## TESTING REPORT Unit Testing

The most basic kind of testing that ensures a piece of code operates correctly on its own is unit testing. The smallest testable component of an application is a unit test. The primary goal is to test every feature or part as soon as it is built. A unit test typically generates a single result from one or more inputs.

Jest's primary emphasis is on large-scale web application support and function simplicity. It is compatible with Angular, Babel, TypeScript, Node.js, React, Vue.js, Svelte, and other online apps. Since Jest is a well-known framework in the JavaScript community, that is why we picked it. Their documentation is very straight forward and easy to understand while also having a big community to help in case the Documentation is not enough. Developers would therefore have a lower learning curve than with existing unit testing frameworks.

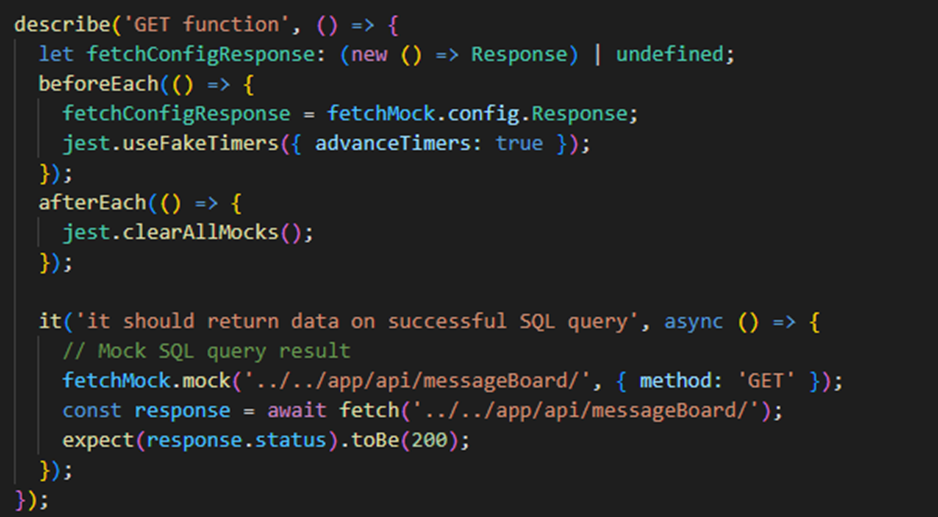
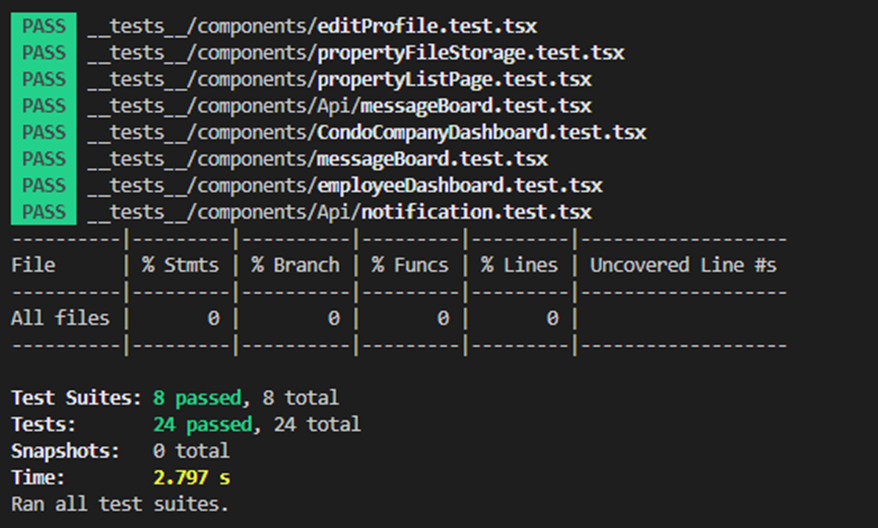


Figure 1: Example of our Unit Testing using Jest to test one of our API call

## Code Coverage

Although code coverage tools do not promise to uncover every hidden problem in the present codebase, they do enable engineers to assess the extent to which the test cases cover the code base. Therefore, knowing code coverage data will contribute to developers' confidence-building. To activate the code coverage tool included with Jest, all you need to do is add the "--coverage" parameter to the test run command. Jest will automatically go through the project and gather all the written test files; no further setup is necessary. Unfortunately, we ran into a problem last minute before the deadline of the third sprint. Even with the unit test files, when Jest is asked to do a report test coverage, it gives 0 on the percentage for the different types of test coverages.



## System Testing

In the software development process, system testing, often known as end-to-end testing, is a popular methodology used to evaluate whether an application functions successfully in conditions like those of a product and using data that simulates real-world situations. Making a realistic experience that adheres to the procedures of an actual user scenario is the aim. Testing on the system under test as well as any linked subsystems should be done to completely validate the system.

We selected Cypress because it is one of the most well-known frameworks in the JavaScript community for system (end-to-end) testing. Cypress is very intuitive to learn and code to accomplish the same tasks than other frameworks that are comparable, such Nightwatch. Furthermore, Cypress has additional features that enable developers to visually see the testing stages, which is quite helpful in identifying the precise point at which the test failed, if that is indeed the case. Also, their feature where you can select an element on the website through the browser of cypress and it will give you the line of code to select that element in the code if ever I need it for any actions.

