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)

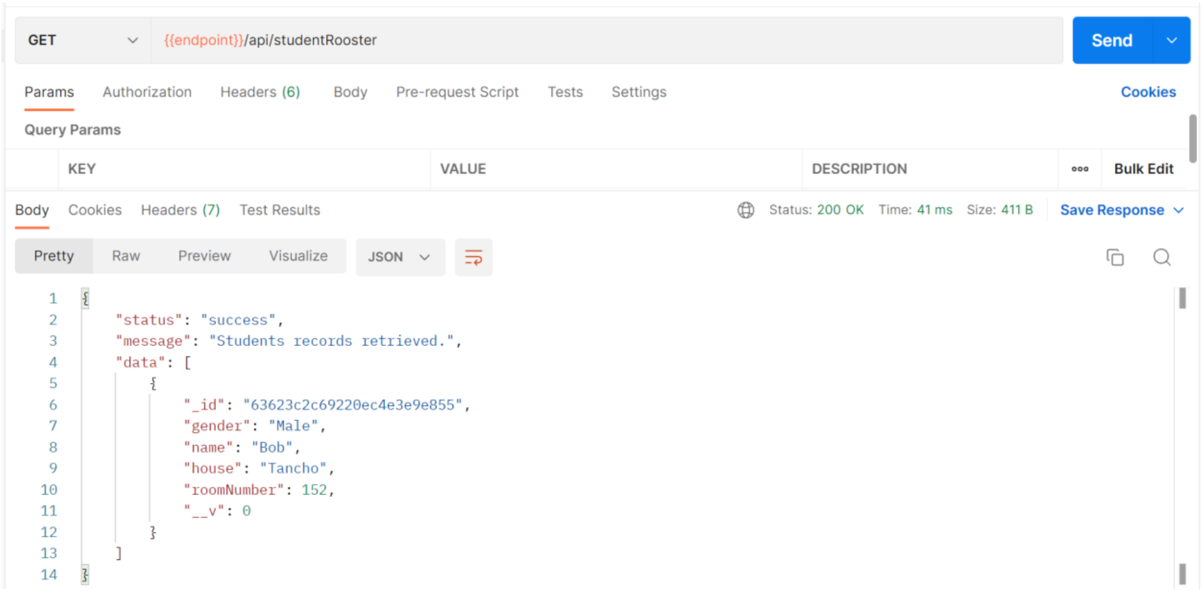
The API I created is an interface to store the information of students residing at a hall of residence (hostel). The API has been named 'studentRooster'.

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.

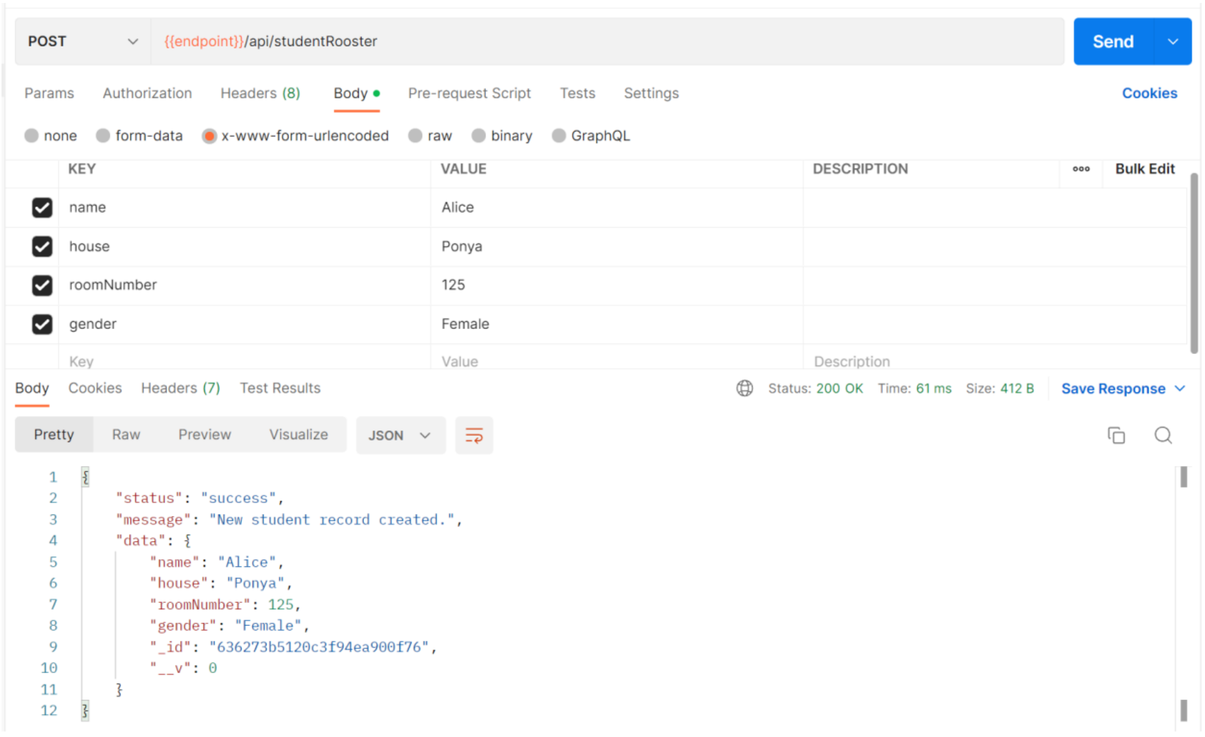


{{endpoint}}/api/studentRooster

Status: 200 OK Time: 41 ms Size: 411 B

```
{
    "status": "success",
    "message": "Students records retrieved.",
    "data": [
        {
            "_id": "63623c2c69220ec4e3e9e855",
            "gender": "Male",
            "name": "Bob",
            "house": "Tancho",
            "roomNumber": 152,
            "__v": 0
        }
    ]
}
```

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


POST
{{endpoint}}/api/studentRooster
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL

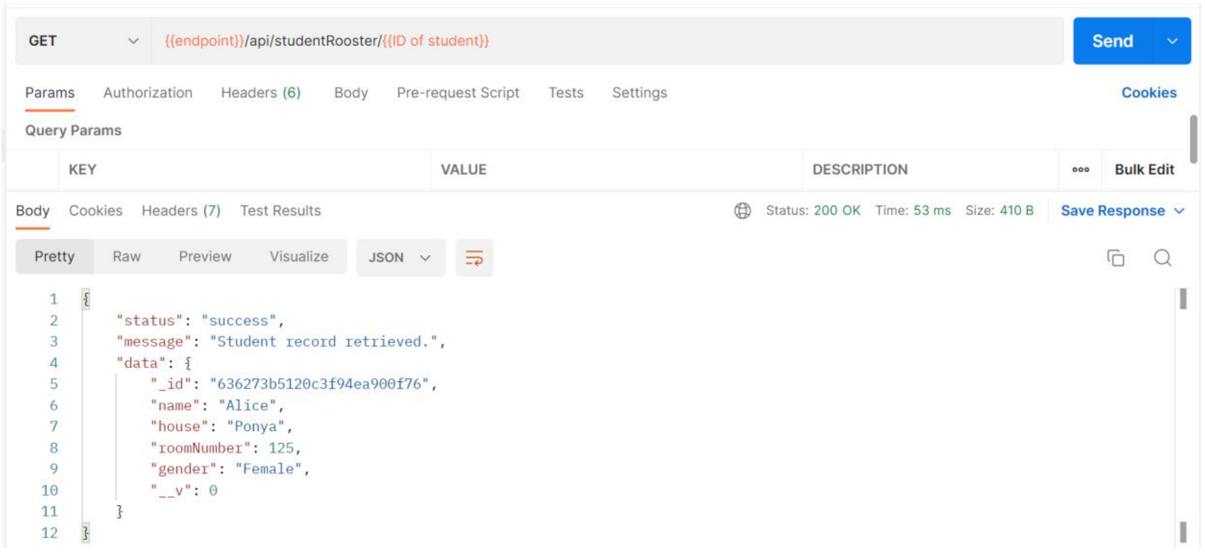
KEY	VALUE	DESCRIPTION
name	Alice	
house	Ponya	
roomNumber	125	
gender	Female	

Body Cookies Headers (7) Test Results
Status: 200 OK Time: 61 ms Size: 412 B
Save Response
Pretty Raw Preview Visualize JSON

```
{
    "status": "success",
    "message": "New student record created.",
    "data": {
        "name": "Alice",
        "house": "Ponya",
        "roomNumber": 125,
        "gender": "Female",
        "_id": "636273b5120c3f94ea900f76",
        "__v": 0
    }
}
```

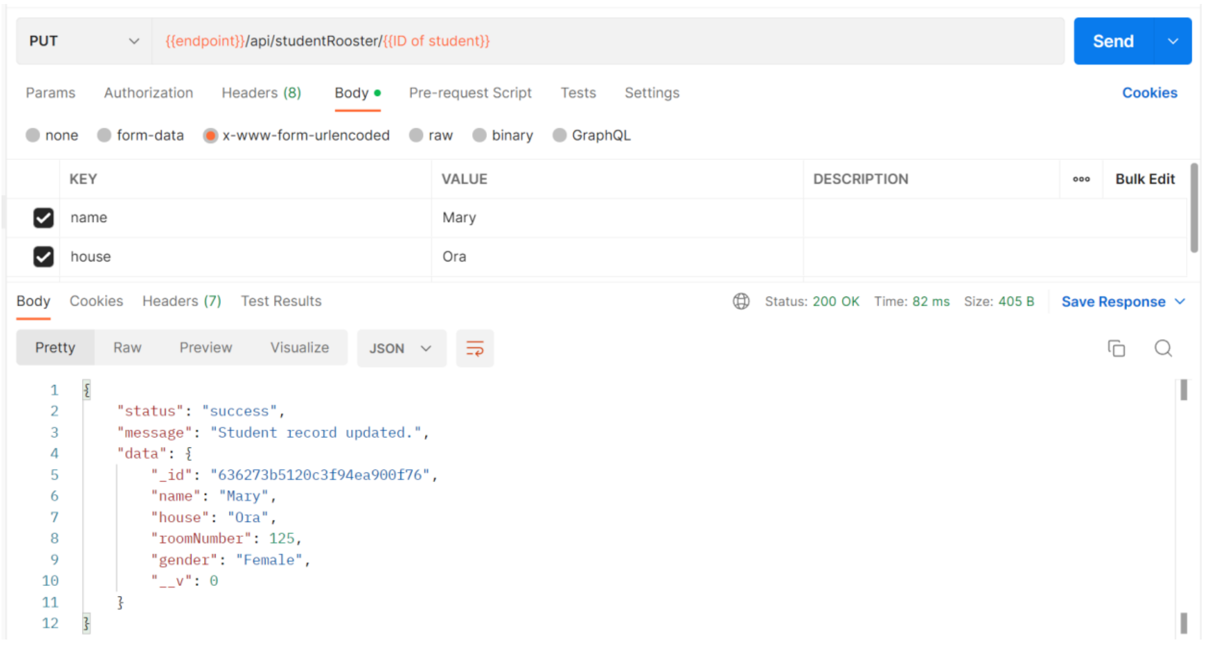
Note that from now on, we set the variable ‘ID of student’ to ‘636273b5120c3f94ea900f76’ to refer to the newly created record.

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


GET
{{endpoint}}/api/studentRooster/{{ID of student}}
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
Query Params
KEY VALUE DESCRIPTION
Bulk Edit
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 53 ms Size: 410 B
Save Response
Pretty Raw Preview Visualize JSON

```
{
    "status": "success",
    "message": "Student record retrieved.",
    "data": {
        "_id": "636273b5120c3f94ea900f76",
        "name": "Alice",
        "house": "Ponya",
        "roomNumber": 125,
        "gender": "Female",
        "__v": 0
    }
}
```

5. 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.



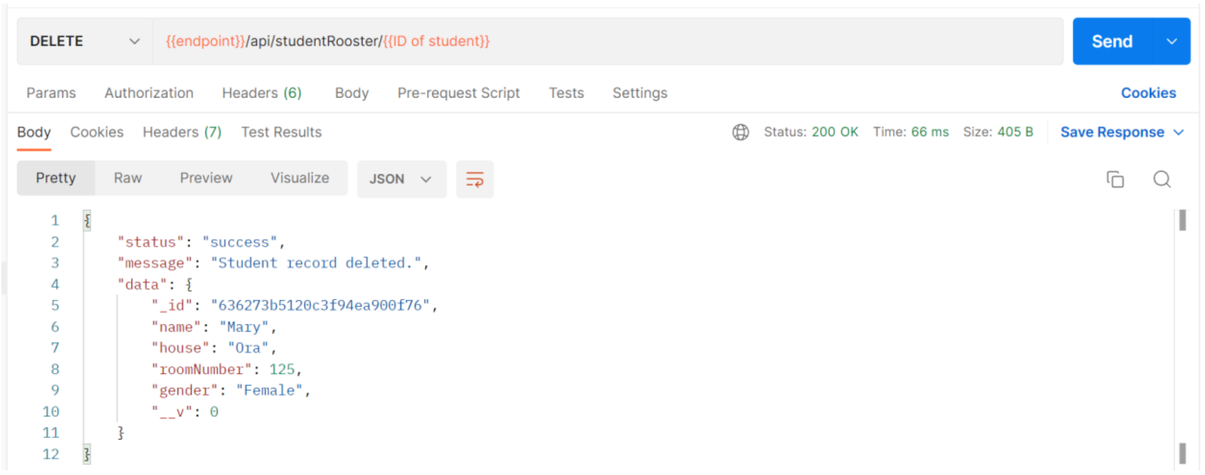
{{endpoint}}/api/studentRooster/{{ID of student}}

KEY	VALUE	DESCRIPTION
✓ name	Mary	
✓ house	Ora	

Status: 200 OK Time: 82 ms Size: 405 B

```
{
    "status": "success",
    "message": "Student record updated.",
    "data": {
        "_id": "636273b5120c3f94ea900f76",
        "name": "Mary",
        "house": "Ora",
        "roomNumber": 125,
        "gender": "Female",
        "__v": 0
    }
}
```

6. A DELETE request to '/api/studentRooster/*studentID*' will delete the entire record of a particular student.



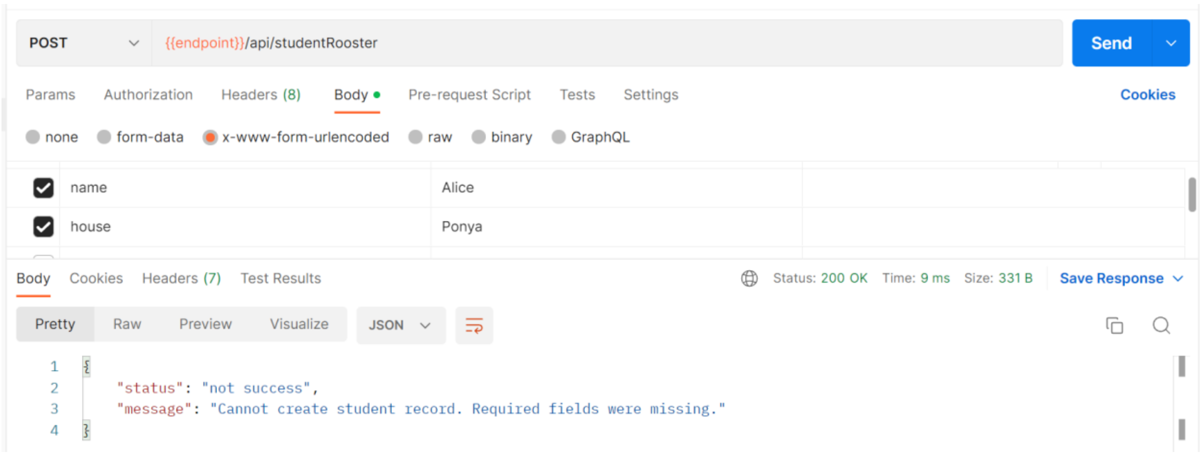
{{endpoint}}/api/studentRooster/{{ID of student}}

Status: 200 OK Time: 66 ms Size: 405 B

```
{
    "status": "success",
    "message": "Student record deleted.",
    "data": {
        "_id": "636273b5120c3f94ea900f76",
        "name": "Mary",
        "house": "Ora",
        "roomNumber": 125,
        "gender": "Female",
        "__v": 0
    }
}
```

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.


POST {{endpoint}}/api/studentRooster

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

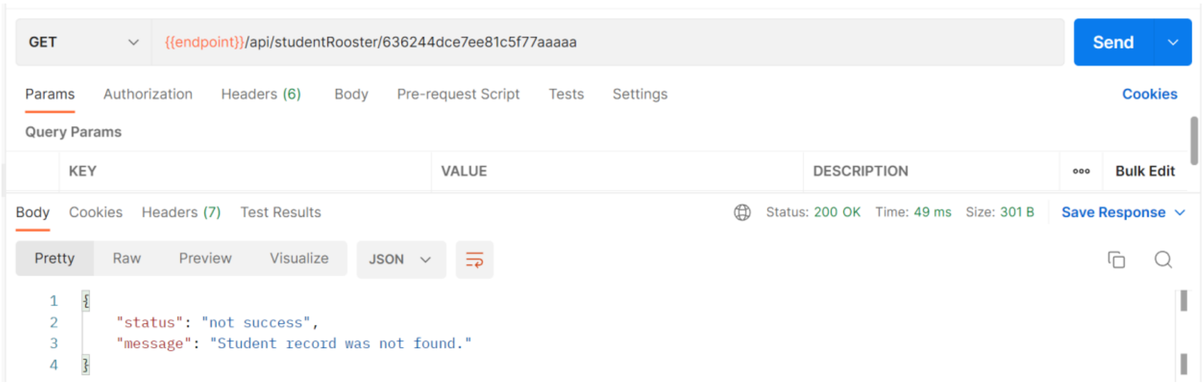
x-www-form-urlencoded

name	Alice
house	Ponya

Status: 200 OK Time: 9 ms Size: 331 B

```
{
    "status": "not success",
    "message": "Cannot create student record. Required fields were missing."
}
```

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


GET {{endpoint}}/api/studentRooster/636244dce7ee81c5f77aaaaa

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings | Cookies

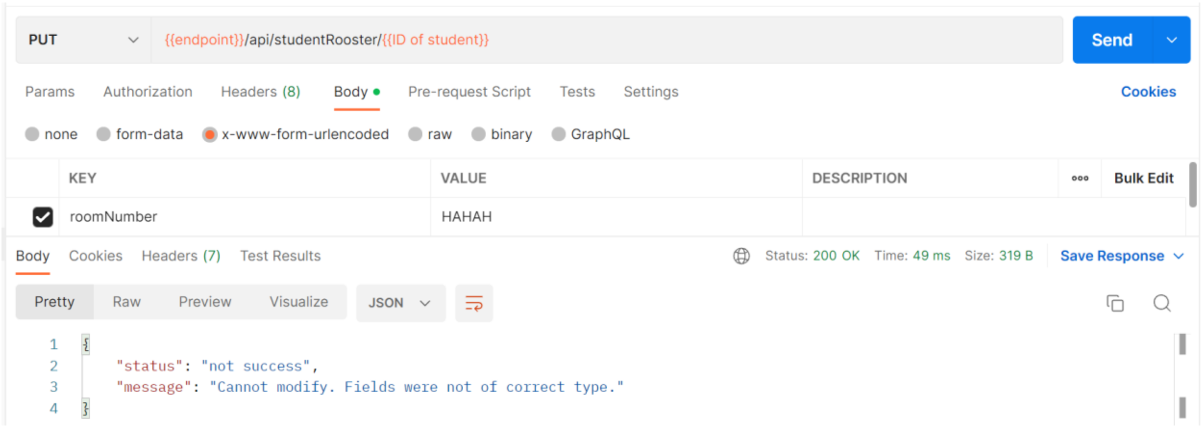
Query Params

KEY	VALUE	DESCRIPTION

Status: 200 OK Time: 49 ms Size: 301 B

```
{
    "status": "not success",
    "message": "Student record was not found."
}
```

3. 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).


PUT {{endpoint}}/api/studentRooster/{{ID of student}}

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

x-www-form-urlencoded

KEY	VALUE	DESCRIPTION
roomNumber	HAHAH	

Status: 200 OK Time: 49 ms Size: 319 B

```
{
    "status": "not success",
    "message": "Cannot modify. Fields were not of correct type."
}
```

Link to postman collection showing the testing:
https://www.getpostman.com/collections/e5d64a04169b1aba5ae4

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

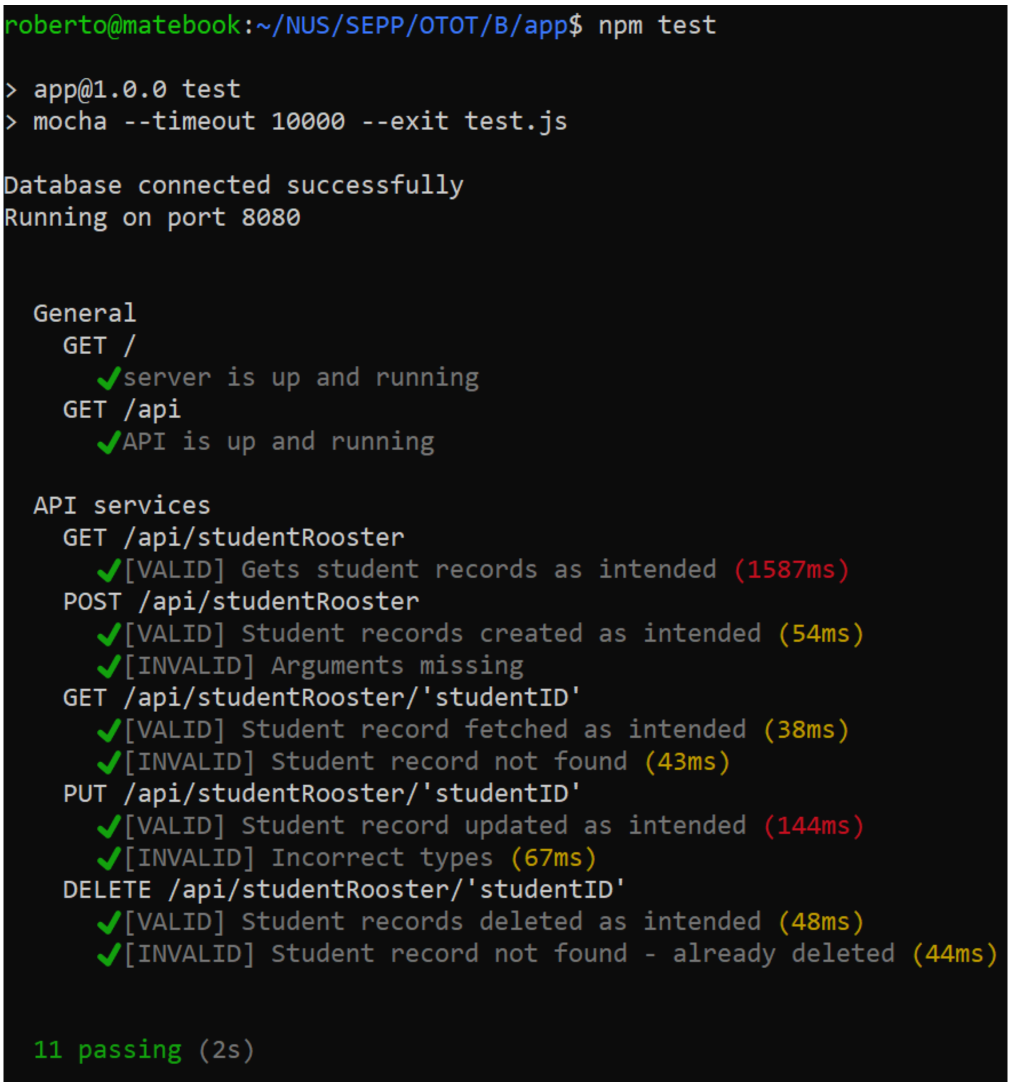
No commands need to be run as the deployed API is already on the web. Simply access ‘https://otottaskb-367404.as.r.appspot.com’ to access the server. Note that the calls to the API endpoint need to be made to ‘https://otottaskb-367404.as.r.appspot.com/api/studentRooster’ or ‘https://otottaskb-367404.as.r.appspot.com/api/studentRooster/*studentId*’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

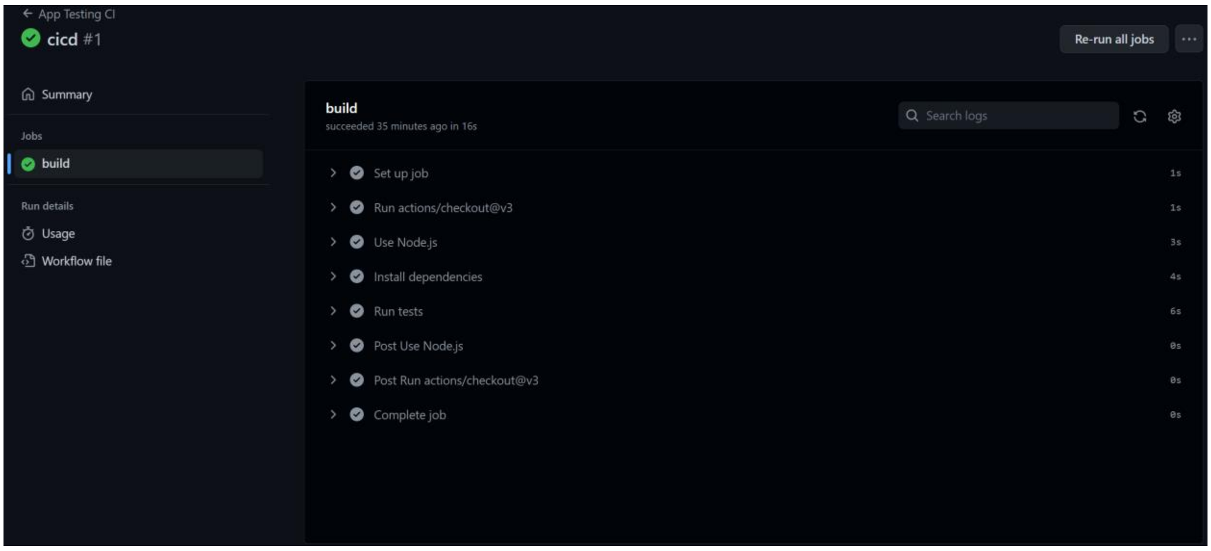
I made tests to test the functionality of all HTTP methods.

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



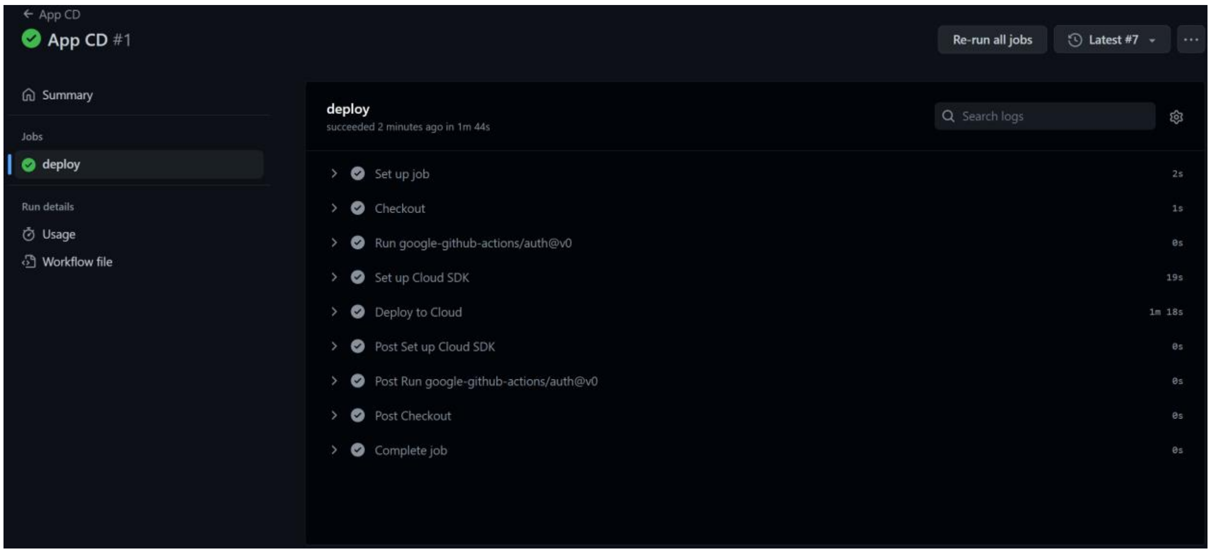
```
roberto@matebook:~/NUS/SEPP/OTOT/B/app$ npm test

> app@1.0.0 test
> mocha --timeout 10000 --exit test.js

Database connected successfully
Running on port 8080


  General
    GET /
      ✓server is up and running
    GET /api
      ✓API is up and running

  API services
    GET /api/studentRooster
      ✓[VALID] Gets student records as intended (1587ms)
    POST /api/studentRooster
      ✓[VALID] Student records created as intended (54ms)
      ✓[INVALID] Arguments missing
    GET /api/studentRooster/'studentID'
      ✓[VALID] Student record fetched as intended (38ms)
      ✓[INVALID] Student record not found (43ms)
    PUT /api/studentRooster/'studentID'
      ✓[VALID] Student record updated as intended (144ms)
      ✓[INVALID] Incorrect types (67ms)
    DELETE /api/studentRooster/'studentID'
      ✓[VALID] Student records deleted as intended (48ms)
      ✓[INVALID] Student record not found - already deleted (44ms)


  11 passing (2s)
```

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.



```
← App Testing CI
cicd #1                                          Re-run all jobs

Summary
Jobs
build
Run details
Usage
Workflow file

build
succeeded 35 minutes ago in 16s

> Set up job                          1s
> Run actions/checkout@v3             1s
> Use Node.js                         3s
> Install dependencies                4s
> Run tests                           6s
> Post Use Node.js                    0s
> Post Run actions/checkout@v3        0s
> Complete job                        0s
```

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.



```
← App CD
App CD #1                                 Re-run all jobs   Latest #7

Summary
Jobs
deploy
Run details
Usage
Workflow file

deploy
succeeded 2 minutes ago in 1m 44s

> Set up job                                    2s
> Checkout                                      1s
> Run google-github-actions/auth@v0             0s
> Set up Cloud SDK                              19s
> Deploy to Cloud                               1m 18s
> Post Set up Cloud SDK                         0s
> Post Run google-github-actions/auth@v0        0s
> Post Checkout                                 0s
> Complete job                                  0s
```

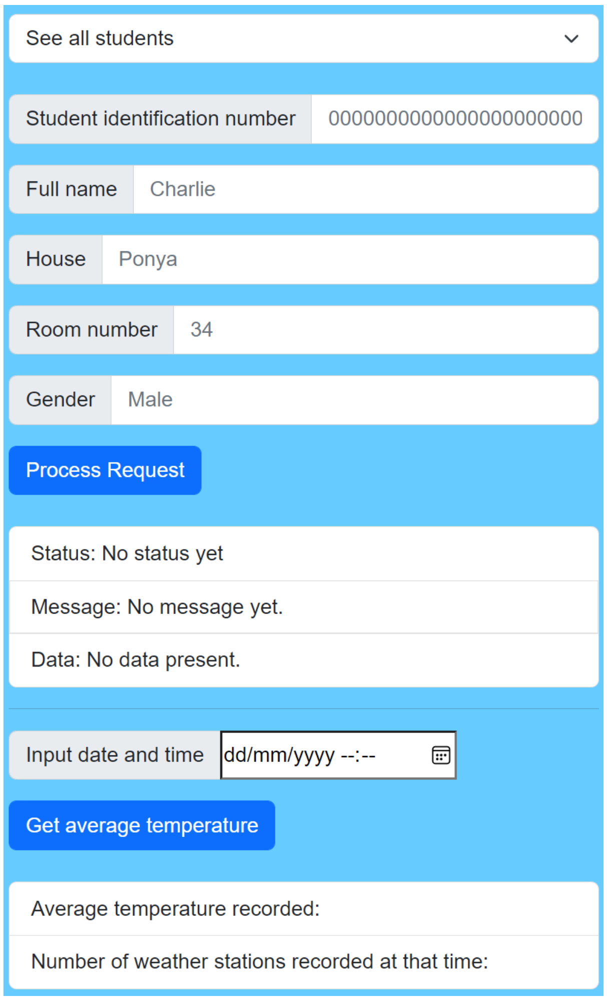
iv) Instructions on how to set up frontend (B3)

My frontend is located entirely in the 'frontend' directory. To set it up, simply move to that directory and run the following commands:

1. npm install`
2. `npm start`

3. The frontend server is now running locally on ‘http://localhost: 3000’

After running the command and accessing the webpage, should be able to see the following:



See all students

Student identification number	000000000000000000000
Full name	Charlie
House	Ponya
Room number	34
Gender	Male

Status: No status yet

Message: No message yet.

Data: No data present.

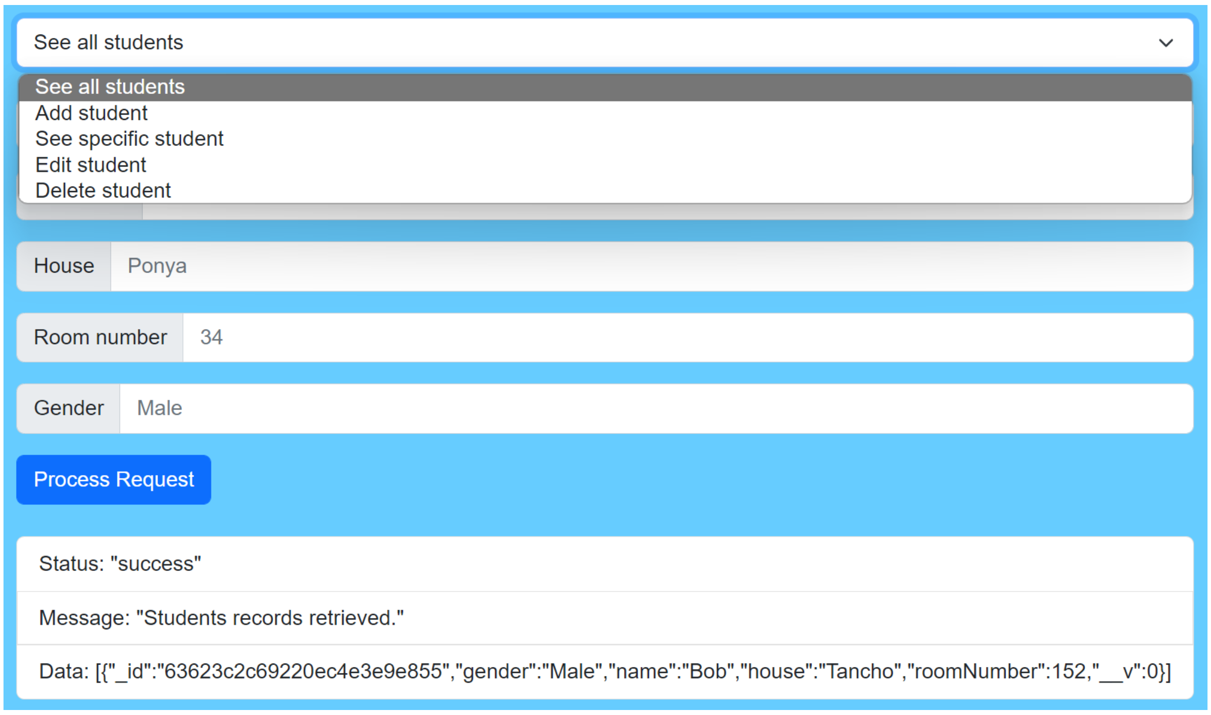
Input date and time	dd/mm/yyyy --:--

Average temperature recorded:

Number of weather stations recorded at that time:

The top half of the webpage is the Single Page Application (SPA) frontend I created to interact with my deployed API. The bottom half of the webpage is the frontend I created to interact with my serverless function (this part will be explained later in the report).

How to use the frontend for my deployed API:


See all students

House Ponya

Room number 34

Gender Male

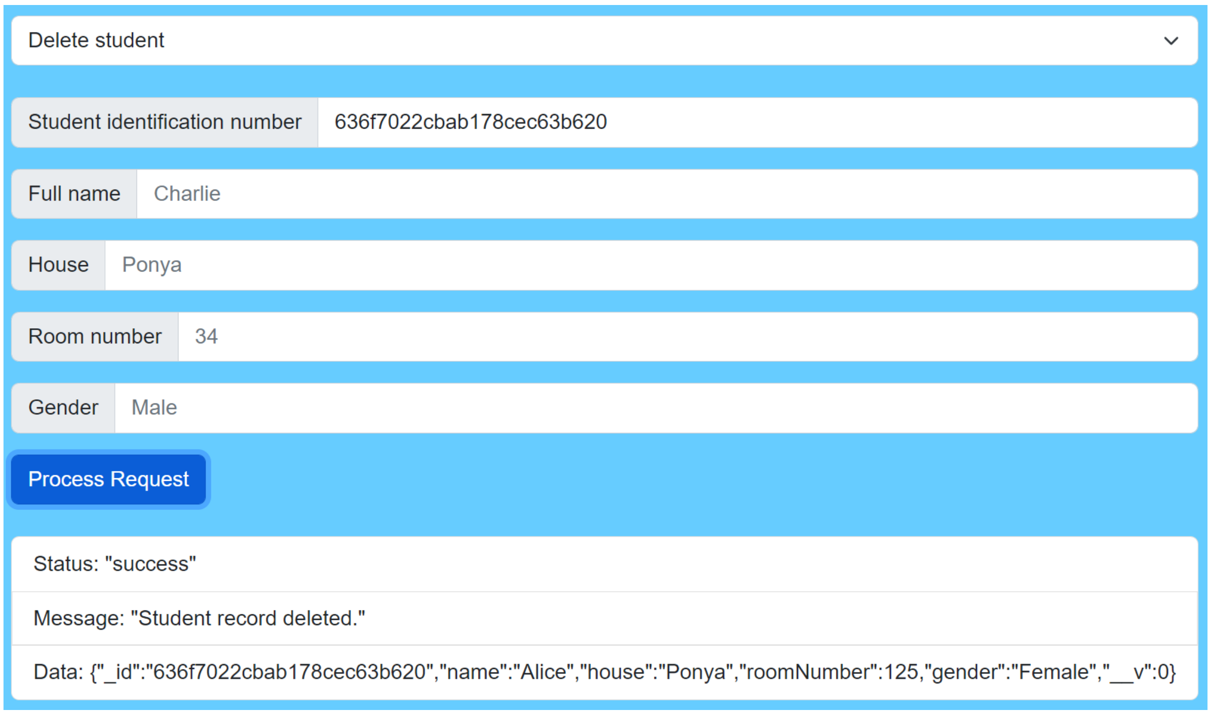
Status: "success"

Message: "Students records retrieved."

Data: [{"_id":"63623c2c69220ec4e3e9e855","gender":"Male","name":"Bob","house":"Tancho","roomNumber":152,"__v":0}]

The frontend can be used by choosing the command that the user wants to execute. In the screenshot above, I show what the options possible are. Specifically, I show that the student records are received correctly.

In the screenshot below, I show how a student is deleted. I select the "Delete student" request and add the student identification number of the student I want to delete to the form. Subsequently, once the request is processed, the status is shown.


Delete student

Student identification number 636f7022cbab178cec63b620

Full name Charlie

House Ponya

Room number 34

Gender Male

Status: "success"

Message: "Student record deleted."

Data: {"_id":"636f7022cbab178cec63b620","name":"Alice","house":"Ponya","roomNumber":125,"gender":"Female","__v":0}

Note that not all fields are required for all requests. For example, when a user wants to create a user, a student identification number does not need to be provided as will be assigned randomly.

Also, note how the frontpage never reloads: this is to make for a good user experience.

Finally, note how the webpage is styled. I used some custom styling coupled with boostrap.

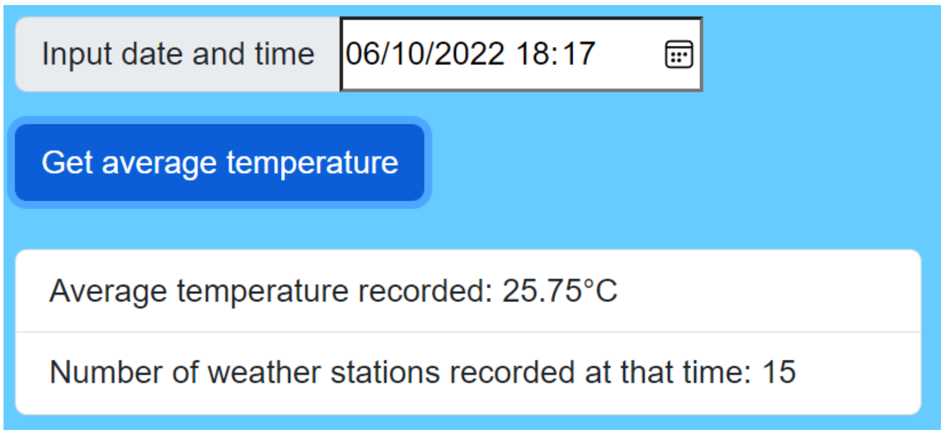
v) A brief explanation of what your serverless function does

My serverless function prompts a user for a date and time and subsequently displays what the average temperature in Singapore was at that specific date and time.

To do this, I query the Realtime Weather Readings across Singapore-Data.gov.sg website which is made by the Government of Singapore. This website provides the temperature of individual weather stations across the country. In my code, I clean up the data provided by the service to get the temperature for a specific date and time and subsequently I calculate the average temperature of the weather stations across Singapore.

This serverless function is hosted on Google Cloud on the same endpoint as my previous tasks (https://otottaskb-367404.as.r.appspot.com) and can be accessed with GET to '/api/tempAvg' and by providing the parameter 'date_time'. The response will look like the following JSON: {"dataPoints":16,"avg":26.849999999999994} . The 'dataPoints' indicates the number of weather stations that were active at the specified date and time that were then subsequently used to calculate the average temperature.

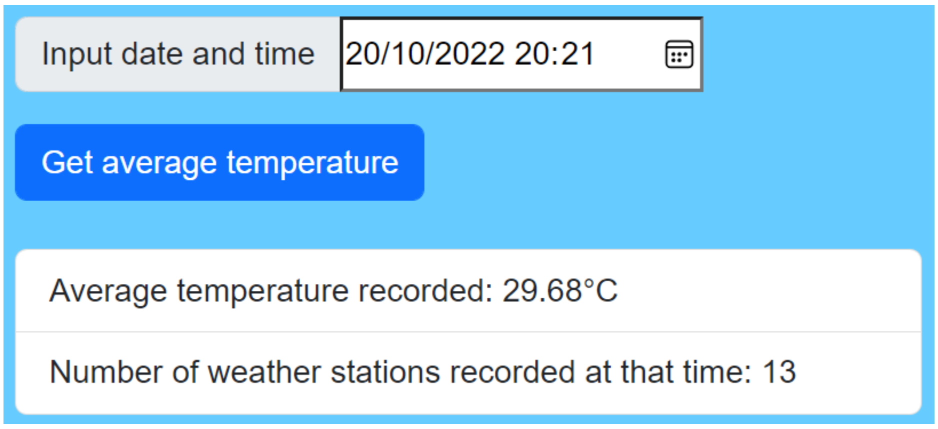
vi) Screenshots of the front-end interaction with your serverless function



06/10/2022 18:17

Average temperature recorded: 25.75°C

Number of weather stations recorded at that time: 15



Average temperature recorded: 29.68°C

Number of weather stations recorded at that time: 13