

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

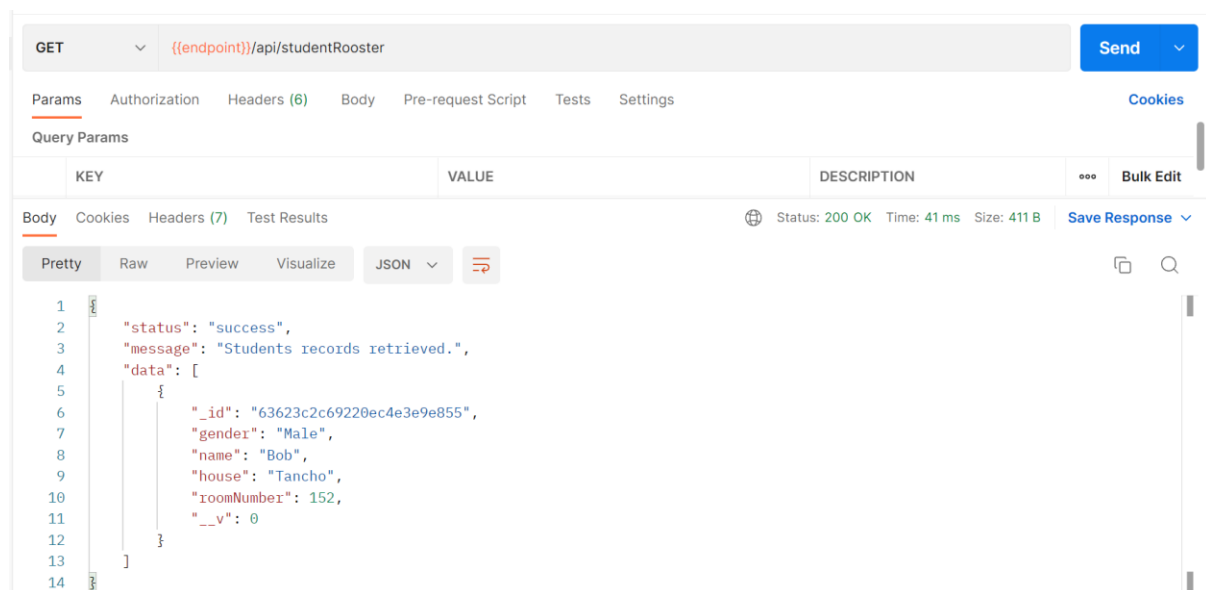
The API I created is an interface to store the information of students residing at a hall of residence (hostel). The API has been named 'studentRooster'.

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.



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	...	Bulk Edit
<input checked="" type="checkbox"/> name	Alice			
<input checked="" type="checkbox"/> house	Ponya			
<input checked="" type="checkbox"/> roomNumber	125			
<input checked="" type="checkbox"/> gender	Female			
Key	Value	Description		

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

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "success",
3   "message": "New student record created.",
4   "data": {
5     "name": "Alice",
6     "house": "Ponya",
7     "roomNumber": 125,
8     "gender": "Female",
9     "_id": "636273b5120c3f94ea900f76",
10    "__v": 0
11  }
12 }
```

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

```
1 {
2   "status": "success",
3   "message": "Student record retrieved.",
4   "data": {
5     "_id": "636273b5120c3f94ea900f76",
6     "name": "Alice",
7     "house": "Ponya",
8     "roomNumber": 125,
9     "gender": "Female",
10    "__v": 0
11  }
12 }
```

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.

PUT `{{endpoint}}/api/studentRooster/{{ID of student}}` 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	...	Bulk Edit
<input checked="" type="checkbox"/>	name	Mary			
<input checked="" type="checkbox"/>	house	Ora			

Body Cookies Headers (7) Test Results Status: 200 OK Time: 82 ms Size: 405 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "status": "success",
3    "message": "Student record updated.",
4    "data": {
5      "_id": "636273b5120c3f94ea900f76",
6      "name": "Mary",
7      "house": "Ora",
8      "roomNumber": 125,
9      "gender": "Female",
10     "__v": 0
11   }
12 }
```

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

DELETE `{{endpoint}}/api/studentRooster/{{ID of student}}` Send

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

Body Cookies Headers (7) Test Results Status: 200 OK Time: 66 ms Size: 405 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "status": "success",
3    "message": "Student record deleted.",
4    "data": {
5      "_id": "636273b5120c3f94ea900f76",
6      "name": "Mary",
7      "house": "Ora",
8      "roomNumber": 125,
9      "gender": "Female",
10     "__v": 0
11   }
12 }
```

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` Send

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

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	name	Alice
<input checked="" type="checkbox"/>	house	Ponya

Body Cookies Headers (7) Test Results Status: 200 OK Time: 9 ms Size: 331 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "status": "not success",
3    "message": "Cannot create student record. Required fields were missing."
4  }

```

- 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` 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: 49 ms Size: 301 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "status": "not success",
3    "message": "Student record was not found."
4  }

```

- 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}}` 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	...	Bulk Edit
<input checked="" type="checkbox"/>	roomNumber	HAHAH		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 49 ms Size: 319 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "status": "not success",
3    "message": "Cannot modify. Fields were not of correct type."
4  }

```

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

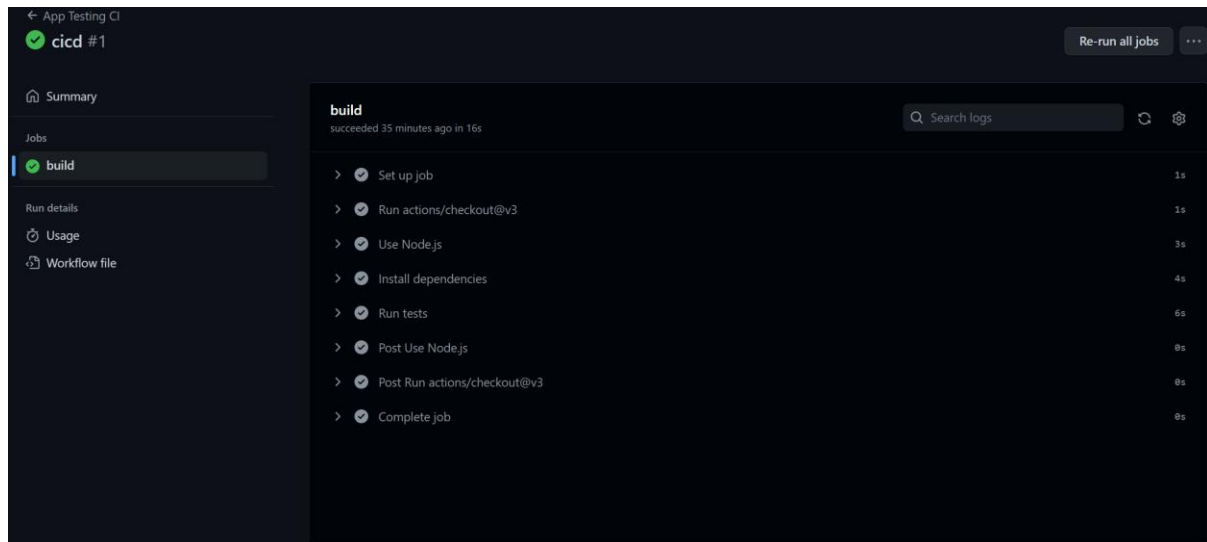
Database connected successfully
Running on port 8080

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

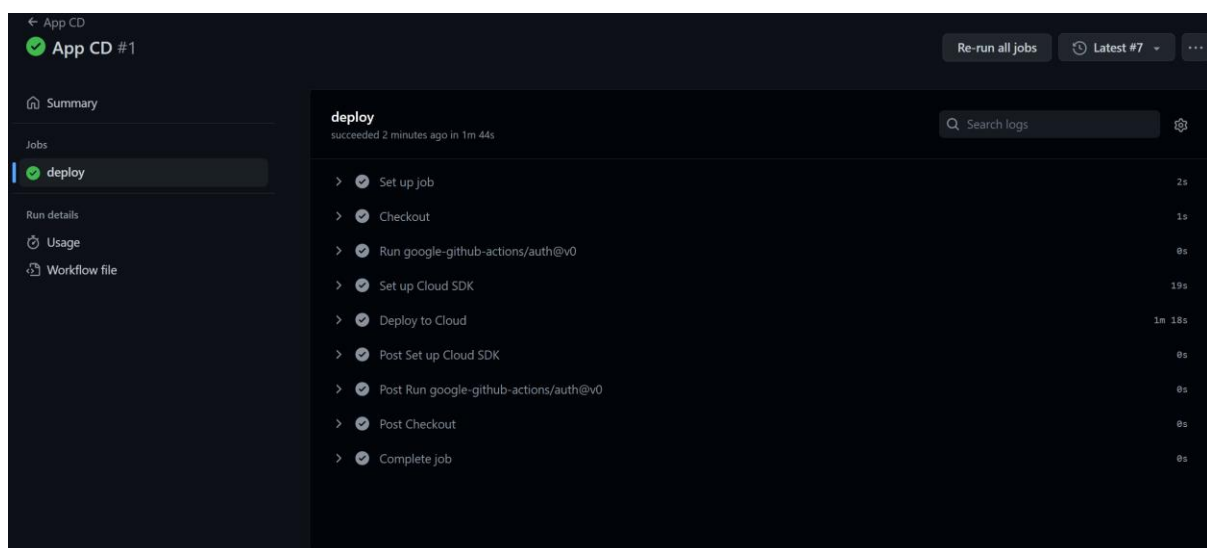
API services
  GET /api/studentRooster
    ✓[VALID] Gets student records as intended (1587ms)
  POST /api/studentRooster
    ✓[VALID] Student records created as intended (54ms)
    ✓[INVALID] Arguments missing
  GET /api/studentRooster/'studentID'
    ✓[VALID] Student record fetched as intended (38ms)
    ✓[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.



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.



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 00000000000000000000

Full name Charlie

House Ponya

Room number 34

Gender Male

Process Request

Status: No status yet

Message: No message yet.

Data: No data present.

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

Get average temperature

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

See all students
Add student
See specific student
Edit student
Delete student

House Ponya

Room number 34

Gender Male

Process Request

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

Process Request

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 bootstrap.

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](https://data.gov.sg/dataset/realtime-weather-readings-across-singapore) 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

The screenshot shows a web application interface with a light blue background. At the top, there is a text input field labeled "Input date and time" containing the text "06/10/2022 18:17" and a calendar icon. Below this is a blue button with white text that says "Get average temperature". Underneath the button is a white box with a light blue border containing two lines of text: "Average temperature recorded: 25.75°C" and "Number of weather stations recorded at that time: 15".

Input date and time

20/10/2022 20:21



Get average temperature

Average temperature recorded: 29.68°C

Number of weather stations recorded at that time: 13