Static testing tools

Static testing tools are quite important in order to maintain good coding standards. We mainly rely on ESLint and Typescript to make sure our code is both uniform and fulfills best practices. ESLint is basically a set of rules for your code. As developers we can add new rules or disable default rules we don't want. ESLint provides static code analysis which tells us what is wrong with our code by underlining parts of the code that don't adhere to the defined rules and stating which rules have been broken. ESLint comes packaged with a script that checks all of your files and tells you which files/lines are breaking the rules and how. We run this script as part of our CI to catch any errors related to our code structure.

ESLint also allows us to set up a format script, which we can call to automatically make all of our code refactor and reformat to fit the rules. On top of ESLint we use prettier. Prettier only accounts for the code format which helps keep our code uniform, but can be used in extension of ESLint which provides a powerful combo for keeping the code high quality and uniform.

Our application runs on Typescript. We use Typescript to enforce strong typing in our system. Typescript checks our code for type mismatching. This means if we try to call length() on a number, typescript will display it as an error. In addition to that Typescript allows us to set up types for variables, so we can benefit from the automatic variable and function detection. Typescript also allows us to set up types for parameters, so we know if a variable/object can be passed to a function.

Code Coverage

We used the testing library Jest to generate code coverage reports for our backend. Jest serves both statement-based coverage and decision based coverage. This helps us identify test cases for lines of code we haven't covered yet. These reports have to be taken with a grain of salt since we don't necessarily want to implement tests for all statements and decision in every file. We decided to focus on our backend services, since that is where most logic is applied. Another thing to note: Code coverage doesn’t include the quality of the tests. So it’s next to useless when considering how users are going to use the system. Our focus was on applying a good black-box testing strategy and then using that in tandem with coverage reports to have an overview of how much of the application was THOROUGHLY tested.

Overall the code coverage reports helped us make sure we had unit test for every relevant decision. But the decisions were tested, using black-box techniques.