SC-T-213-VEFF - Web Programming I

Project Assignment 3 – RESTful Backend

Reykjavik University | Deadline: **16th March 2020, 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 event booking applications. 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., the hard-coded array of events and bookings in the sample project).

2 Resources

For this assignment, we require two resource types: *events* and *bookings*. Events represent one-time arrangements (e.g., concerts, movie screenings or similar), bookings represent a registration of a number of slots/seats/spaces a person has made for a specific event (e.g., reserving 3 seats at a movie screening). A booking is *always* associated with an event, it cannot exist without it (a booking "belongs" to an event).

An event has the following attributes:

1. *id* - A unique number identifying each event.
2. *name* - A string providing a heading/title for the event.
3. *description* - A string describing the event.
4. *location* - A string describing the venue at which the event takes place.
5. *capacity* - A number describing the amount of spaces available (e.g., the number of seats at a movie screening).
6. *startDate* - A date describing the starting date and time of the event. As an input, the date is expected in Unix Time Stamp format (Milliseconds since Jan 1st, 1970 UTC). As a return value, any valid date format is accepted.

7. *endDate* - A date describing the end date and time of the event. As an input, the date is expected in Unix Time Stamp format (Milliseconds since Jan 1st, 1970 UTC). As a return value, any valid date format is accepted.

8. *bookings* - An array associating bookings with the event. This array is initially empty when an event is created.

A booking has the following attributes:

1. *id* - A unique number identifying each booking.

2. *firstName* - A string describing the first name of the person creating the booking.

3. *lastName* - A string describing the last name of the person creating the booking.

4. *spots* - A number describing the amount of spots required for the booking.

5. *email* - A string describing the contact email address for the booking.

6. *tel* - A string describing a contact telephone number for the booking.

3 Endpoints

The following endpoints shall be implemented for the events:

1. **Read** all events
   Returns an array of all events. For each event, only the name, id, capacity, startDate and endDate is included in the response.

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

3. **Create** a new event
   Creates a new event. The endpoint expects at least the name, capacity, startDate and endDate parameter. Description and location are optional. The id shall be auto-generated (i.e., not provided in the request body). Similarly, the bookings array shall be initialised to an empty array. The request, if successful, shall return the new event (all attributes, including id and bookings array).

4. **Update** an event
   (Completely) Updates an existing event. The updated data is expected in the request body (all attributes, excluding the id and the bookings array). The request is only successful if there are no existing bookings for the event. The request, if successful, returns all attributes of the event.

5. **Delete** an event
   Deletes an existing event. The request is only successful if there are no existing bookings for the event. The request, if successful, returns all attributes of the deleted event.

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

The following endpoints shall be implemented for the bookings:

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

2. **Read** an individual booking
   Returns all attributes of a specified booking (for an event).

3. **Create** a new booking
   Creates a new booking for a specified event. The endpoint expects firstName, lastName, spots and tel and/or email (meaning that you can provide tel and email, only tel, or only email) attributes in the request body. The id (unique, non-negative number) shall be auto-generated. The request is only successful if there are still enough spots left in the corresponding event. The request, if successful, shall return the new booking (all attributes, including id). Furthermore, the id of the new booking shall be added to the bookings array in the corresponding event.

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

5. **Delete** all bookings for an event
   Deletes all existing bookings for a specified event. The request, if successful, returns all deleted bookings (all attributes). The bookings array for the corresponding event is emptied.

4 Requirements

The following requirements/best practices shall be followed:

1. The application shall adhere to the REST constraints.

2. 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 bookings and events.

   (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 a booking), 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, as discussed below. 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) Date parameters shall adhere to the UNIX time stamp format. Make sure they are a number (same validation as for number parameters). Furthermore, the start and end dates for an event shall be in the future, and the end date shall be later than the start date.

(c) The capacity of an event has to be larger or equal 0.

(d) The spots for a booking have to be larger 0.

(e) Spots cannot be larger than the remaining capacity of an event.

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, body-parser, and cors[1].
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*.

[1] 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.