SC-T-213-VEFF - Web Programming I

Project Assignment 4 – Endpoint Testing

Reykjavík University Deadline: **1st April 2019, 23:59**

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

1 Overview

In this assignment, you will write endpoint tests that test a modified solution to Project 3 (weather back-end).

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

Even though the deadline is on 1st April, this assignment is no joke. It 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 understand the server code - it can be treated as a black box. Indeed, you are likely to face this situation regularly in your professional life: Having to develop something (a sub-system, tests, a frontend) using another system which you do not know (or even have access to). 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 "Project 4 Introduction" on Echo360.

2 Setup

In the supplementary material, you find a modified solution to Project 3 that uses persistence (MongoDB). The project supports 8 different endpoints for *stations* and *observations*. The first seven follow the same requirements as in Project 3 (what kind of input they expect and what they return), with the following differences: the id attributes start with an underscore (*_id* instead of *id*) and expect a string input; the date has been removed from observations (to make your life easier when writing tests). The endpoints are as follows:

1. **GET /api/v1/stations**
2. **GET /api/v1/stations/:id**
3. **POST /api/v1/stations**
4. **GET /api/v1/stations/:stationId/observations**
5. **GET /api/v1/stations/:stationId/observations/:obsId**
6. **POST /api/v1/stations/:stationId/observations**
7. **DELETE /api/v1/stations/:stationId/observations/:obsId**

The eight endpoint has additionally been modified to require authorisation using HMAC request signing:

8. **DELETE /api/v1/stations/:id**

In the test sub-folder, you will find a file called *app.test.js*, already set up so that you can start writing your tests. The current setup makes sure that you have exactly one station and one observation 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 station, you know what a get all stations request should return). Currently, the file includes a single test (that tests that 1 == 1).

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 Endpoint Tests

Your task in this assignment is to write a number of endpoint tests. We divide these tests into two classes: regular endpoint tests (described in Section 3.1) and advanced endpoint tests (described in Section 3.2). The regular endpoint tests account for 7 out of 10 points in this assignment. The remaining three points are awarded for successful advanced endpoint tests.

3.1 Regular Endpoint Tests

(A total of ten tests is required here)
For endpoints 1-7 (inclusive), write a test each that captures the success case (the request succeeds, resulting a 2xx response code). For each of these tests, assert the following:

- The status code shall be as expected (e.g., 200 for endpoint 1).
- The response body is as expected
  - The right elements and attributes are in the body
  - No additional attributes are in the body
  - The attributes have the expected values
  - The body is of the right type (array for multiple resources, object for single resources)
- The content-type header of the response is "application/json"

For endpoint 6, write a test that captures the following two failure cases:

- Leaving out any of the required body attributes results in a 400 response code

- Providing a humidity (hum) of higher than 100.0 (and with all other attributes being correct) results in a 400 response code

Finally, write a test capturing that a call to /api/v1/stations with an unsupported HTTP verb results in an 405 response code.

3.2 Advanced Endpoint Tests

(A total of two tests is required here)
Endpoint 1 (get all stations) supports a filter query that limits the returned stations to only those that have a certain description. For instance, a GET request to `/api/v1/stations?description=Reykjavik` would return all stations that have *Reykjavik* as their description attribute. Write a test that demonstrates that it is possible to run a (MongoDB) injection attack for this endpoint/query. Hint: It is possible to introduce special Mongo constructs that cause the query to be inverted ("return all stations that do NOT have *Reykjavik* as their description attribute).

A regular request to Endpoint 8 returns a 401 return code with message *unauthorised*. The author of this API has decided that deleting a station needs authorisation, as accidental deletion of a station has substantial impact (all observations ever made at that station would disappear as well). The endpoint requires a basic (and in practice not very secure) HMAC request signing approach:

- The string that is hashed is the request method (in upper case) and the endpoint path (in lower case), separated by a space. For example, when trying to delete station 5, the string would be "DELETE /api/v1/stations/5".
- The secret used in the hashing is *mysecret* (all lower case).
- The hash is created using the SHA256 function.
- The resulting hash is added to the Authorization header of the request (without any additional characters/preamble).

In lines 27 to 36 in app.js, you can also see how the hash is re-calculated on the server side and compared to the authorization header. Write a test that runs a successful request to Endpoint 8 (the request succeeds, resulting a 2xx response code). It is sufficient to assert that the response code is correct. For testing in Postman, you can use tools such as `https://www.freeformatter.com/hmac-generator.html` to calculate the HMAC hash.

4 Requirements

The following requirements/best practices shall be followed:

1. Only app.test.js shall be modified. The application code shall remain unchanged.
2. The tests shall be written using the mocha, chai and chai-http modules. js-sha256 or similar modules may be used for request signing. The test file also includes a dependency to mongoose and to model files used in the application code - these are required to set up the data before each test (already done in the file).
3. There are no restrictions on the ECMAScript (JavaScript) version.

Submission

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