Playlist Generator

Final Project Assignment

Java Team Project

This document describes the final project assignment for the Java cohort at Telerik Academy.

Project Description

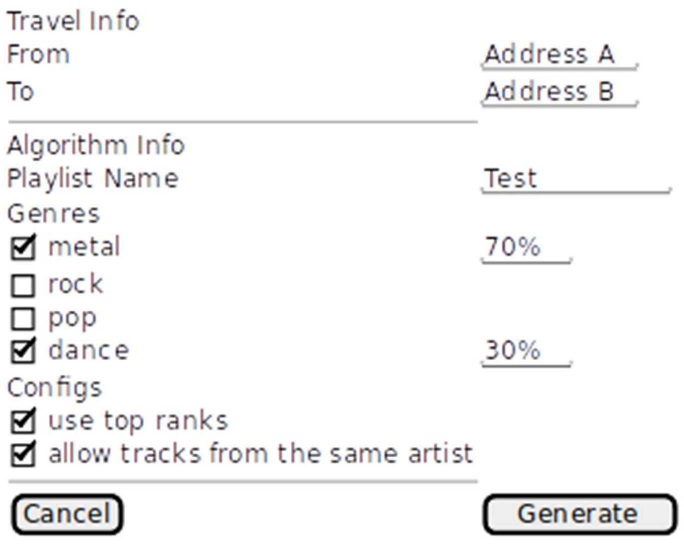
Your task is to develop the **Playlist Generator** web application. **Playlist Generator** enables your users to generate playlists for specific travel duration periods based on their preferred genres.

A playlist consists of a list of individual tracks(songs). Each track has an artist, title, album, duration (playtime length) and rank (a numeric value). Tracks may or may not have a preview URL to an audio stream (e.g. the users can click and play the preview part of the track).
Each playlist has a user given title, associated tags (e.g. musical genres), a list of tracks with the associated track details, total playtime (the sum of the playtimes of all tracks in that playlist) and rank (the average of the ranks of all tracks in that playlist).

Main Usecase

A user travellling from A to B wants to have something to listen to during the duration of the travel. The user wants to generate a track list based on his musical tastes (the user selects genres from a predefined list). An algorithm, which uses external service as track sample data, generates the playlist.

1) User MUST be presented with an UI which lets them enter starting and destination address, selects musical genres and clicks "Generate". Something like:



Travel Info	
From	Address A
To	Address B
Algorithm Info	
Playlist Name	Test
Genres	
☑ metal	70%
☐ rock	
☐ pop	
☑ dance	30%
Configs	
☑ use top ranks	
☑ allow tracks from the same artist	
Cancel	Generate

Note: The purpose of the above wireframe is to only illustrate the basic UI components, and is not a guideline of how UI look like. Although the focus of the project is on the backend, a more sensible and intuitive UI design is expected.

2) **Playlist Generator** calculates the travel duration time between the starting and destination locations and combines tracks chosen randomly in the specified genres, until the playing time roughly matches the travel time duration. Rounding of +/- 5 minutes is allowed (e.g. for a travel duration of 89 minutes a playlist with total playtime between 84 or 94 and is fine).

3) The generated playlist is saved under this user’s profile and they can start listening to it.

The application offers browsing playlists created by other users and allows filtering by total duration and genre tags.

By default playlists should be sorted by average rank descending.

The application offers certain configuration over the playlist generation algorithm and the possibility to play a preview of the tracks playlist.

Project Requirements

UI

Use whatever UI technology suits you best.
Create a “don’t-make-me-think”, simple, intuitive UI, but keep in mind UI is not the main focus of the project (the backend is). You may checkout some existing music streaming services that offer playlists for design ideas.

Web Application

Public Part

The public part of your project should be visible without authentication. This includes the application start page, the user login and user registration forms, as well as the list of all user generated playlists. People that are not authenticated cannot see any user specific details, neither they can interact with the website. They can only browse the playlists and see the tracks list and details of them.

- Public page should contain at least 3 generated playlists
- Playlists can be clicked/expanded to show the list of artists/tracks
- Playlists show average rank (avg from tracks ranks) and total playtime
- Playlists are sorted by rank and can be filtered by name, genre and duration

Private Part (Users Only)

The private part is accessible only to users who have successfully authenticated (registered users).
The private part of the web application provides the users the ability to generate new playlists, control the generation algorithm, edit or delete their own existing playlists.
Editing existing playlists is limited to changing the title or associated genre tags, but does not include editing of the tracklist (e.g. removing or adding individual songs).

Administration Part

Users with the system administrator role can administer all major information objects. The administrators have the following functionality

- CRUD over users and other administrators
- CRUD over the playlists
- Can manually trigger Genre synchronization (if that optional requirement is implemented)

REST API

The backend and the frontend of the application should communicate using the REST architecture. The REST API should leverage HTTP as a transport protocol and clear text JSON for the request and response payloads.

Database

The data of the application MUST be stored in a relational database (MySQL is preferred). You need to identify the core domain objects and model their relationships accordingly.

External Services

The **Playlist Generator** web application will consume two public REST services in order to achieve the main functionality.

Microsoft Bing Maps

Microsoft Bing Maps offers similar functionality to Google maps but is free for non-commercial use. For usage details please see the Appendix.

Deezer

Deezer is a subscription music streaming service similar to Spotify and Google Play.
The API usage is free for non-commercial use and does not require any registration.
For usage details please see the Appendix.

Technical / Development Requirements

General development guidelines include, but are not limited to:

- Following OOP principles when coding
- Following KISS, SOLID, DRY principles when coding
- Following REST API design best practices when designing the REST API (see Apendix)
- Following BDD when writing tests
- You should implement sensible Exception handling and propagation
- every time you use System.out.println or e.printstacktrace() in a none-joking, serious manner, a kitten dies (see Optional requirements)

Backend

- The minimum JDK version is 1.8
- Use tiered project structure (separate the application components in layers)
- Use SpringMVC or SpringBoot framework
- For Persistence use MySQL/MariaDB
- Use Hibernate/JPA (and/or Spring Data) in the Persistence layer
- Use Spring Security to handle user registration and user roles
- Service layer (e.g. “business” functionality) should have at least 80% test unit coverage

Backend Data

- Pre-fetch the genres from Deezer and store them in the DB
- Choose at least 3 genres and pre-fetch 1,000 tracks from Deezer and store them in the DB

- User generated playlists should be stored in the DB
- User registration information and credentials should be stored in the DB
- Each track should have the following properties stored in the DB – id, title, link, duration, rank, preview URL, artist = {id, name, artist tracklist URL}, album = {id, name, album tracklist URL}

It's mandatory to create a relational database model between tracks, artists, albums, generated playlists etc. and store them in the database.

Each team is advised to choose at least 3 genres from https://api.deezer.com/genre and pre-fetch at least 1,000 tracks per genre.

The tracks should be stored in the DB and when the algorithm runs, it should read the tracks from the DB.

Note: We want to demonstrate the principles of working with external service and understanding its API and transforming the data from the external service into a local domain model.

We don't look for a total completeness (e.g. being able to work with all genres and all playlists and tracks). For example the genre "Rock" has 300 playlists with each playlist containing hundreds (about 200) of tracks. The tracks are returned in pages of 25 items per page. It's inadvisable to dump all tracks for "Rock" (~60K). Doing this programmatically will most likely exceed the quota.

Generation Algorithm Spec

- MUST NOT repeat tracks
- SHOULD NOT repeat artists (unless "allow tracks from the same artists" is selected)
- MUST generate random playlists. For example for the same two starting and destination locations, clicking Generate multiple times MUST NOT generate the same playlist twice (repeating tracks/artists between playlists is allowed).
- SHOULD allow the user to specify if they want more from specific genre using percentage as shown in the wireframe (or it can be left blank)
- Generated playlist total playtime is allowed to be +/- 5mins of the calculated travel duration
- **Optional** "allow tracks from the same artists" - by default the algorithm should not allow the same artist to appear twice in a single generated playlist. This checkbox overrides the default behavior.
- **Optional** "use top tracks" - each track has rank score. If this option is checked the algorithm should pick the highest ranking tracks from the tracks pool in the DB and generate a playlist with them (descending from highest to lowest rank).

Optional Backend Requirements

- implement a synchronization job for the Deezer Genres; The running period should be configurable (e.g. each hour synch the genres from Deezer). In the Admin panel display the last time the synchronization was ran, the status (success/failure) and relevant details (e.g. if failure some detailed message of the Exception)
- Implement "use top ranks" functionality
- Implement "allow tracks from the same artists" functionality
- Implement play functionality using the preview URL of each track. Use browser support or JS to bootstrap a player that can stream audio

- Use pixabay REST service to attach images related to music to newly generated playlists, for better presentation on the public page
- Use Logger in your application – see Spring Boot 2 logging with logback

Deliverables

- Each project source code MUST be available in a dedicated GitLab repository
- Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first etc. and the people who contributed. Contributions from all team members MUST be evident through the git commit history (so don't squash commits)!
- Read https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn and https://chris.beams.io/posts/git-commit/ for a guide to write good commit messages
- The repository MUST contain the complete application source code and any run scripts
- The project MUST have at least README.md documentation describing how to build and run the project
- Screenshots of the major application user facing screens with some data on them
- the url of the application, if hosted online

Public Project Defense

Each student must make a public defense of his work to the trainers, partner and students (~30-40 minutes). The defense includes:

- live demo of the application
- explanation of the application structure, major architectural components and selected source code pieces demonstrating the implementation of key features

Expectations

You MUST understand the system you have created.
Any defects or incomplete functionality MUST be properly documented and secured.
It's OK if your application has flaws or is missing one or two MUST's. What's not OK is if you don't know what's working and what isn't and if you present an incomplete project as functional.

Some things you need to be able to explain during your project presentation:

- What are the most important things you've learned while working on this project?
- What are the worst "hacks" in the project, or where do you think it needs more work?
- What would you do differently if you were implementing the system again?

Appendix

Guidelines for designing good REST API

https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy

Guidelines for URL encoding

http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/

Always prefer constructor injection

https://www.vojtechruzicka.com/field-dependency-injection-considered-harmful/

Microsoft Bing Maps External Service

We will use this external service to help us calculate the travel duration between two addresses.

Note: We're using this service because it's free. It doesn't require any payment information to be used, the registration process is straight forward as well as the generation and usage of API Key. Because of the service limitations, for addresses outside of the US, we can only use travelMode=driving. We can't use walking or transit. The point is to get familiar with consuming a REST service, understanding it's domain and do some data transformations.

The getting started documentation is available here: https://docs.microsoft.com/en-us/bingmaps/#pivot=main&panel=BingMapsAPI

API Key

Each team needs to register at least one account and get a free API key, which will allow them to make HTTP calls to the REST service. Each API key is limited in the number of requests etc. so it's not advisable to share you API key with another team as it may lead to locking or completely disabling the API key from Bing Maps. The API key is passed to every HTTP request in "key" query parameter.

Integration Guidelines

Locations

The first API endpoint that needs to be called is "Locations" and we will use it to find a Location by address. (official doc: https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address).

We will use the structured URL form, which specifies the location query parameters as part of the URL path:
http://dev.virtualearth.net/REST/v1/Locations/{countryRegion}/{adminDistrict}/{postalCode}/{locality}/{addressLine}?key=<YOUR_API_KEY>

For example the structured URL query for Telerik Academy's Location will look like:

http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia%20City/Mladost/1729/Alexandar%20Malinov%2031?key=<YOUR_API_KEY>

The result contains a collection of "geocodePoints". If there's more than one Geo Point returned we want the one of usageType "route".

Note: You can use the lat/long values returned as a result in any mapping service (e.g. Google Maps) just to test if your results are correct.

Distance Matrix
The next API endpoint that we will use is the Route's API distance matrix. which can calculate the time to travel from point A(lat, long) to point B (lat, long) for us. Official doc:
https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix

For example the URL to calculate the distance between The National Place of Culture and Teleric Academy will look like:

https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins=42.685428619384766,23.318979263305664&destinations=42.6508241,23.3790428&travelMode=driving&key=<YOUR_API_KEY>

Deezer External Service
Deezer is an audio streaming service, which will help us search and gather the tracks needed for the playlist generation.

Note: We're using this service because it's free. The service does not require registration, but there's a quota: "The number of requests per second is limited to 50 requests within 5 seconds." If you abuse the service it's very possible that the IP from which you're doing calls will be blocked.
The official docs start here: https://developers.deezer.com/api

Integration Guidelines
The resources in the Deezer public API have the following relationship:

Genres → Playlists → Tracks

Genres
The core genres offered to the user by your application will be seeded from
https://api.deezer.com/genre

It's preferable to store the genres in your DB and when needed fetch them from the DB instead of calling the Deezer service each time.

Playlists
Playlists can be loaded using https://api.deezer.com/search/playlist?q=<GENRE>.
This will fetch a list of playlists, which contain the word <GENRE> in their title.
There's a total number of playlists existing playlists as well as links to next and prev pages of results.

Each playlist contains a list of tracks, which can be fetched like:

https://api.deezer.com/playlist/1306931615/tracks

There’s a total number of tracks in the playlist as well as links to next and prev pages of results.

Pixabay External Service (Optional)

If you want to attach a random music picture over your generated playlists, you can use this service like: https://pixabay.com/api/?key=<YOUR_API_KEY>&q=music – it will return you random music pics which you can associate with your playlists

Suggested Approach

- Proof of Concept for work with 3rd party REST APIs
- Test out how to store incoming data
- Carefully design your Database (no user access control at this point)
- Start Implementing the algorithm
- Design and Create your user access control tables in the DB
- UI
- Optional Requirements

Contact

Жерар Хованесян - gerard.hovanessyan@tick42.com