SC-T-213-VEFF - Web Programming I

Project Assignment 3 – RESTful Backend

Reykjavik University | Deadline: **11th March 2021, 23:59**

The topic of this assignment is: **Writing a RESTful Backend**.

1 Overview

In this assignment, you will develop a backend that can be used for board/task applications. The idea is the same as in project 2 and, in fact, your backend needs to be compatible with the a modified solution of assignment 2[1]. You will provide two kinds of resources with several endpoints that follow the REST principles and best practices.

This assignment can be done in groups of **one to three students**. No groups of 4 or more students are permitted.

Note: Similar to Assignment 2, this task might seem overwhelming at first. Start by designing the API following the lecture slides for L15/16 (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 sample project).

2 Resources

For this assignment, we require two resource types: *boards* and *tasks*. Boards represent containers for tasks (imagine a number of post-it notes attached to a blackboard/whiteboard). Tasks then represent individual items that need to be done or considered. This is a common setup for project management applications such as Trello[2]. A task is *always* associated with a board, it cannot exist without it (a task "belongs" to a board).

A board has the following attributes:

1. *id* - A unique number/string identifying each board.
2. *name* - A string providing a heading/title for the board.
3. *description* - A string describing the board.
4. *tasks* - An array associating tasks with the board. This array is initially empty when a board is created.

A task has the following attributes:

1. *id* - A unique number/string identifying each task.

[1] Provided as supplementary material, will be published Friday.

[2] https://trello.com

2. *taskName* - A string describing the task.

3. *boardId* - The id of the board to which the task is attached.

4. *dateCreated* - An ISO date string describing when the task was created. The date string returned by the backend shall be in milliseconds elapsed since January 1, 1970, 00:00:00 UTC.

5. *archived* - A boolean flag describing whether or not a task is archived (We will support both deleting and archiving of tasks).

You are free to implement these two resources as you like in your backend. However, the backends need to return the required attributes in the required format. For instance, you could implement the tasks attribute in a board as a Map, but then make sure to return an array.

3 Endpoints

The following endpoints shall be implemented for the boards:

1. **Read** all boards
   Returns an array of all boards. For each board, only the id, name, and description is included in the response.

2. **Read** an individual board
   Returns all attributes of a specified board.

3. **Create** a new board
   Creates a new board. The endpoint expects at least the name and description. The description may be an empty string, the name may not be empty. However, duplicate board names are permitted. The id shall be auto-generated (i.e., not provided in the request body). Similarly, the tasks array shall be initialised to an empty array. The request, if successful, shall return the new board (all attributes, including id and tasks array).

4. **Update** a board
   (Completely) Updates an existing board. The updated data is expected in the request body (all attributes, excluding the id and the tasks array). The request shall fail if there are existing tasks for the board that are not archived. The request, if successful, returns all attributes of the board.

5. **Delete** a board
   Deletes an existing board. The request shall fail if there are existing tasks for the board that are not archived. The request, if successful, returns all attributes of the deleted board.

6. **Delete** all boards
   Deletes all existing boards. The request also deletes all tasks for all existing boards. The request, if successful, returns all deleted boards (all attributes), as well as their tasks (as a part of the tasks attribute).

The following endpoints shall be implemented for the tasks:

1. **Read** all tasks for a board
   Returns an array of all tasks (with all attributes) for a specified board.

Additionally, providing the “sort” query parameter with either one of the values “taskName”, “dateCreated” or “id” will cause the returned array to be sorted (by taskName, dateCreated, or by id) in ascending order.

2. **Read** an individual task
   Returns the specified task (with all attributes) for the selected board.

3. **Create** a new task
   Creates a new task for a specified board. The endpoint expects only the taskName attribute in the request body. The id (unique, non-negative number) shall be auto-generated. Similarly, the dateCreated attributed is set to the current date, and the boardId parameter is taken from the request URL. The request, if successful, shall return the new task (all attributes, including id). Furthermore, the id of the new task shall be added to the tasks array in the corresponding board.

4. **Delete** a task
   Deletes an existing task for a specified board. The request, if successful, returns all attributes of the deleted task. The task is removed from the corresponding board.

5. **Partially update** a task for a board
   Updates any of the three attributes taskName, archived and/or boardId of a specified task. If the boardId is changed, the tasks arrays in both the original and target boards have to be modified as well. The updated task is returned.

4 Requirements

The following requirements/best practices shall be followed:

1. The (unmodified) frontend code provided as supplementary material needs to work with the application.

2. The application shall adhere to the REST constraints.

3. The best practices from L15/16 shall be followed. This means 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. Ids shall not be sent in the query part of the URL or the request body.

   (c) Sub-resources shall be used to show relations between tasks and boards.

   (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 PUT) properties shall be adhered to.

   (f) Appropriate HTTP status codes shall be used for responses. 200 should be used for successful GET, DELETE and PUT 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. 405 shall be used if a resource is requested with an unsupported HTTP verb (e.g., trying to delete all tasks), or if a non-existing endpoint is called.

   (g) You are **NOT** required to implement HATEOAS/Links.

4. 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.
5. The application shall be written as a Node.js application. A package.json file shall be part of the project, including also the required dependencies.
6. The application shall be started using the command *node index.js*.
7. The application is only permitted to use in-built modules of Node.js, as well as Express.js, body-parser, and cors[3].
8. The application shall additionally be deployed on Heroku. As a part of your hand-in, provide a text file that contains the URL of your deployed backend.
9. Persistence is not part of the assignment.
10. There are no restrictions on the ECMAScript (JavaScript) version.

5 Sample Project

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 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 (additional files are of course permitted). 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.

Submission

The lab is submitted via Canvas. Submit a zip file containing your entire node project. Do **NOT** include the *node_modules* folder and the *package-lock.json*.

[3] The *cors* module enables cross-origin resource sharing (CORS). This is not required for this assignment, but it makes sure that requests are not blocked in case you try out your backend using a browser.