SC-T-213-VEFF - Web Programming I

Project Assignment 3 – RESTful Backend

Reykjavik University | Deadline: **18th March 2019, 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 various weather applications. You will provide two kinds of resources with several endpoints that follow the REST principles and best practices.

This assignment can either be done **alone** or as a **group of 3 students**.

Note: Similar to Project 2, this task might seem overwhelming at first. Start by designing the API following the lecture slides for L13/14 (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., the hard-coded array of stations and observations in the sample project).

2 Resources

For this assignment, we require two resource types: *stations* and *observations*. Stations represent individual weather stations, observations represent weather observations at different points in time. An observation is *always* associated with a station, it cannot exist without it (an observation "belongs" to a station, or "is made" at a station).

A station has the following attributes:

1. *id* - A unique number identifying each station
2. *description* - A string describing the station
3. *lat* - A number describing the latitude (geographical coordinate) of the station
4. *lon* - A number describing the longitude (geographical coordinate) of the station
5. *observations* - An array associating observations with a station

An observation has the following attributes:

1. *id* - A unique number identifying each observation
2. *date* - A number following the date standard (Unix Time Stamp - Milliseconds since Jan 1st, 1970 UTC) describing the time and date at which the observation was made
3. *temp* - A number describing the temperature in degrees centigrade

4. *windSpeed* - A number describing the wind speed in meters per second.
5. *windDir* - A string describing the direction of the wind
6. *prec* - A number describing the precipitation (downpour/rain/snow) in millimeters.
7. *hum* - A number describing the humidity in percent.

3 Endpoints

The following endpoints shall be implemented for the stations:

1. **Read** all stations
   Returns an array of all stations. For each station, only the description and the id is included in the response.
2. **Read** an individual station
   Returns all attributes of a specified station.
3. **Create** a new station
   Creates a new station. The endpoint expects all attributes apart from the id in the request body. The id shall be auto-generated. The request, if successful, shall return the new station (all attributes, including id).
4. **Delete** a station
   Deletes an existing station. The request also deletes all observations for the given station. The request, if successful, returns all attributes of the deleted station (including all observations in the observations attribute).
5. **Update** a station
   (Completely) Updates an existing station. The updated data is expected in the request body (excluding the id). The request, if successful, returns all updated attributes of the station.
6. **Delete** all stations
   Deletes all existing stations. The request also deletes all observations for all existing stations. The request, if successful, returns all deleted stations (all attributes), as well as their observations (as a part of the observations attribute).

The following endpoints shall be implemented for the observations:

1. **Read** all observations for a station
   Returns an array of all observations (with all attributes) for a specified station.
2. **Read** an individual observation
   Returns all attributes of a specified observation (for a station).
3. **Create** a new observation
   Creates a new observation for a specified station. The endpoint expects all attributes apart from the id and the date in the request body. The id (unique, non-negative number) and the date (current date) shall be auto-generated. The request, if successful, shall return the new observation (all attributes, including id and date).

4. **Delete** an observation
   Deletes an existing observation for a specified station. The request, if successful, returns all attributes of the deleted observation.

5. **Delete** all observations for a station
   Deletes all existing observations for a specified station. The request, if successful, returns all deleted observations (all attributes).

4 Requirements

The following requirements/best practices shall be followed:

1. The application shall adhere to the REST constraints.
2. The best practices from L13/14 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 resources
   (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 update an observation), or if a non-existing endpoint is called.
   (g) You are **NOT** required to implement HATEOAS/Links.
3. Basic input parameter validation shall be performed for numbers. For string parameters, no validation is required.
   (a) For numbers, only parameters that can be converted to a number are allowed. NaN is not allowed.
   (b) Latitudes range between -90 and 90, longitudes between -180 and 180.
   (c) Precipitation cannot be negative.
   (d) Humidity is between 0 and 100.
4. The application/backend shall be served at *http://localhost:3000/api/v1/*
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 and body-parser.

8. Persistence is not part of the assignment.

9. 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 can extend it to include your backend code (additional files are of course permitted). The dependencies to Express.js and body-parser 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*.