

RidePal – Playlist Generator

Final Team Project Assignment

# Js Team Project

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

# Project Description

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

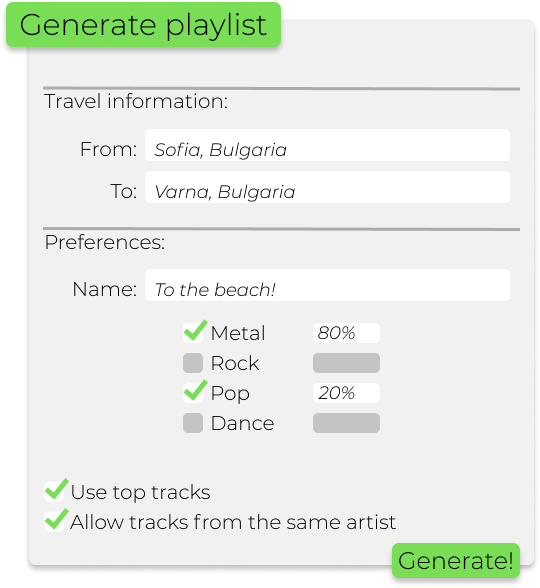
A playlist consists of a list of individual tracks (songs). Each track has an artist, title, album, duration (playtime length) and rank (a numeric value). Tracks may or may not have a preview URL to an audio stream (e.g., the users can click and play the preview part of the track).

Each playlist *must* have 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).

# Use cases

A user travelling from point A to point 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:



Note: The purpose of the above wireframe is to **only illustrate** the basic UI components and is not a guideline of how your UI must look like.

1. ***RidePal Playlist Generator*** *must* calculate the travel duration time between the starting and destination locations and combine 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).
2. The generated playlist is saved under this user’s profile, and they can start listening to it.

The application *must* offer the option to browse playlists created by other users and allow filtering by total duration and genre tags.

By default, playlists *should* be sorted by average rank descending.

The application *must* offer certain configuration over the playlist generation algorithm and *could* provide the possibility to play a preview of the tracks in a playlist.

# Project Requirements

### UI

You may look for inspiration at [Soundtrack your ride from Spotify](https://soundtrackyourride.byspotify.com/) or express your creativity! The goal is to have a place that makes it easy for a user to create a playlist and hit the road. Build a clean, simple application that gets the job done!

## 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 *must* contain at least 3 generated playlists (they can be the top 3 playlists on the platform)
* Playlists *must* be clickable/expandable to show the list of artists/tracks.
* Playlists *must* show average rank (average of its tracks) and total playtime without being opened.
* Playlists *should* be sorted by rank and *should* be filterable by **name** or **genre** or **duration**, or any combination of the three.

### 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 with 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 track list (e.g., removing or adding individual songs).

### Administration Part

System administrators can administer all major information objects in the system. On top of the regular user capabilities, the administrators have the following capabilities:

* Administrators *must* be able to edit/delete users and other administrators.
* Administrators *must* be able to edit/delete over the playlists.
* Administrators *must* be able to manually trigger Genre synchronization (if that optional requirement is implemented)

## External Services

The ***RidePal Playlist Generator*** web application will consume two public REST services 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 like Spotify and Google Play. Its API usage is free for non-commercial use and does not require any registration. For usage details please see the Appendix.

# Technical Requirements

## General development guidelines

* Use **ReactJS** (recommended), or VueJS, Angular, EmberJS or any other frontend SPA framework..
* Use **Express** (recommended), or NestJS, Koa, Fastify, or any other backend framework. Alternatively, skip backend development and use Firebase.
* Use **MariaDB** (recommended), or MongoDB, PostgreSQL or any other database system.
* All the data should be loaded from a web server using **services**.
* Create usable user interface.
* Apply proper data validation and error handling.
* Your project should pass the default **ESLint** linting configuration without any errors.
* Use **Git** as source control.
* **Documentation** of the project and project architecture (as .md file, including screenshots).

## 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 *must* 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 track list URL*), **album** (*id*, *name*, *album track list URL*)

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*,* understanding its API*,* and transforming the data from the external service into a local domain model.

We do not 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 is inadvisable to dump all tracks for “*Rock*” (~60K). Doing this programmatically will most likely exceed the free quota.

## Generation Algorithm Specification

* *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 can be +/- 5mins of the calculated travel duration.

# Optional Requirements (could)

* Take advantage of **Git branches** for writing your features.
* **Unit test** a few components.
* Integrate your app with a **Continuous Integration** server (e.g., GitLab’s own) and set up your unit tests to run on each commit to the master branch.
* Host your application’s backend in a public hosting provider of your choice (e.g., Heroku).
* Host your application’s database server in a public hosting provider of your choice.
* Implement pagination to handle displaying large amounts of data.
* Originality of the implementation (uniqueness).
* Develop a **standalone mobile client**, using React Native, Ionic, Apache Cordova or any other similar technology.

# Deliverables

* Each project *must* use **GitLab issues** and board (see Teamwork Guidelines)**.**
* Each project source code *must* be available in a dedicated GitLab/GitHub repository, with:
  + The ***complete source*** code and any ***run scripts.***
  + A ***README.md*** documentation describing how to build and run the project (or some other form of documentation).
  + ***Screenshots*** of the major application user-facing screens with some data on them.
  + **URL** of the application (*if hosted online*).
  + Images of the database relations.
  + Documentation how to create and fill the database with data.
  + Documentation how to use the main features of the project.
* 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 do not squash commits)! (See guides on best practices when working with Git in Appendix)

# Expectations

You *must* understand the system you have created.

It is OK if your application has flaws or is missing one or two must’s. What’s not OK is if you do not know what is working and what is not. Also, if you present an incomplete project as functional. Any defects or incomplete functionality *must*be properly documented and secured.

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

* What are the most important things you have 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 or had more time?

# Public Project Presentation

Each team will present their work to the trainers and students (~10 minutes). The presentation can either be a live demo, video, slide deck presentation, or whatever you feel showcases you project best (please prepare sample data). Get creative!

Each team will also have a meeting with the Telerik Academy partners, where they are expected to be able to explain the application structure, major architectural components and selected source code pieces demonstrating the implementation of key features.

# Legend

* must – The requirements marked as must are vital for the project. Implement them first.
* should – It would be nice if the requirements marked as should are implemented but they are not top priority.
* could – If, and only if, you finish early and have time to spare, you can implement the requirements marked as could. They really are optional.

# Appendix

1. [Guidelines for designing good REST API](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy)
2. [Guidelines for URL encoding](http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/)
3. [Git issue tracking docs](https://docs.gitlab.com/ee/user/project/issues/)
4. [Gitflow Workflow](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)
5. [Git commits: An Effective Style Guide](https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn)
6. [How to write a Git Commit message](https://chris.beams.io/posts/git-commit/)
7. [Microsoft Bing Maps External Service Docs](https://docs.microsoft.com/en-us/bingmaps/#pivot=main&panel=BingMapsAPI)
8. [Deezer Docs](https://developers.deezer.com/api)

## 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 its domain and do some data transformations.

### 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 [docs](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*](http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia/Mladost/1729/Alexandar%20Malinov%2031?key)[*=*](http://dev.virtualearth.net/REST/v1/Locations/BG/Sofia/Mladost/1729/Alexandar%20Malinov%2031?key=AhqpRyyyhbWVwrrxOVnvjKJuIfcPRcBWe2YS-mL4BXAVXuUH3lfx_QilDP50vOJO)*<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 docs](https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix). The Deezer API also has a Java library. You can find it [here](https://github.com/michalvich/deezer-api-client). For example, the URL to calculate the distance between The National Place of Culture and Telerik 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.

### 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 [here](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 the following format, which will fetch a list of playlists, which contain the word <GENRE> in their title.

*https://api.deezer.com/search/playlist?q*=<GENRE>

There’s a total number of existing playlists as well as links to next and previous 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 previous 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](https://designli.co/blog/5-steps-proof-concept-successful-software-development/) 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