Roberto Morassi del Blanco

A0265239X

<https://github.com/rmorassi/CS3219-OTOT-B>

**i) Instructions on how to run the API locally including Postman calls used to demonstrate a working API (B1/B2.2)**

The API I created is an interface to store the information of students residing at a hall of residence (hostel). The API has been named ‘studentRooster’.

**Commands to run locally (to be run within app/):**

1. `npm install`
2. `npm start`
3. Server is now running locally at ‘http://localhost:8080’ and the API running on ‘http://localhost:8080/api/studentRooster’

**Now I will demonstrate how the API runs including GET, POST, PUT, DELETE API calls:**

1. Note that in the following postman calls, the variable ‘endpoint’ is set to ‘http://localhost:8080’ to showcase how the API can run locally, but can easily be swapped for the deployed endpoint without modifying the functionality of the API.
2. A GET request to ‘/api/studentRooster’ shows all the students stored in the student rooster.

Graphical user interface, text, application, email

Description automatically generated

1. A POST request to ‘/api/studentRooster’ with the relevant parameters needed for creating the student record will add the student to the database.

Graphical user interface, text, application, email

Description automatically generated

Note that from now on, we set the variable ‘ID of student’ to ‘636273b5120c3f94ea900f76’ to refer to the newly created record.

1. A GET request to ‘/api/studentRooster/*studentID*’ shows the record of a particular student.

Graphical user interface, text, application, email

Description automatically generated

1. A PUT request to ‘/api/studentRooster/*studentID*’ along with the relevant fields that want to be modified will modify the data of a particular student, leaving unmodified data untouched.

Graphical user interface, text, application, email

Description automatically generated

1. A DELETE request to ‘/api/studentRooster/*studentID*’ will delete the entire record of a particular student.

Graphical user interface, text, application, email

Description automatically generated

**Demonstrating the error resiliency of the API:**

1. When making a POST request to create a new student record, if the user does not include all required fields, an error message is returned.

Graphical user interface, text, application, email

Description automatically generated

1. When making a GET request to get the records of a specific student, if the student id does not exist, null is returned.

Graphical user interface, text, application, email

Description automatically generated

1. When making a PUT request to modify the records of a specific student, if the user includes data that does not match the required type, an error message is returned (roomNumber should be a number).

Graphical user interface, text, application, email

Description automatically generated

**ii) Instructions on how to access the deployed API (B1/B2.2)**

No commands need to be run as the deployed API is already on the web. Simply access ‘https://otottaskb-367404.as.r.appspot.com’ to access the server. Note that the calls to the API endpoint need to be made to ‘https://otottaskb-367404.as.r.appspot.com/api/studentRooster’ or ‘https://otottaskb-367404.as.r.appspot.com/api/studentRooster/*studentId*’ depending on the instruction to be executed.

All the HTTP requests shown above for localhost, run identically in the deployed endpoint. During the testing shown above, the endpoint does not crash.

**iii) Instructions on run tests locally and via CI tool (B2.1)**

I made tests to test the functionality of all HTTP methods.

The screenshot below shows the tests running (and passing) locally on my machine.

Text

Description automatically generated

The screenshot below shows the Continuous Integration tests running and passing on the GitHub Actions – my CI tool of choice. The tests are run every time somebody pushes to the branch ‘main’. Please refer to the file ‘app-ci.yml’ in ‘.github/workflows’ where I created the Continuous Integration workflow. In this file, I specify the version of NodeJS to use, and different commands that need to be run before testing begins such as ‘npm ci’ to install dependencies. We can see that the tests pass given the green tick on the left-hand side.

A screenshot of a computer

Description automatically generated with medium confidence

The screenshot below shows the Continuous Deployment workflow I created to run on GitHub Actions – my CD tool of choice. Please refer to the file ‘app-cd.yml’ in ‘.github/workflows’ where I created the Continuous Deployment workflow. In this file, I specify when to deploy (once the CI finishes and succeeds) and different parameters such as the Google Cloud Compute credentials. Note that the deployment is done to Google App Engine (to the same endpoint specified in the previous exercise). We can see that the deployment succeeds given the green tick on the left-hand side.

A screenshot of a computer

Description automatically generated with medium confidence

iv) Set up frontend (B3)

v) A brief explanation of what your serverless function does

vi) Screenshots of the front-end interaction with your serverless function