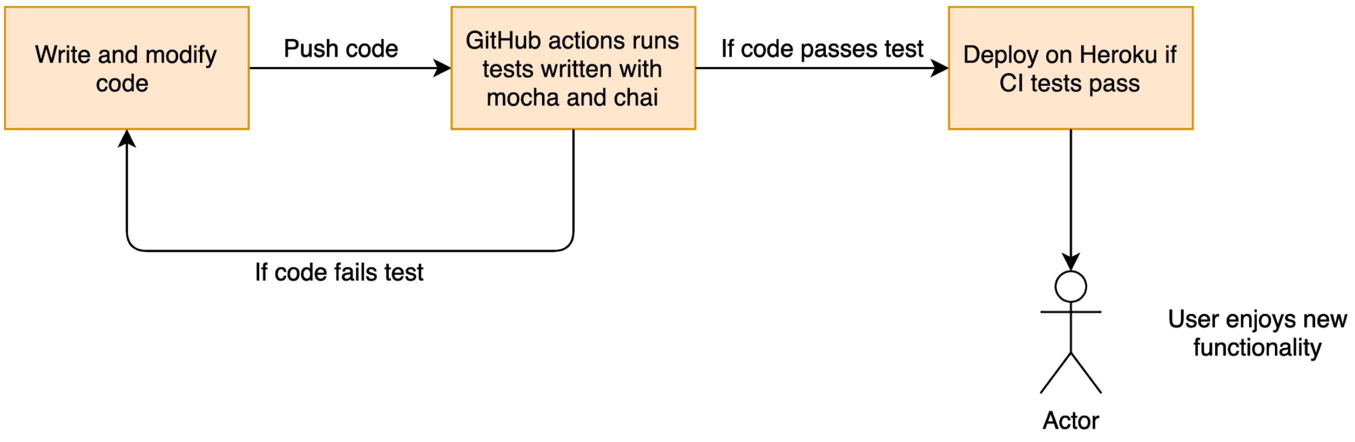
Testing Report

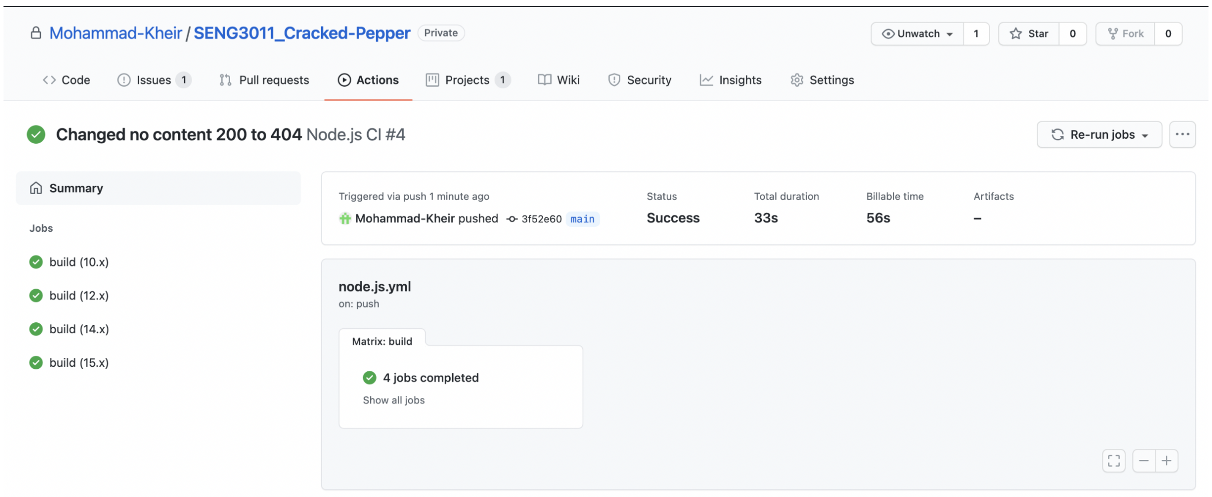
<u>Testing Process and Philosophy overview:</u>

- We have integrated Continuous integration and Continues deployment practices by utilizing GitHub actions and Heroku configurations. Hence everytime we push to the repo automatic tests are run, this ensures the quality of code and minimises bugs present.

- The following is a flow diagram with explanation of our testing process:


Write and modify code
Push code
GitHub actions runs tests written with mocha and chai
If code passes test
Deploy on Heroku if CI tests pass
If code fails test
User enjoys new functionality
Actor

1. Push code onto main. Example “Changed Status code of an error from 200 to 404”.
2. Github Actions automatically runs tests. Tests are in tests.js made with Mocha and Chai modules.



Changed no content 200 to 404 Node.js CI #4

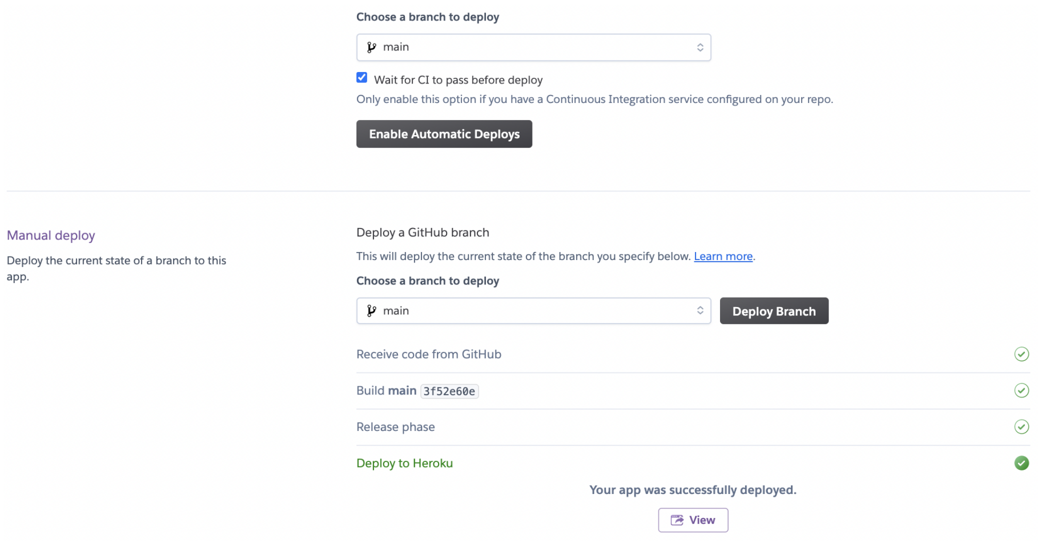
- build (10.x)
- build (12.x)
- build (14.x)
- build (15.x)

Triggered via push 1 minute ago	Status	Total duration	Billable time	Artifacts
Mohammad-Kheir pushed 3f52e60 main	Success	33s	56s	–

node.js.yml
on: push

Matrix: build
4 jobs completed
Show all jobs

3. If tests pass then Heroku will use its premade build packs to deploy the project.



Choose a branch to deploy

main

[✓] Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub	✓
Build **main** `3f52e60e`	✓
Release phase	✓
Deploy to Heroku	✓

Your app was successfully deployed.

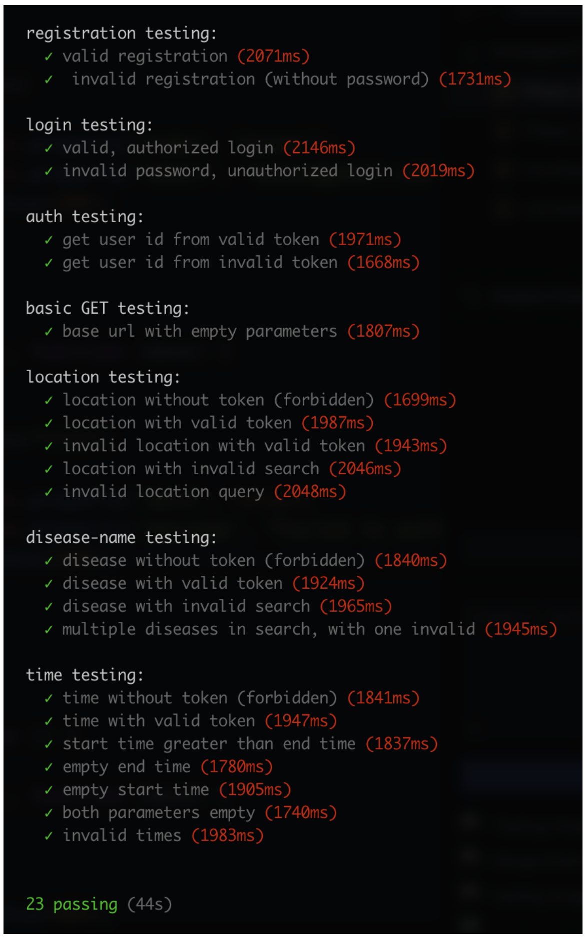
View

4. We can then access new deployments through github or the server.

In Depth testing:

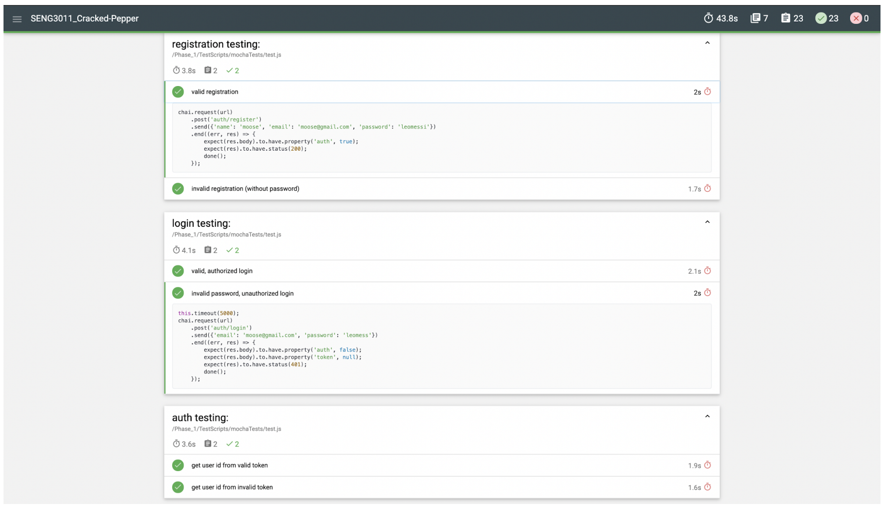
- We first tested all the endpoints with different queries and authentication accesses using PyTest, which was TDD (Test Driven Development). We also then did the same with **Mocha** and **Chai** in JS using BDD (Behaviour Driven Development), which is more suitable for API testing.

  What we aim to test for in every endpoint and route:
  1) schema validation
  2) correct authentication
  3) status code assertions
  4) correct JSON response assertions



```
registration testing:
  ✓ valid registration (2071ms)
  ✓  invalid registration (without password) (1731ms)

login testing:
  ✓ valid, authorized login (2146ms)
  ✓ invalid password, unauthorized login (2019ms)

auth testing:
  ✓ get user id from valid token (1971ms)
  ✓ get user id from invalid token (1668ms)

basic GET testing:
  ✓ base url with empty parameters (1807ms)

location testing:
  ✓ location without token (forbidden) (1699ms)
  ✓ location with valid token (1987ms)
  ✓ invalid location with valid token (1943ms)
  ✓ location with invalid search (2046ms)
  ✓ invalid location query (2048ms)

disease-name testing:
  ✓ disease without token (forbidden) (1840ms)
  ✓ disease with valid token (1924ms)
  ✓ disease with invalid search (1965ms)
  ✓ multiple diseases in search, with one invalid (1945ms)

time testing:
  ✓ time without token (forbidden) (1841ms)
  ✓ time with valid token (1947ms)
  ✓ start time greater than end time (1837ms)
  ✓ empty end time (1780ms)
  ✓ empty start time (1905ms)
  ✓ both parameters empty (1740ms)
  ✓ invalid times (1983ms)


23 passing (44s)
```

- The testing begins with a registration and login tests, followed by authentication tests to verify if the tokens passed are valid. We then look to test different queries (location, disease-name, and time) on sample data present in the DB. As per the picture above, we tested the endpoint with many different input combinations. The sub-tests in every section of testing, are descriptions of the input passed for the respective tests.



registration testing:
/Phase_1/TestScripts/mochaTests/test.js
3.8s 2 ✓ 2

valid registration 2s

```
chai.request(url)
    .post('auth/register')
    .send({'name': 'moose', 'email': 'moose@gmail.com', 'password': 'leomessi'})
    .end((err, res) => {
        expect(res.body).to.have.property('auth', true);
        expect(res).to.have.status(200);
        done();
    });
```

invalid registration (without password) 1.7s

login testing:
/Phase_1/TestScripts/mochaTests/test.js
4.1s 2 ✓ 2

valid, authorized login 2.1s

invalid password, unauthorized login 2s

```
this.timeout(5000);
chai.request(url)
    .post('auth/login')
    .send({'email': 'moose@gmail.com', 'password': 'leomess'})
    .end((err, res) => {
        expect(res.body).to.have.property('auth', false);
        expect(res.body).to.have.property('token', null);
        expect(res).to.have.status(401);
        done();
    });
```

auth testing:
/Phase_1/TestScripts/mochaTests/test.js
3.6s 2 ✓ 2

get user id from valid token 1.9s

get user id from invalid token 1.6s

- To have a valid token for all API calls after the login test, we simply used the token generated from the login call in the headers for our core API requests.

- The testing for the moment produces a clean result, with no tests failing. By trying many different input combinations, we uncovered some issues with registration while testing, (and some small minor issues as listed on Github Issues) that we will aim to tackle in the coming weeks. Rigorous schema testing will also be a priority to make sure all returned json responses are valid.

File locations:

- Instructions on how to run tests and view online html reports are in readme.md.

- The **sample data** which we tested our endpoint on, is available at:
  SENG3011_Cracked-Pepper/Phase_1/TestScripts/mochaTests

- The **test report**, which is generated as a HTML page, is available on:
  SENG3011_Cracked-Pepper/mochawesome-report/mochawesome.html

- **Test files** and **configuration** files for testing are located at:
  SENG3011_Cracked-Pepper/Phase_1/TestScripts/mochaTests

- Limitations with regards to testing include more sample data needed for a more diverse range of test data, as well as initial difficulties encountered with the time each API call took in the testing (>2000ms for a call).

- The tests aim to find any glaring issues with the endpoints created for the API. Along with Mocha and Chai, we set up CI (continuous integration) using Github Actions that runs tests on pushed code. If it passes the tests, the API is ready to be pushed for production.