Task 3

To implement the features requested in Part 3, we've employed the Strategy design pattern. This pattern helps us to abstract the details of execution of the same operation — perfect for the search and recommendation feature in which we need to conduct an operation using one of either Spotify, Last.fm or Local.

The approach we took is to create an abstract class in which there is a method that will need to be overridden by the respective Search/Recommendation strategy class that will be made concrete (along with some other helper functions).

Besides the Java changes, there are changes to the AngularJS frontend:

- Buttons were added in the playlist view to get recommendations.
- Added a recommendation result page where the user can view recommendations for a certain playlist.
- Radio buttons were added next to the search bar to allow the user to select the strategy in which the search will be conducted. (By default, it searches locally)

Here's a summary of the per-file Java code changes:

SearchResource.java

This class no longer runs a local search, like before. It instead chooses a strategy according to the `strat` path parameter supplied to the route. And then runs the search.

```
package com.sismics.music.rest.resource;

// ... imports

/**
 * Search REST resources.
 *
 * @author jtremeaux
 */
@Path("/search")
```

```
public class SearchResource extends BaseResource {
    /**
     * Run a full text search.
     *
     * @param query Search query
     * @param limit Page limit
     * @param offset Page offset
     * @return Response
     */
    @GET
    @Path("{query: .+}")
    public Response get(
            @PathParam("query") String query,
            @QueryParam("strat") String strat,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) {

        //
        //  removed the local searching and moved it to
        //  LocalSearch.java
        //

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

        if(strat == null || strat == "") {
            strat = "local";
        }
        Validation.required(query, "query");

        SearchStrategy searcher = new LocalSearch();
        switch(strat) {
            case "spotify":
                searcher = new SpotifySearch();
                break;
            case "lastfm":
                searcher = new LastFmSearch();
                break;
            default:
                searcher = new LocalSearch();
        }

        return searcher.getSearchResponse(query, limit,
offset, principal.getId());
```

```
    }
}
```

SearchStrategy.java

This class is our abstract strategy class for the search operation. This class defines `getSearchResponse()` that needs to be overridden by any child class that wants to be a search strategy. It also contains some helper functions for behaviour that's shared between search strategies.

```
package com.sismics.music.rest.search;

// ... imports

public abstract class SearchStrategy {
    abstract public Response getSearchResponse(String query,
Integer limit, Integer offset, String userID);

    protected Response renderJson(JsonObjectBuilder response)
{
        return Response.ok()
                .entity(response.build())
                .build();
    }

    protected Response generateResponse(List<TrackDto>
trackList, List<AlbumDto> albumList, List<ArtistDto>
artistList) {

        JsonObjectBuilder response =
Json.createObjectBuilder();

        // ... tracks
        response.add("tracks", tracks);

        // ... albums
        response.add("albums", albums);
```

```
        // ... add artists
        response.add("artists", artists);

        return renderJson(response);
}
}
```

LocalSearch.java

This class contains the old behaviour that was in `SearchResource.java` that conducted a Local Search. It overrides `getSearchResponse()` and returns a result from the local collection.

```
package com.sismics.music.rest.search;

// ... imports

public class LocalSearch extends SearchStrategy{
    @Override
    public Response getSearchResponse(String query, Integer
limit, Integer offset, String userID) {
        // Search tracks
        PaginatedList<TrackDto> paginatedList =
PaginatedLists.create(limit, offset);
        new TrackDao().findByCriteria(paginatedList, new
TrackCriteria()
                .setUserId(userID)
                .setLike(query), null, null);
        List<TrackDto> trackList =
paginatedList.getResultList();

        // Search albums
        AlbumDao albumDao = new AlbumDao();
        List<AlbumDto> albumList =
albumDao.findByCriteria(new
AlbumCriteria().setUserId(userID).setNameLike(query));
```

```
        // Search artists
        ArtistDao artistDao = new ArtistDao();
        List<ArtistDto> artistList =
artistDao.findByCriteria(new
ArtistCriteria().setNameLike(query));

        return generateResponse(trackList, albumList,
artistList);
    }
}
```

SpotifySearch.java

This class contains the behaviour that conducts a Spotify search. It overrides `getSearchResponse()` and returns a result generated from calling the Spotify API.

```
package com.sismics.music.rest.search;

// ... imports

public class SpotifySearch extends SearchStrategy{
    static final String SPOTIFY_CLIENT_ID =
"d263a52b62814e78badf281a32d02702";
    static final String SPOTIFY_CLIENT_SECRET =
"********************************";

    private List<TrackDto>
convertSpotifyJsonToTrackDTOs(String jsonString) {
        List<TrackDto> trackList =  new ArrayList<TrackDto>
();
        JsonReader jsonReader = Json.createReader(new
StringReader(jsonString));
        JsonObject jsonObj = jsonReader.readObject();

        JsonArray tracksArray =
jsonObj.getJsonObject("tracks").getJsonArray("items");
```

```
            for (int i = 0; i < tracksArray.size(); i++) {
                JsonObject track = tracksArray.getJsonObject(i);
                // ...
                TrackDto tdto = new TrackDto();
                // ... construct tdto from track
                trackList.add(tdto);
            }
            return trackList;
        }

        // ...

        String getAccessToken() {
            String access_token = null;
            try {
                // ... call Spotify search API and save
                // access_token
            } catch (Exception e) {
                System.out.println("Access token Error: " +
e.getMessage());
            }
            return access_token;
        }


        @Override
        public Response getSearchResponse(String query, Integer
limit, Integer offset, String userID) {
            offset = 0;
            StringBuffer responseString = new StringBuffer();
            String access_token = getAccessToken();

            if (access_token != null) {
                try {
                    // ... call Spotify API and save response
                } catch (Exception e) {
                    System.out.println("Error: " +
e.getMessage());
                }
            }
            List<TrackDto> trackList =
convertSpotifyJsonToTrackDTOs(responseString.toString());
            List<AlbumDto> albumList =
convertSpotifyJsonToAlbumDTOs(responseString.toString());
```

```
        List<ArtistDto> artistList =
convertSpotifyJsonToArtistDTOs(responseString.toString());
        return generateResponse(trackList, albumList,
artistList);
    }

    // used in Spotify Recommendation to get the spotifyID of
a track
    public String getSpotifyID(String songQuery, String
userID) {
        JsonObject responseJson = (JsonObject)
getSearchResponse(songQuery, 1, 0, userID).getEntity();
        return
responseJson.getJsonArray("tracks").getJsonObject(0).getString
("id");
    }
}
```

LastfmSearch.java

This class contains the behaviour that conducts a Last.fm search. It overrides `getSearchResponse()` and returns a result generated from calling the Last.fm API.

```
package com.sismics.music.rest.search;

// ... imports

public class LastFmSearch extends SearchStrategy{

    final String MY_LAST_FM_KEY =
"61a8565397242d3b4127aaa4f3cf767c";
    final String MY_LAST_FM_SECRET =
"********************************";

    private List<TrackDto>
convertLastFmJsonToTrackDTOs(String jsonString) {
```

```
        List<TrackDto> trackList =  new ArrayList<TrackDto>
();
        JsonReader jsonReader = Json.createReader(new
StringReader(jsonString));
        JsonObject jsonObj = jsonReader.readObject();

        JsonArray tracksArray =
jsonObj.getJsonObject("results").getJsonObject("trackmatches")
.getJsonArray("track");

        for (int i = 0; i < tracksArray.size(); i++) {
            JsonObject track = tracksArray.getJsonObject(i);
            TrackDto tdto = new TrackDto();
            // .. construct TrackDto
        }
        return trackList;
    }

    @Override
    public Response getSearchResponse(String query, Integer
limit, Integer offset, String userID) {
        offset = 0;
        StringBuffer responseString = new StringBuffer();
        try{
            // ... call the Last.fm search API and save
            // response
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

        List<TrackDto> trackList =
convertLastFmJsonToTrackDTOs(responseString.toString());
        List<AlbumDto> albumList = new ArrayList<AlbumDto>();
        List<ArtistDto> artistList = new ArrayList<ArtistDto>
();

        return generateResponse(trackList, albumList,
artistList);
    }
}
```

RecommendResource.java

Creates a backend API route for the frontend to call when it needs recommendations. `playlistid` and `strat` is provided as a path parameter, they contain specifying information about which playlist needs recommendations and which strategy is to be used. This class instantiates a child of `RecommendStrategy` and assigns to it the appropriate concrete object according to `strat` and calls `getRecommendations()`.

```
package com.sismics.music.rest.resource;

// ... imports

/**
 * Search REST resources.
 *
 * @author jtremeaux
 */
@Path("/recommend")
public class RecommendResource extends BaseResource {
    /**
     * Run a full text search.
     *
     * @param query Search query
     * @param limit Page limit
     * @param offset Page offset
     * @return Response
     */
    @GET
    @Path("{playlistid: .+}")
    public Response get(
            @PathParam("playlistid") String playlistid,
            @QueryParam("strat") String strat,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) {

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

        if(strat == null || strat == "") {
            strat = "lastfm";
```

```
        }
        Validation.required(playlistid, "playlistid");

        RecommendStrategy searcher = new LastFmRecommend();
        if(strat.equals("spotify")) searcher = new
SpotifyRecommend();

        return searcher.getRecommendations(playlistid, limit,
principal.getId());
    }
}
```

RecommendStrategy.java

This class is our abstract strategy class for the recommend operation. This class defines `getRecommendations()` that needs to be overridden by any child class that wants to be a recommendation strategy. It also contains some helper functions for behaviour that's shared between recommendation strategies.

```
package com.sismics.music.rest.resource.recommend;

// ... imports

public abstract class RecommendStrategy {
    abstract public Response getRecommendations(String
playlistID, Integer limit, String userID);

    protected Response renderJson(JsonObjectBuilder response)
{
        return Response.ok()
                .entity(response.build())
                .build();
    }

        protected Response generateResponse(List<TrackDto>
trackDtoList) {
```

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

            // ... add tracks
            response.add("tracks", tracks);

            return renderJson(response);
    }
}
```

SpotifyRecommend.java

This class contains the behaviour that conducts a Spotify recommendation. It overrides `getRecommendations()` and returns a result generated from calling the Spotify API. The method is supplied with the playlist ID and gets the list of tracks from `TrackDao`.

```
package com.sismics.music.rest.resource.recommend;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.Base64;
import java.util.List;

import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.ws.rs.core.Response;

import com.sismics.music.core.dao.dbi.TrackDao;
import com.sismics.music.core.dao.dbi.criteria.TrackCriteria;
import com.sismics.music.core.dao.dbi.dto.TrackDto;
import com.sismics.music.rest.search.SpotifySearch;
```

```
public class SpotifyRecommend extends RecommendStrategy{
    static final String SPOTIFY_CLIENT_ID =
"d263a52b62814e78badf281a32d02702";
    static final String SPOTIFY_CLIENT_SECRET =
"********************************";

    private List<TrackDto>
convertSpotifyJsonToTrackDTOs(String jsonString) {
        List<TrackDto> trackList =  new ArrayList<TrackDto>
();
        JsonReader jsonReader = Json.createReader(new
StringReader(jsonString));
        JsonObject jsonObj = jsonReader.readObject();

        JsonArray tracksArray =
jsonObj.getJsonArray("tracks");

        for (int i = 0; i < tracksArray.size(); i++) {
            JsonObject track = tracksArray.getJsonObject(i);
            // ...
            TrackDto tdto = new TrackDto();
            // ... construct tdto from track
            trackList.add(tdto);
        }
        return trackList;
    }

    private String getSpotifyIdsOfPlaylistTracks (String
playlistid, String userID) {
        // ... constructs a list Spotify IDs for each
        // track in a playlist specified by playlistid
        return spotifyIdsString;
    }

    String getAccessToken() {
        String access_token = null;
        try {
            // ... call Spotify search API and save
            // access_token
        } catch (Exception e) {
            System.out.println("Access token Error: " +
e.getMessage());
        }
        return access_token;
    }
```

```
    @Override
    public Response getRecommendations(String playlistid,
Integer limit, String userID) {
        limit = 10;
        String spotifyIds =
getSpotifyIdsOfPlaylistTracks(playlistid, userID);

        String access_token = getAccessToken();
        StringBuffer responseString = new StringBuffer();

        if (access_token != null) {
            try {
                // ... call the Spotify Recommendations
                // API and save the response

            } catch (Exception e) {
                System.out.println("Error: " +
e.getMessage());
            }
        }
        return
generateResponse(convertSpotifyJsonToTrackDTOs(responseString.
toString()));
    }
}
```

LastfmRecommend.java

This class contains the behaviour that conducts a Last.fm recommendation. It overrides `getRecommendations()` and returns a result generated from calling the Last.fm API. The method is supplied with the playlist ID and gets the list of tracks from `TrackDao`.

```
package com.sismics.music.rest.resource.recommend;

// ... imports

public class LastFmRecommend extends RecommendStrategy{
```

```
    final String MY_LAST_FM_KEY =
"61a8565397242d3b4127aaa4f3cf767c";
    final String MY_LAST_FM_SECRET =
"********************************";

    private List<TrackDto>
convertLastFmJsonToTrackDTOs(String jsonString) {
        List<TrackDto> trackList =  new ArrayList<TrackDto>
();
        JsonReader jsonReader = Json.createReader(new
StringReader(jsonString));
        JsonObject jsonObj = jsonReader.readObject();

        JsonArray tracksArray =
jsonObj.getJsonObject("similartracks").getJsonArray("track");

        for (int i = 0; i < tracksArray.size(); i++) {
            JsonObject track = tracksArray.getJsonObject(i);
            TrackDto tdto = new TrackDto();
            // ... construct tdto from track
            trackList.add(tdto);
        }
        return trackList;
    }

    @Override
    public Response getRecommendations(String playlistid,
Integer limit, String userID) {
        limit = 2;
        List<TrackDto> trackList =  new ArrayList<TrackDto>
();

        List<TrackDto> playlistTrackList = new
TrackDao().findByCriteria(new TrackCriteria()
            .setUserId(userID)
            .setPlaylistId(playlistid));

        for(TrackDto playlistTrack : playlistTrackList)
        {
            StringBuffer responseString = new
StringBuffer();
            try{
                String trackName = playlistTrack.getTitle();
```

```
                    String trackArtist =
playlistTrack.getArtistName();

                    // ... call the Last.fm Recommendations
                    // API and save the response

 trackList.addAll(convertLastFmJsonToTrackDTOs(responseString.
toString()));
                } catch (Exception e) {
                    System.out.println("Error: " +
e.getMessage());
                }
            }
            return generateResponse(trackList);
        }
}
```