“Unit Testing” CSS:

Snapshot Testing and Visual Regression

Visual regression is simply a fancy term for “before styling” and “after styling” comparison. In common practice, this is achieved by running an instance of a webpage or web-based program, and refreshing it after applying a style to see what changed. To take this a step further, support libraries, such as JEST, help to automate this process slightly by taking “snapshots” which are custom files that are able to be tested against each other (e.g. if snapshot A doesn’t match snapshot B, the test will fail). Additionally, these snapshots are meant to be easily deciphered by a human reviewer, making code reviews on said snapshots less cryptic.

According to JEST, snapshot testing is not visual regression testing. Visual regression tools take screenshots of web pages and compare the resulting images pixel by pixel. Snapshot testing compares unique snapshot files that generate based on the behavior of your component. Snapshot testing is incredibly good at a certain thing: making sure your component doesn’t change unexpectedly. This is useful when it comes to interactable features like clicking, dragging, or any other common interaction. In a way, then, it isn’t meant solely to test just your CSS, but the functionality of your React and CSS together, and thus would be nearly the same as our current test suite, but would require a whole host of changes.

Issues With Snapshot Testing

As stated above, snapshot testing is very useful when it comes to testing your React component and your CSS at the same time, but it has some interesting quirks not directly stated on the main JEST snapshot documentation page. For starters, JEST has a tendency to produce false negatives, especially on styled components, when interacting with CSS class names. For our project, this poses a huge problem, as everything in the primary CSS file is organized through a class name. With JEST, the class names it automatically generates update with even the most minor of changes, eventually exploding the directory that houses the snapshots as minor commits are made over time. Most notably, JEST interacts in odd ways with Toast, a React component we use extensively throughout our program, such as generating a snapshot that pretends text isn’t there when it should be.

In addition to this, the snapshot files are *long*, as in thousands of lines long. If that wasn’t enough, these files explode in size when they run into SVGs, of which our project uses a lot of, and the expectation is that you preserve every snapshot, eventually just bloating every clone from then on. Interestingly enough, JEST’s snapshot framework also works rather poorly in continuous integration systems. To recap, this method of “unit testing” CSS bloats the repository

with niche files, said niche files are also laden with hard to spot bugs, said niche files are thousands of lines long, and the framework doesn't play nice with other components currently in use throughout our project.

Alternatives: Storybook

Unlike seemingly most software that solves a unique issue, Storybook is free. Storybook is a framework that allows you to create components in a vacuum through mocking. It allows for the testing and documentation of UIs away from the main project, allowing for easier understanding and eventual component reuse. At face value, Storybook is incredibly powerful, as having an environment to test the UI thoroughly and reuse common components vastly improves the quality of frontend development as well as its overall efficiency. Unfortunately, Storybook is incredibly powerful if it's planned for at the start of a project, not near the end. In our project's case, incorporating Storybook now would require a refactor of the entire frontend to work alongside the Storybook framework and to utilize it as a bonafide UI testing platform as it is meant to be used.

Despite this, through viewing Storybook's documentation and free tutorials, this is an incredibly powerful tool when used from the beginning of a project, as it allows an effective transition from a Figma prototype to a UI development workshop before creating a whole user interface on a main branch. If our group had known of this and adopted this framework before, it would've allowed us to have extensive data on our UI's latency and which styles were being applied and from where, effectively saving headaches.