SISMICS MUSIC PROJECT -2 BONUS TASK

SOFTWARE ENGINEERING

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

BONUS - COLLABORATIVE PLAYLISTS

LIMITATIONS

The Playlists are either categorized as private or public only. The individual users can view and add songs to their private playlist. All the users can only view the public playlist, they have no provision to add music to public playlist(except owner user could do so).

IMPROVEMENT

Library Management services have moved to another extent where the playlist could be designated as a collaborative playlist. A collaborative playlist can be simply considered as a public playlist where all the users can view it and add songs to it. While creating playlists the users can it designate it as either private, public, or collaborative. A collaborative playlist allows the users to add songs to the public playlist so that multiple users can collaborate to create a better playlist.

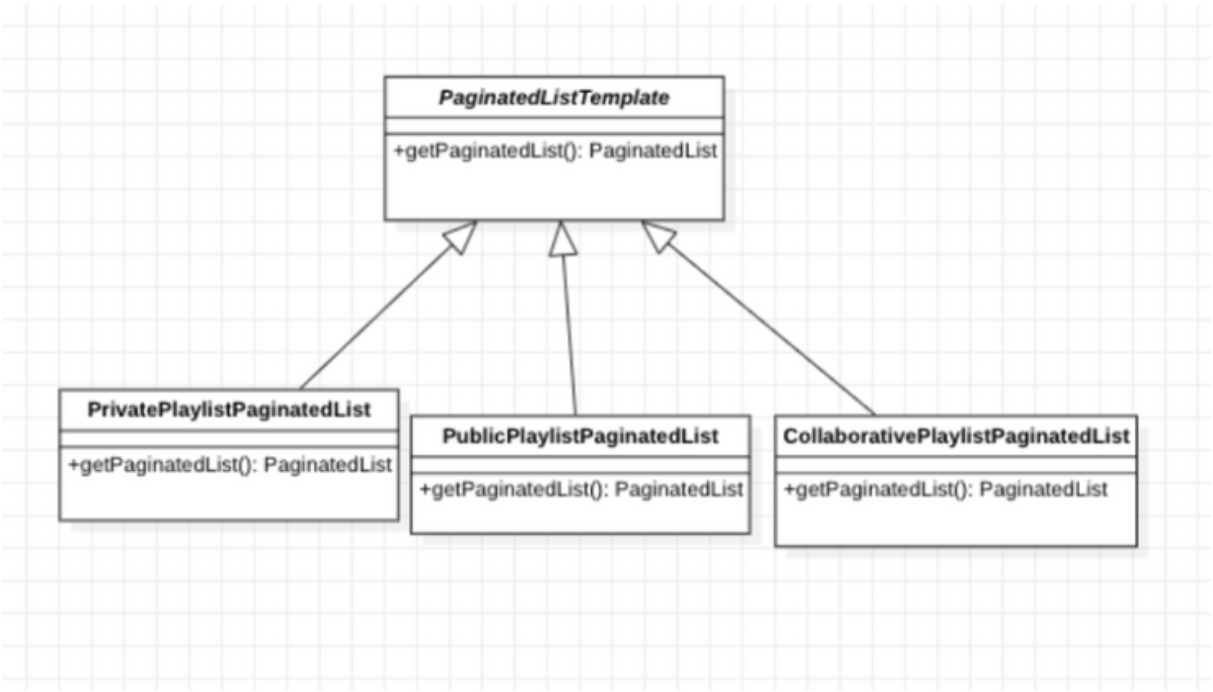
DESIGN PATTERN - BUILDER + TEMPLATE METHOD

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.

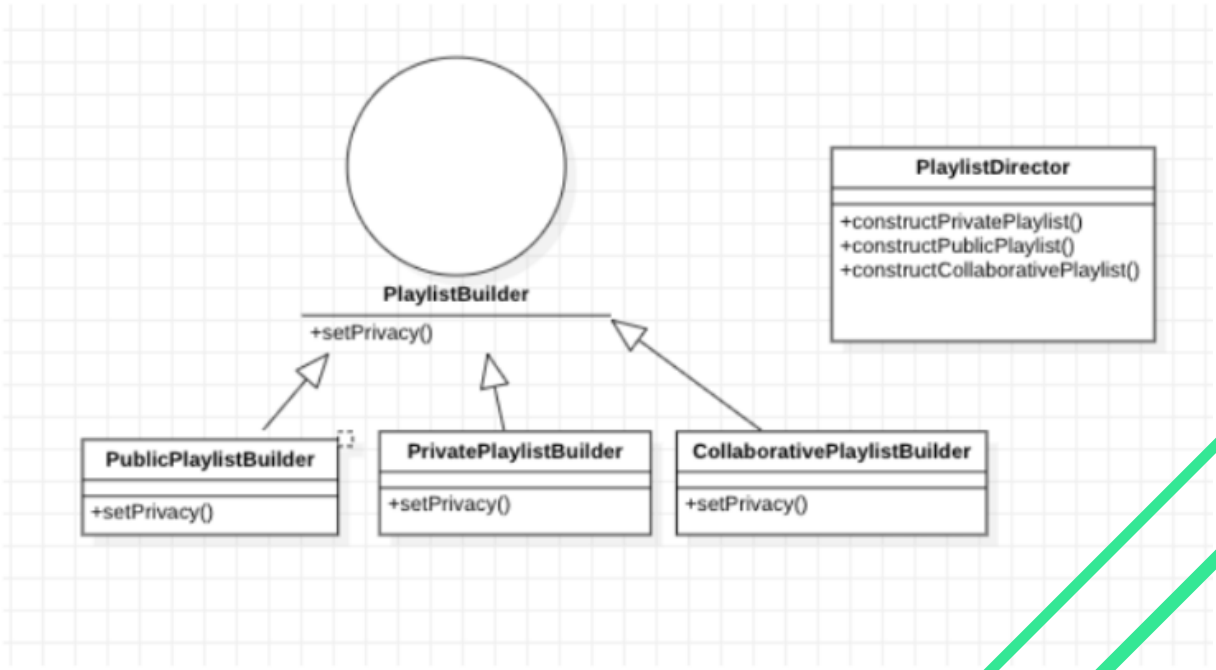
The Template Method is a behavioral design pattern that defines the skeleton of an algorithm in a superclass but allows subclasses to override specific steps of the algorithm without changing its structure.In this pattern, the template method is defined in the base class and provides a series of steps that make up the algorithm.

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 , private or collaborative 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 a criteria to get the private, public and collaborative playlist

CODE SNIPPETS

PlaylistResource Template

```java
@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

```java

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

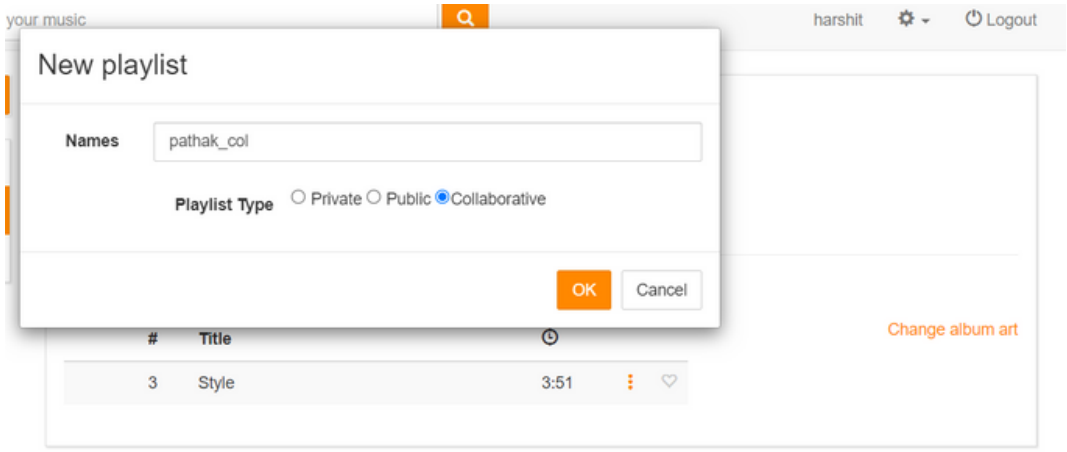
```
@Path("/playlist")
public class PlaylistResource extends BaseResource {
  public static final String DEFAULt_playlist = "default";
  @PUT
  public Response createPlaylist(
      @FormParam("name") String name, @FormParam("privacy") String privacy) {
    if (!authenticate()) {
      throw new ForbiddenClientException();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

    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()));
  }

```

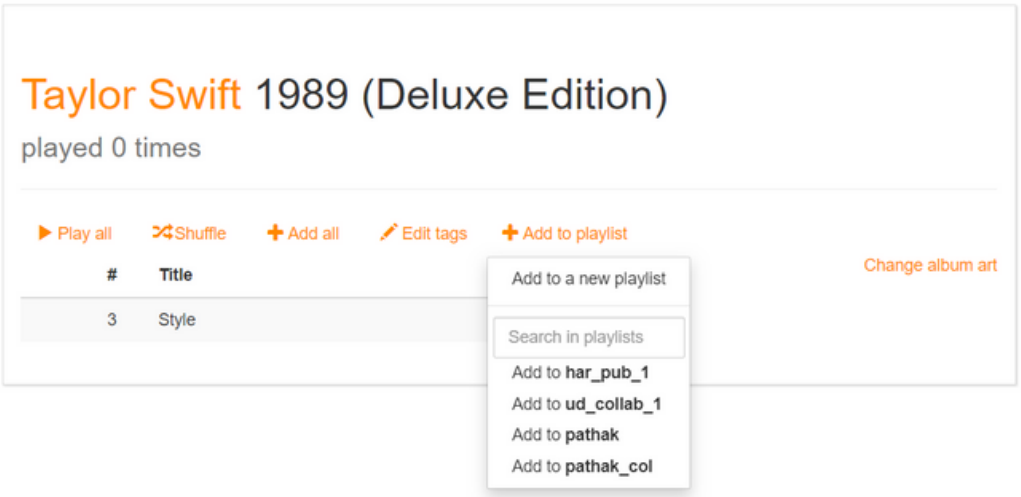
CODE SNIPPETS

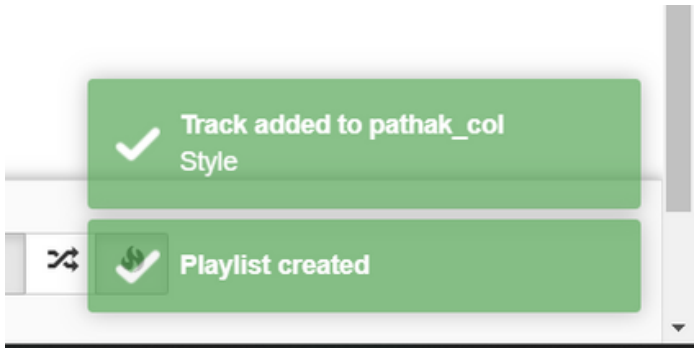
PlaylistDao

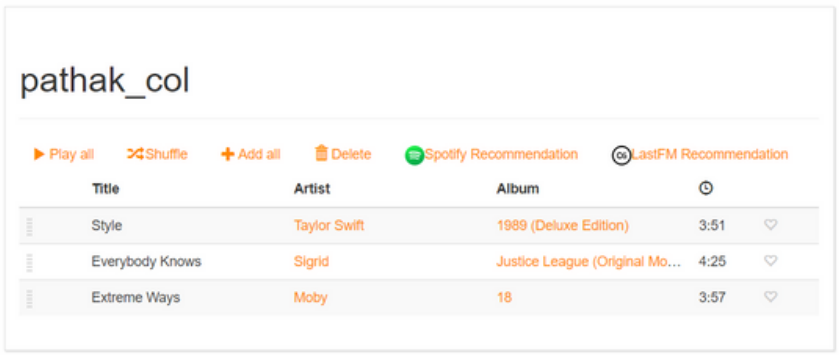
```java
public class PlaylistDao extends BaseDao<PlaylistDto, PlaylistCriteria> {
    @Override
    protected QueryParam getQueryParam(PlaylistCriteria criteria, FilterCriteria filterCriteria) {
        StringBuilder sb = new StringBuilder("select p.id as id, p.name as c0,")
                .append("  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









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.