SC-T-213-VEFF - Web Programming I

Project Assignment 4 – Endpoint Testing

Reykjavik University | Deadline: **30th March 2020, 23:59**

The topic of this assignment is: **Adding authentication and writing endpoint tests for a RESTful Backend**.

1 Overview

In this assignment, you will enhance a backend with basic authentication and write endpoint tests that test a modified solution to Assignment 3 (event & booking backend).

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

The assignment builds intentionally on a rather complex project - several of the used concepts are beyond the lecture scope. Note that to solve it, you do not need to fully understand the server code, only the overall structure. This structure is similar to your solution, just the individual endpoints are programmed differently to include persistence. This situation is intended to reflect real-life practice, since you are likely to develop something (a sub-system, tests, a frontend) using another system which you do not know or fully understand. Instead, you have to rely on documentation (such as this assignment description and the description for Project 3) and communication (asking questions on Piazza) - use these resources and avoid making assumptions. To get started, you are encouraged to try out all endpoints using Postman. Please also watch the video named "Assignment 4 Introduction" on Echo360.

2 Setup

In the supplementary material, you find a modified solution to Assignment 3 that uses persistence (MongoDB). The project supports 8 different endpoints for *events* and *bookings*.

They follow the same requirements as in Assignment 3 (what kind of input they expect and what they return), with the following difference: the id attributes start with an underscore (*_id* instead of *id*) and expect a string input. The endpoints are as follows:

1. **GET /api/v1/events**
2. **GET /api/v1/events/:eventId**
3. **POST /api/v1/events**
4. **DELETE /api/v1/events/:eventId**
5. **GET /api/v1/events/:eventId/bookings**
6. **GET /api/v1/events/:eventId/bookings/:bookingId**

7. **POST /api/v1/events/:eventId/bookings**

8. **DELETE /api/v1/events/:eventId/bookings/:bookingId**

In the *test* sub-folder, you will find a file called *index.test.js*, already set up so that you can start writing your tests. The current setup makes sure that you have exactly one event and one booking (belonging to that event) before each test. This way, you have enough resources to call all endpoints (e.g., also delete requests), and know exactly what the return values will be (e.g., you know the description of the event, you know what a get all events request should return). Currently, the file includes a single test. Note in particular that the ids are different in each test since they're auto-generated by MongoDB. The id variables are stored in the *eventId* and *bookingId* variables.

A number of npm scripts have been included, so that you can easily run the server, the tests, and the debugger:

- *npm start* starts the server
- *npm test* runs all tests in *test/app.test.js* (and all other files in that folder that end on *.test.js*)
- *npm debug* starts the server in debug mode on port 9229 (compatible with VSCode *Launch via NPM*).

To run the server successfully (and your tests as well), you will need a running instance of MongoDB.

3 Task

Your task in this assignment is to add authentication to an already existing backend, and then write a number of endpoint tests. Adding authentication is awarded 2 points, writing the tests the remaining 8 points. Note however that the tests include tests for authentication.

3.1 Adding HTTP Basic Authentication

Two endpoints exist that delete resources from the database. These are:

1. **DELETE /api/v1/events/:eventId**

2. **DELETE /api/v1/events/:eventId/bookings/:bookingId**

For these two endpoints, you shall add HTTP basic authentication so that only an authenticated user is able to successfully execute them. The following requirements apply:

1. The backend shall expect HTTP Basic Auth. This means that a successful request includes the *Authorization* header with method Basic and the *username:password* string encoded as Base64.

2. The only supported user is *admin* with password *secret*

3. In the backend, the passwords shall be compared using SHA256 hashes. That is, the password shall not be visible in clear text in the code.

4. You are allowed to use the express-basic-auth module to support authentication, and the js-sha256 module to support SHA256 hashing.

3.2 Endpoint Tests

(A total of six tests is required here)
For each endpoint apart from the DELETE endpoints, write a test that captures the success case (the request succeeds, resulting a 2xx response code). For endpoints that return arrays, assert the following:

- The status code shall be as expected (e.g., 200 for endpoint 1)
- The response body is in json format
- The return type is an array
- The array contains the right amount of elements

For endpoints that return individual objects, assert the following:

- The status code shall be as expected (e.g., 200 for endpoint 1)
- The response body is in json format
- The response body is as expected
  - The right attributes are in the body
  - No additional attributes are in the body
  - The attributes have the expected values (you may exclude checking the dates for events)

Note that for POST requests, you cannot check whether the value of the object's id is correct, just that it exists.
(A total of two tests is required here)
Write two tests for the **DELETE /api/v1/events/:eventId/bookings/:bookingId** endpoint. The first test shall test the success case. That is, a DELETE request with the right authentication information is successful. It is sufficient to assert that the status code of the response is 200. The second test shall assert that a request with the wrong or missing authentication information is not successful. It is sufficient to assert that the status code of the response is NOT 200.

Note: For obtaining a Base64 encoding in Node.js, you can use the following command:

```
Buffer.from("stringToBeHashed", "binary").toString("base64");
```

4 Requirements

The following requirements/best practices shall be followed:

1. The backend code shall remain unchanged apart from adding authentication to the required endpoints in the index.js file.
2. No extra files shall be added. All test code shall be added in test/index.test.js.
3. The tests shall be written using the mocha, chai and chai-http modules. The test file also includes a dependency to mongoose and to model files used in the backend code - these are required to set up the data before each test (already done in the file).
4. There are no restrictions on the ECMAScript (JavaScript) version.

Submission

The lab is submitted via Canvas. Submit a zip file containing your test file (*index.test.js*) and the modified *index.js* file. Do **NOT** include the *node_modules* folder and the *package-lock.json*.