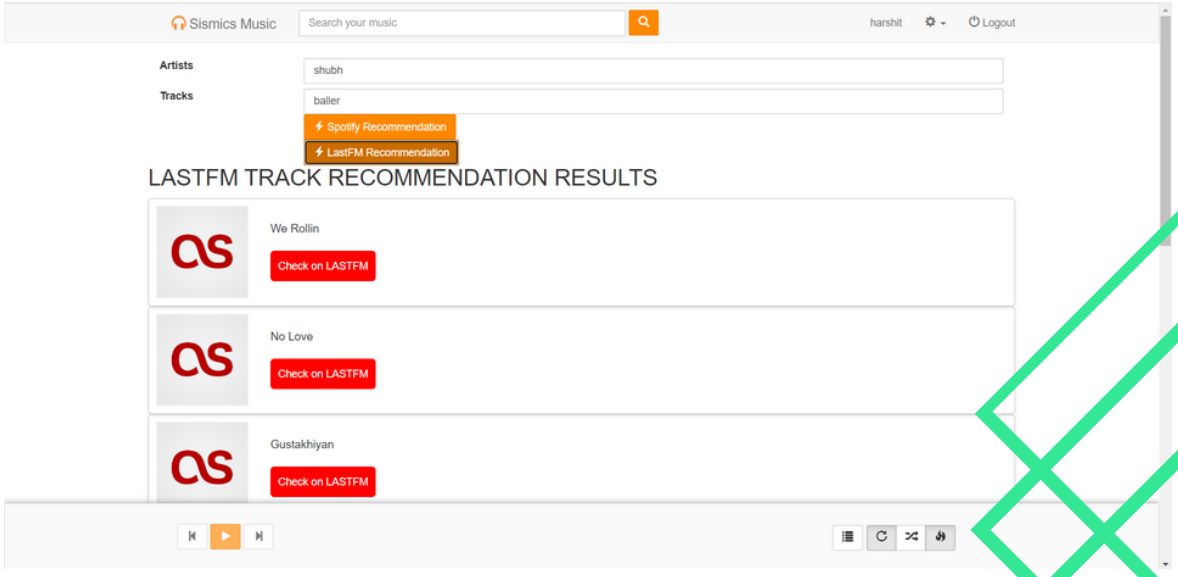
Recommendation through input



SPOTIFY Track Search Results

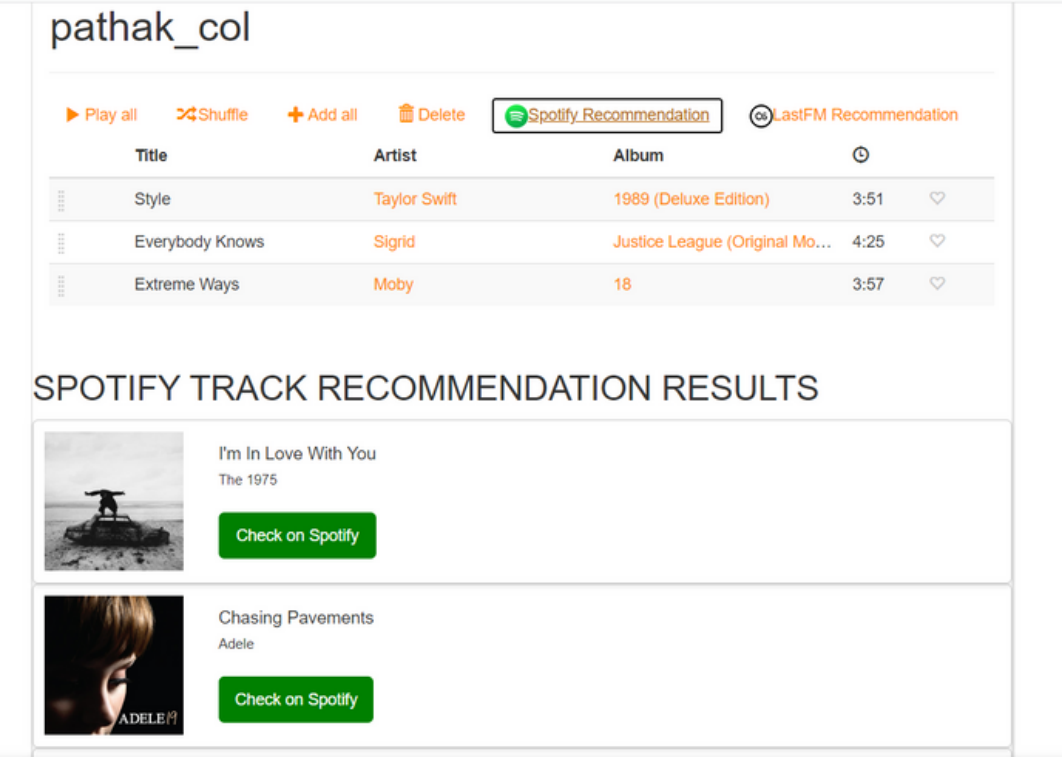


LASTFM Track Search Results

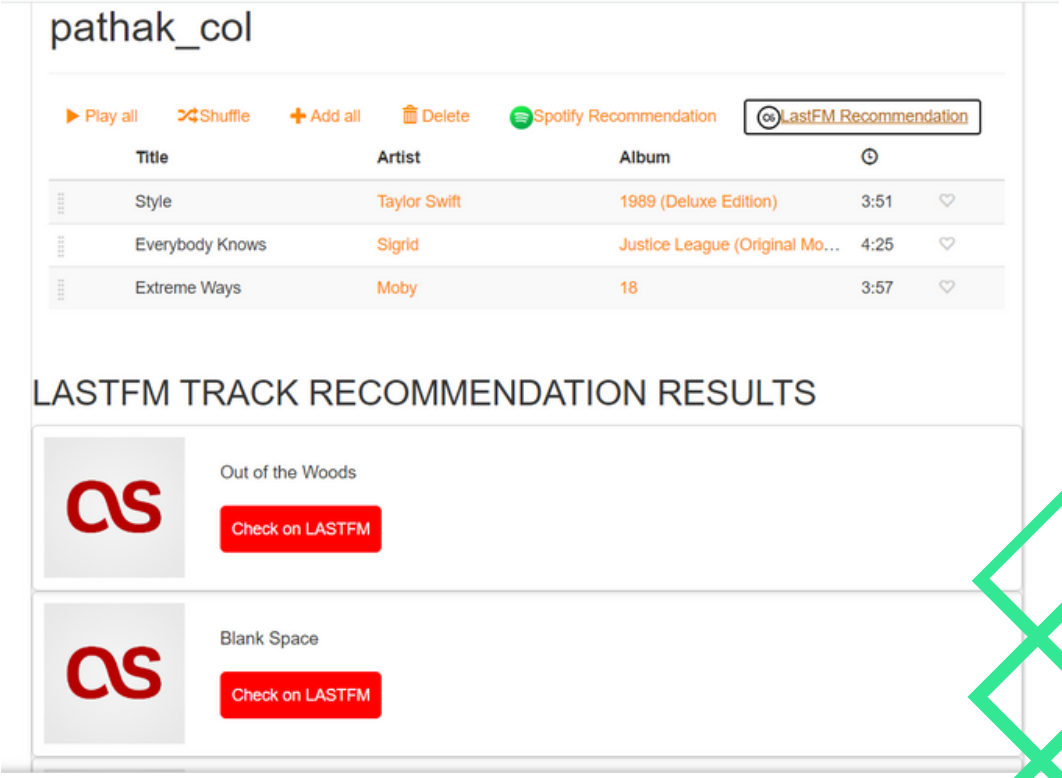


UI SCREENSHOTS

SPOTIFY track recommendation of playlist



LASTFM track recommendation of playlist



CONTRIBUTION OF TEAM MEMBERS

- **Dishant Sharma:** Search music based on artist name, album name, and Recommendations of music based on the playlist music using lastFm and Spotify. Applied Strategy design pattern on recommendation and search and also front end for search and recommendation.

- **Harshit Kashyap:** Improvement in Library Management of music that involves the facility to make the playlists public for all so that all the users can view songs in the playlist. It also includes making the songs uploaded by the users private for all the users. Added Builder , Template and Chain of responsibility design pattern and frontend for making the music private and private to public and collaborative(bonus).

- **Shubham Deshmukh:** Enhancement of the user Management system, where the new intended users do not rely on the administrator where they can only make register the user, they can make an account for themselves on the login and account creation page. Also created the front end for user management.

- **Udrasht Pal:** Search music based on artist name, album name, and Recommendations of music based on the playlist music using lastFm and Spotify. Applied Strategy design pattern on recommendation and search and also front end for search and recommendation.

- **Utkarsh Pathak:** Enhancement of the user Management system, where the new intended users do not rely on the administrator where they can only make register the user, they can make an account for themselves on the login and account creation page. Also created the front end for user management.