Assignment 3 (12.5%)
RESTful API

T-213-VEFF, Web programming I, 2025-1

Reykjavik University - Department of Computer Science, Menntavegi 1, 101 Reykjavík

Deadline: Friday, March 14th 2025, 23:59

This is the third assignment in Web Programming I, with the topic of writing a RESTful backend for a Music System. **The assignment can be completed in groups between 2-3 people.**

1 Overview

In this assignment, you will develop a backend that can be used for keeping track of songs and playlists. You will provide two kinds of resources with several endpoints that follow the REST principles and best practices. Please note that Postman can be a huge asset when developing solutions like this one.

Note: Similar to Assignment 2, this task might seem overwhelming at first. Start by designing the API following the lecture slides for L15/16 and in the friday lecture on February 28th (writing down the HTTP methods and URLs for each endpoint). Then, implement easy endpoints first (e.g., read requests are typically easiest), and use dummy data in the beginning (e.g., hard-coded arrays in the starter pack).

2 Resources

For this project, we require two resource types: **songs** and **playlists**. **Songs** have title, and an artist. **Playlists** are a way to manage songs in different playlists (the same song can be used in multiple playlists).

A **song** has the following attributes:

- ***id*** - A unique number identifying each song.
- ***title*** - The title of the song, provided as a string.
- ***artist*** - The name of the artist(s) of the song, given as a string.

A **playlist** has the following attributes:

- ***id*** - A unique number identifying each playlist.
- ***name*** - The name of the playlist, provided as a string.
- ***songIds*** - An array of song ids, which refer to songs in the songs resource.

You are encouraged to implement these resources in your backend as you see fit. However, it's essential that the backend can return and manage these attributes in the specified format.

3 Endpoints

The following endpoints shall be implemented for the **songs**.

1. **Read** all songs

   Returns an array of all songs. For each song, the id, the title, and the artist are included in the response. Additionally, a filter should be provided as a query parameter (the only allowed query parameter is 'filter') that returns only those songs that contain the given string in the title or artist name (case-insensitive). If no songs are found, an empty array is returned.

2. **Create** a new song

   Creates a new song. The endpoint expects a title and artist in the body of the request. Duplicate songs are **not** allowed in the system (i.e. songs with the same title and artist can only be added once - case-insensitive). The request shall fail if an attempt is made to register a song that already exists. The request, if successful, shall return the new song (all properties, including id and all information).

3. **Partially update** a song

   Partially updates an existing song. All properties can be updated except the id. Endpoint accepts an id as part of a path. All properties specified in the request are updated. The request, if successful, returns the updated layer. If the id is invalid or the song is not found, an appropriate error message is returned.

4. **Delete** a song

   Deletes a song that already exists. The request shall fail if any playlists are using this song. If the id is invalid or the song is not found, an appropriate error message is returned. The request, if successful, returns all the properties of the deleted song.

The following endpoints shall be implemented for the **playlists**.

1. **Read** all playlists

   Returns an array of all playlists (with all attributes).

2. **Read** a specific playlist

   Returns a requested playlist with all its attributes, including the songs in the playlist, as an array of objects with full information about each song. Accepts an id as part of a path. If the id is invalid or the playlist is not found, an appropriate error message is returned.

3. **Create** a new playlist

   Creates a new playlist. The endpoint only expects a name in the body of the request. The id shall be generated automatically and when a playlist is created, songIds shall be an empty array for a newly created playlist. Duplicate playlist names are **not** allowed. The request, if successful, shall return the new playlist (all properties, including id).

4. **Add song to** an existing playlist

   Adds a given song to a given playlist. Accepts a playlist id and a song id as part of the URL. If the ids are invalid, the playlist or song is not found, an appropriate error message is returned. If the song is already in the playlist, an error is returned. Otherwise, the song is added to the playlist. The request, if successful, shall return the playlist with all its attributes (similar to what is returned in "Read a specific playlist").

4 Music System - frontend

For this assignment, we use an online frontend that keeps track of songs and playlists. We deployed the frontend at 8 urls, one for each group (with Akureyri and Austurland sharing a backend). These are:

- Students in group 1 should use: https://2025-veff-assignment3-group1.netlify.app/
- Students in group 2 should use: https://2025-veff-assignment3-group2.netlify.app/
- Students in group 3 should use: https://2025-veff-assignment3-group3.netlify.app/
- Students in group 4 should use: https://2025-veff-assignment3-group4.netlify.app/
- Students in group 5 should use: https://2025-veff-assignment3-group5.netlify.app/
- Students in group 6 should use: https://2025-veff-assignment3-group6.netlify.app/
- Students in HMV should use: https://2025-veff-assignment3-hmv.netlify.app/
- Students in Akureyri and Austurland should use: https://2025-veff-assignment3-unak.netlify.app/

Whenever these websites are not available, it means that the frontend is down and cannot be reached. If this happens, please contact us on Piazza as soon as possible.

Hint: You can use this frontend to help you develop your endpoints. But please note that not all endpoints that are required in this assignment are use by the frontend, also the frontend works well to try "success" calls, and some error handling but for detailed error handling you will need to follow the requirements and think "outside" of the current frontend.

5 Requirements

The following requirements/best practices shall be followed:

1. The frontend application provided above, in section 4, needs to work with the application.
2. You should opt to use arrow functions when possible, do not use “var” and use inbuilt JavaScript functions over generic for-loops etc.
3. You should comment your code, explaining all functionality.
4. The application shall adhere to the REST constraints.
5. The best practices from L15/16 shall be followed. This means for example that:
    a. Plural nouns shall be used for resource collections.
    b. Specific resources shall be addressed using their ids, as a part of the resource URL.
    c. Sub-resources shall be used to show relations between songs and playlists, as stated in Section 3.
    d. JSON shall be used as a request/response body format.
    e. The HTTP verbs shall be used to describe CRUD actions. The safe (for GET) and idempotent (for DELETE and PATCH) properties shall be adhered to.
    f. Appropriate HTTP status codes shall be used for responses. 200 should be used for successful GET, DELETE and PATCH requests, 201 for successful POST requests. In error situations, 400 shall be used if the request was not valid, 404 shall be used if a resource was requested that does not exist, or if a non-existing endpoint is called. 405 shall be used if a resource is requested with an unsupported HTTP verb (e.g., trying to delete all songs)
    g. You are NOT required to implement HATEOAS/Links.
6. The application/backend shall be served at http://localhost:3000/api/v1/. In case you have issues running your backend on port 3000, contact us - do not just change the port in the solution code.
7. The application shall be written as a Node.js application. The package.json file included in the "starter pack" should be used to develop the project. You are not allowed to edit the package.json, it should already contain a list of all required packages you need for this project.
8. The application shall be started using the command *npm start*.
9. The application is only permitted to use in-built modules of Node.js, as well as Express.js, body-parser, and cors[1].
10. You are not supposed to/allowed to add persistence (a database, file storage, or similar) to the assignment.
11. There are no restrictions on the ECMAScript (JavaScript) version.

[1]The cors module enables cross-origin resource sharing (CORS). It makes sure that requests are not blocked in case you try out your backend using a browser.

6 Starter Pack

In the supplementary material to this assignment, you will find a sample Node.js project (a package.json file and an index.js file). The index.js currently only includes some structure and two variables that contain examples for the resource format defined in Section 2. You may change the internal representation of the resource and/or add additional data structures as needed. You should extend the sample code to include your backend code. The dependencies to express, body-parser, and cors are already defined in the package.json - you can simply install the modules by running npm install in your project directory.

7 Starting the backend project in "development mode"

1. Navigate to the **assignment3_starter_pack** folder in a terminal window.
2. Install the NPM packages listed in ***package.json*** by running ***npm install*** command. This will fetch all the required packages needed to run the project.
3. To start up the process, you can run the ***npm run dev*** command to start the process in developer mode. Which in our case, means that the process will be watching ***index.js*** for changes. When the code is updated, the process will automatically restart to reflect the changes made to the code. The process will print out a message telling your it has started:

   ```
   > assignment_3@0.0.1 dev
   > nodemon index.js

   [nodemon] 3.1.9
   [nodemon] to restart at any time, enter ‘rs‘
   [nodemon] watching path(s): *.*
   [nodemon] watching extensions: js,mjs,json
   [nodemon] starting ‘node index.js‘
   Server running on http://localhost:3000
   ```

4. Your backend is now running http://localhost:3000 and can be accessed via Postman, cURL, or the front end website.

8 Submission

The assignment is submitted through Gradescope. Submission should **only** contain the following files:

- **index.js** containing all of your code for the RESTful API
- (optional*) **assignment3_postman_collection.json**

Do **not** include the *node_modules* folder, *package.json*, *package-lock.json* or any other file. Submitting extra files of folder will result in point deduction. **Submissions will not be accepted after the deadline.**

*Bonus points are available for supplying a complete Postman collection along with your API. See collection requirements for bonus points in Section 9.

9 Grading and point deductions

Below you can see the criteria for grading, this list is not exhaustive but gives you an idea of how grading will be done for the project

Critera	**Point deduction**
All API Endpoint Autograder tests are passed in Gradescope	For failed tests, up to -10 points
Code is commented, and all endpoints are well explained	For lack of descriptive comments or endpoint explanations, up to -2 points. Each endpoint should have a comment detailing its functionality.
Syntax issues (e.g. using "var", regular for-loops, not using arrow functions, using the "promise" syntax instead of async/await)	Depending on severity, up to -2 points
Application is served at `http://localhost:3000/api/v1/`	If the application is not served at the specified URL, -1 point.
Postman collection (2 bonus points). Collection requirements: • The collection includes all of the API endpoints with descriptive names. • All requests in the collection include the required payloads where applicable All request return a successful response You do not have to include requests producing each error of each endpoint, only successful requests. • You can export your Postman collection to a .json file, please include that in your submitted .zip file Here are some instructions on how to get started.	You can get up to 2 bonus points, but it will be lower if some requests are missing, requests do not work, etc.