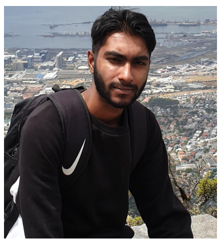
Testing Policy

August 1st, 2023

Group Members

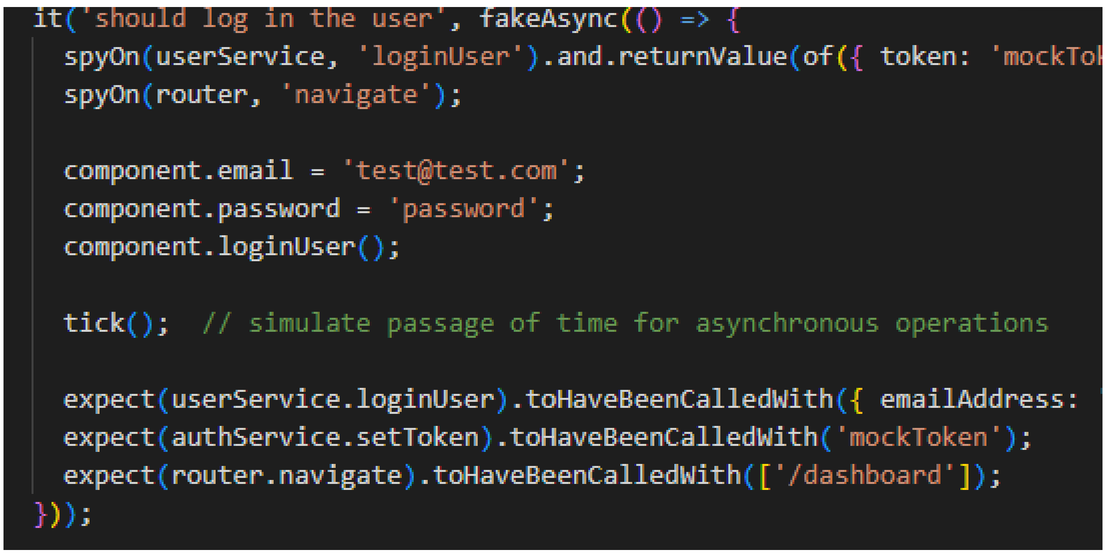
				
Priyul Mahabeer	Ashir Butt	Jaimen Govender	Dharshan Pillay	Edwin Sen-Hong Chang
u20421169	u20422475	u20464348	u19027487	u20424575

Testing Tools:

- Mocha Chai - we used mocha chai for our testing since it was easy to set up and write the tests, the syntax was clear which makes our tests easier to read and understand, and mocha handles asynchronous code well which is crucial for our application.
- Karma - we used Karma for our testing since it allowed for automated testing, it was easy to use and set up, and it provided code coverage reporting.
- Github Workflows - we used this to automate our tests.The GitHub repository is configured with workflows that are activated by a pull request made to the develop branch. These workflows encompass tasks which execute both unit and integration tests encompassed within the system.
- Postman - we used postman to test our API when it was being developed. We used it to make sure our API queries were returning the correct data.

Testing Procedure

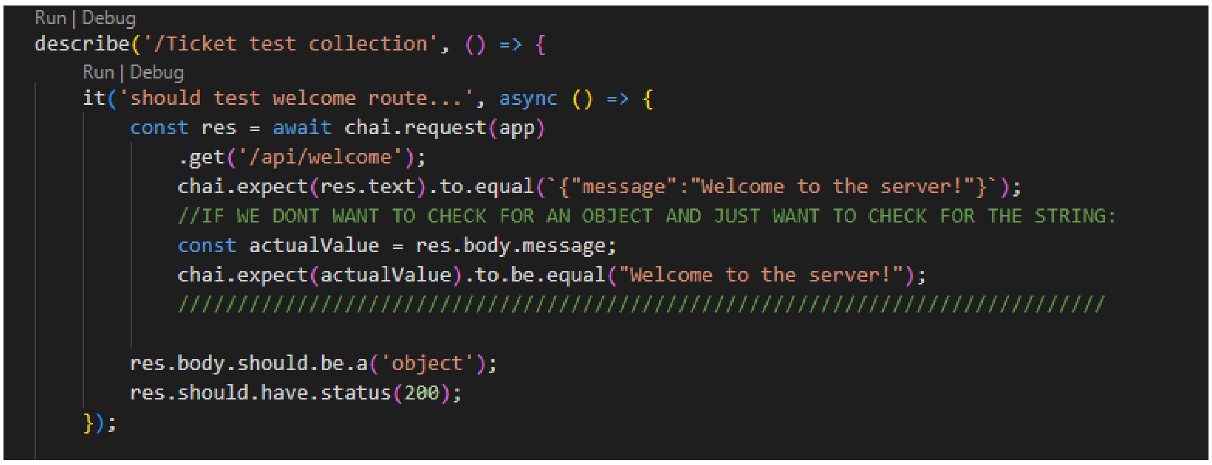
Karma



```
it('should log in the user', fakeAsync(() => {
  spyOn(userService, 'loginUser').and.returnValue(of({ token: 'mockTok
  spyOn(router, 'navigate');

  component.email = 'test@test.com';
  component.password = 'password';
  component.loginUser();

  tick();  // simulate passage of time for asynchronous operations

  expect(userService.loginUser).toHaveBeenCalledWith({ emailAddress:
  expect(authService.setToken).toHaveBeenCalledWith('mockToken');
  expect(router.navigate).toHaveBeenCalledWith(['/dashboard']);
}));
```

1) Set up Karma,create a Karma.conf.js in the project’s root directory.
2) Install required dependencies, ensure you have Node.js and npm installed.
3) Configure Karma, open the karma.conf.js file and configure the settings accordingly.
4) Write tests, create your test files in directory of the component the test is for. Name it componentName.component.spec.ts E.g login-page.component.spec.ts
5) View test results
6) Refactor your code and run the tests again, depending on your results from the tests.

Mocha Chai



```
Run | Debug
describe('/Ticket test collection', () => {
    Run | Debug
    it('should test welcome route...', async () => {
        const res = await chai.request(app)
            .get('/api/welcome');
            chai.expect(res.text).to.equal(`{"message":"Welcome to the server!"}`);
            //IF WE DONT WANT TO CHECK FOR AN OBJECT AND JUST WANT TO CHECK FOR THE STRING:
            const actualValue = res.body.message;
            chai.expect(actualValue).to.be.equal("Welcome to the server!");
            ///////////////////////////////////////////////////////////////////////////////

        res.body.should.be.a('object');
        res.should.have.status(200);
    });
```

1) Install Mocha Chai as dependencies in the project.
2) Create a folder called ‘Test’ to store all of the tests.
3) In the test folder in the backend directory, we have tests for each of our nodes. The files are named *.spec.ts. E.g ticket.spec.ts
4) When writing a test for code in the backend the test itself should have the following:
    a) ‘describe()’ - this is used to group related test cases together, and provide a description of what the test is doing. This provides us with a better understanding of what is happening.
    b) ‘it()’ - these functions are used within the describe block to define individual test cases.
5) Assertions - our assertions are written inside the ‘it’ blocks. We used assertions to check if the actual results matched the expected results. Use assertions like expect, should, assert etc.
6) Review results. Then iterate and refactor as needed.

Git Repo Test Cases

https://github.com/COS301-SE-2023/ABC-Service-Request-System/tree/main/backend/test

Test Reports

Code Coverage:
https://github.com/COS301-SE-2023/ABC-Service-Request-System (and then click on the codecov badge)
or
https://app.codecov.io/gh/COS301-SE-2023/ABC-Service-Request-System

Continuous Integration:
https://github.com/COS301-SE-2023/ABC-Service-Request-System/actions/workflows/develop-deploy.yaml

Continuous Development:
https://github.com/COS301-SE-2023/ABC-Service-Request-System/actions/workflows/ci-development.yaml

Unit and Integration:
https://github.com/COS301-SE-2023/ABC-Service-Request-System/actions/workflows/backend_testing.yaml

Non-functional Testing

Usability Tests:

We performed a series of usability tests for our application. We asked people who are frequently involved with projects or who run businesses. The users were given the app to use and explore. Afterwards they were asked to perform a simple task. E.g View a profile, View a ticket, Create a ticket etc.
https://drive.google.com/drive/u/1/folders/1y-hHHE0p5-6EFA3cqzFfl4hLMMsQvJ1S

Performance Tests:

We performed a series of performance tests by connecting to different servers all over the world and tested their speed. We used Pingdom for this.
https://docs.google.com/document/d/1POPu9SNRkZdZTpKGAhJU6XrUZxLmfJ5C215JGANaQHQ/edit

Security Tests:

We performed a series of security tests by using OWASP ZAP Spidering and Active Scanning. These tests helped us find security vulnerabilities in our system.
https://github.com/COS301-SE-2023/ABC-Service-Request-System/wiki/Non-Functional-Testing#security-testing-using-owasp-zap