## TESTING REPORT

## Unit Testing

An unanticipated issue hindered our progress in the third sprint: the unit testing files were corrupted, necessitating a thorough reimplementation. This difficulty led us to implement Node.js, which aligns with the core goal of unit testing: independently confirming that distinct code segments function as intended. Unit testing is focused on testing individual functions or components once they are created, focusing on the smallest units of an application that can be examined. A unit test typically processes inputs to provide a single, meaningful result.

We kept using Jest as our testing library to keep our testing methodology consistent and to make use of well-known technologies. In order to ensure a full and comprehensive testing process, we improved our techniques to test the API rigorously and used several methodologies to evaluate the rendered elements of the sites. Jest's strength is its wide range of compatibility with various web development tools and its unique focus on making testing complex online applications easier. This framework is our first pick because it works well with Angular, Babel, TypeScript, Node.js, React, Vue.js, Svelte, etc.

Due to several obstacles, such as a great deal of faults that occurred during the deployment and testing stages, we have opted to postpone the integration of Continuous Integration (CI) procedures until Sprint 5. The reason for this calculated delay is the significant time constraints we encountered, which required us to reallocate our resources in order to prioritize and take care of the urgent testing issues. This timetable modification demonstrates our dedication to precision and high standards in our deployment procedure.

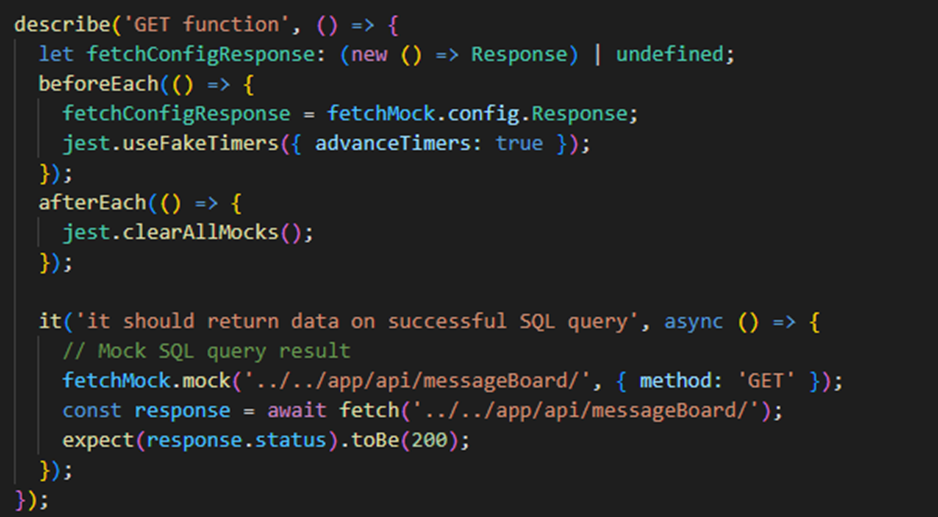
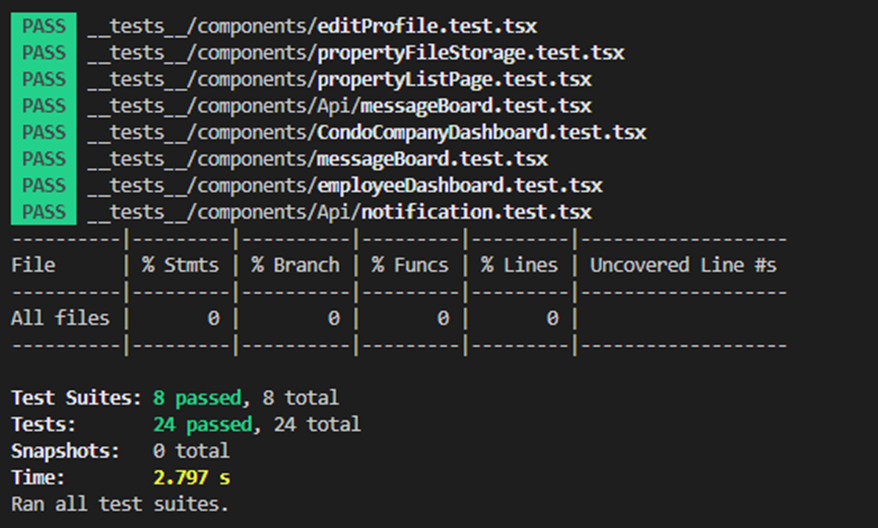
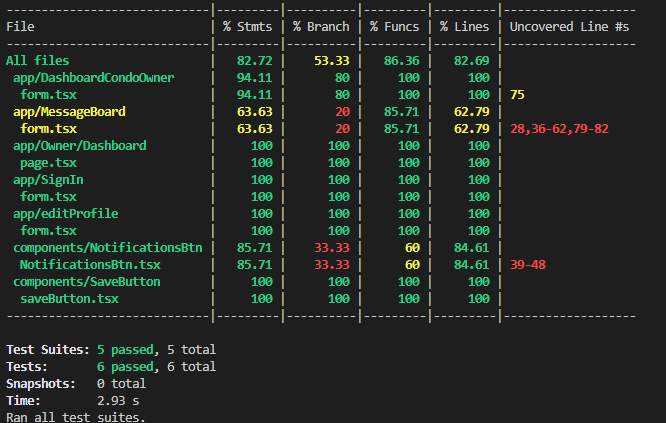


Figure 1: Example of our Unit Testing using Jest to test one of our API call

## Code Coverage



***Sprint 3****: test coverage report*



***Sprint 4****: test coverage report*

A strong and comprehensive testing strategy is shown by the report's overall high percentage of coverage, with many files reaching 100% across a range of parameters. Interestingly, though, some parts of several files—like components/NotificationBtn and app/messageBoard—show less coverage in some measures. This implies that there may be room in those files to add more test cases for certain scenarios, function calls, or lines. There is space for improvement in the components/SaveBtn file, particularly in branch coverage, which suggests that not all conditional branches have undergone thorough testing.

The summary line indicates that all six test suites—that is, all six test runs—have succeeded, indicating a flawless test run devoid of any errors. All of the tests took only 2.93 seconds to complete, which shows that developers may get feedback rather quickly.

In conclusion, the report demonstrates a strict testing methodology, but it also identifies certain areas where the test suite might be improved to obtain even more thorough coverage. Regressions can be prevented by new changes and good code quality is maintained with the help of such comprehensive coverage measurements.

## System Testing

For the system testing, we still implemented Cypress. The goal is to provide a realistic experience that follows the steps of a real-world user scenario. To fully validate the system, testing should be done on both the system that is being tested and any related subsystems. System testing delayed for sprint 5 due to the pages not being linked together and they only way to access each page to through the URL.

