Jest Testing: Your test tools are all wrong

by Jesse Sanders in Testing / Frontend / Jest

I LOVE TESTING. Everyone loves the excitement of a new project. There is planning to do, new technology to work with, and the green field of possibilities gets everyone excited.

Getting Started Testing a Project?

We all agree that with this new project that we are going to do things right. We set lofty goals around test driven development (TDD) and 100% code coverage while we develop our testing strategy. As the project kicks off, we slowly have to let go of TDD and pair programming to meet a few deadlines and before we know it, tests are falling behind to meet the stringent deadlines coming down from management.

Reality begins to sink in… I LOVE the IDEA of testing, especially you testing YOUR code. Mine is probably fine.

The reality is that testing is hard. Testing takes a lot of time. As we write unit tests, we begin writing code to mock out our backend services and before we know it, the mocks feel like they have a life of their own warranting a separate project.

As developers, we become frustrated with the entire process and begin to do the minimum shallow tests to squeak through a code review. Then, as we begin to refactor our code to improve it, a massive number of tests begin to fail and it requires hours of combing through tests and mocks to figure out what changed and why everything is failing. To make matters worse, the tests take a long time to run, and on each iteration we have to execute the entire suite of tests to get to the test we are trying to fix.

Discouraged developers face a grim reality, either comment out the failing tests to meet the deadline, or spend the evening/weekend working through the massive list of failures.

Jest: A Better Way to Test

Jest might just be the testing tool you've been waiting for. It was created by the Facebook team in order to improve the testing process. It was originally created in 2015

Sign Up for Notifications

Recent posts

Angular Seed and Its Successors

Why UX Designers Should Care About Table Relationships

How to Use CodePush in React Native

When To Choose Angular

Big Data Purging in Oracle

but wasn't received well, even at Facebook. It was difficult to configure, confusing, and not intuitive. In 2017 a new version was released that promised zero configuration and an easy to learn syntax.

The features of Jest are impressive. It's super fast as it uses JSDOM, a JavaScript implementation of web standards suitable for testing, and doesn't have the overhead of a traditional browser. Each test is sandboxed as to not affect other tests. This leads to more reliable testing. The error reporting is a huge improvement over other test runners like Karma. No more having to dig through stack traces to figure out which tests failed and why.

Jest offers zero config for JavaScript applications, which makes setup a synch. The feature that separates Jest from the pack, though, is snapshot testing that allows the developer to focus on writing tests, not mocks.

Test Reporting with Jest

The test output is a lot more readable and understandable using Jest. The challenge with using other tools like Karma is having to dig through very large stack traces trying to locate the test(s) that failed, and why. The test output for Jest is very readable and makes finding errors a breeze.

```
~/repo/ngrx-jest-testing (master)
$ jest
 FAIL  src/sum.spec.ts
  ● adds 1 + 2 to equal 3

    expect(received).toBe(expected) // Object.is equality

    Expected value to be:
      4
    Received:
      3

      2 |
      3 | test('adds 1 + 2 to equal 3', () => {
    > 4 |   expect(sumFunc.sum(1, 2)).toBe(4);
      5 | });
      6 |

      at src/sum.spec.ts:4:29
```

Built-in Code Coverage Outputs

Another great feature about Jest is that code coverage is built-in and easy to use. Developers save time as the output is inline and not in an HTML file that needs to be opened/refreshed after every run. Further, the code coverage output can be piped through istanbul or other similar filters to comply with various CI tooling. Running tests with coverage is as simple as `jest --coverage`.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	94.26	78.57	81.82	93.14	
src	92.31	50	75	90.91	
setup-jest.ts	66.67	100	0	66.67	5
sum.ts	100	100	100	100	
test-config.helper.ts	100	50	100	100	10
src/app	100	100	100	100	
app.component.html	100	100	100	100	
app.component.ts	100	100	100	100	
src/app/core/services	70	66.67	33.33	62.5	
customer.service.ts	70	66.67	33.33	62.5	9,11,14
src/app/customers/components/customers	100	100	100	100	
customers.component.html	100	100	100	100	
customers.component.ts	100	100	100	100	
src/app/customers/containers/customers	100	66.67	100	100	
customers.component.html	100	100	100	100	
customers.component.ts	100	66.67	100	100	17
src/app/state/customer	95.77	85	83.33	95.08	
customer.actions.ts	100	100	100	100	
customer.effects.ts	83.33	72.73	40	81.25	22,25,26
customer.reducer.ts	100	100	100	100	
index.ts	100	100	100	100	

Jest CLI Matches Data for Testing Ease

Jest has several CLI features that make running tests even easier. Jest has a powerful CLI file matching feature to execute only the tests that you need to. For example, say I only want to run tests with the name customer in the filename. I can execute `jest

customer` and only those files matching the string “customer” will be selected for testing.

In addition, jest has a watch mode that extends this filtering even further while providing several other powerful tools. It’s as simple as executing `jest --watch`.

```
Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press q to quit watch mode.
 › Press Enter to trigger a test run.
```

From the watch interface, a developer can run all tests, just failed tests, filter test by filename using a regex, and filter tests by test name using a regex. This make focusing on just failing tests or specific tests in a functional area very easy. No more waiting for all the passing tests to run before getting to the failed tests. Focus on what is important.

Migrating to Jest Doesn't Mean Learning New Syntax

One big concern developers have about bringing in a new testing framework is having to learn a whole new setup and syntax. If you’ve been using Karma and Jasmine in your test suite, which is the default for an Angular CLI project, then Jest will be very familiar.

The Jest testing globals are based on Jasmine globals, so for most developers, it looks exactly the same. In addition, there is little to no need to refactor existing tests to work with Jest because they work as is.

Show Changes Over Time with Jest Snapshot Testing

The biggest difference between Jest and other testing frameworks is snapshot testing. Snapshots allow you to capture a “snapshot” of serializable values and compare them to previous values. This simplifies how we write our tests and allows us to analyze how state has changed over time. The serializable values can include JSON objects, HTML, or text, to name a few.

Snapshot testing compares the expectation to a stored snapshot and creates a snapshot if one doesn’t already exist. It performs a diff comparison on the snapshot and produces an output similar to a git diff. The snapshots are stored in a __snapshots folder and the files become part of the code review process.

You might be asking, what are the benefits of using snapshots? The biggest reason is our mocks can be minimal or even elimited. You don’t have to create and maintain complex JSON objects to compare our unit test results to. Imagine storing the expected result of your unit test and receiving a warning anytime it changes.

The same can be done for HTML. Since component views should be deterministic, e.g., if given the same inputs, they should produce the same output each time, then we can simplify our component tests and get better coverage by using snapshots.

```
exports[`AppComponent should render html 1`] = `
<div
  id="root1"
  ng-version="5.2.9"
>
  <router-outlet />
</div>
`;
```

Is Jest the Right Testing Solution for You?

Jest and snapshot testing is here to change how we think about tests, and reduces the

amount of time it takes to deliver quality tests. Updating and fixing tests after refactors becomes much easier, and developers begin to enjoy writing tests. Take Jest for a spin and see if you don't agree.

Author: Jesse Sanders, CEO & Enterprise Principal

You can reach me at:
Email: jesse.sanders@briebug.com
Follow me on Twitter: @JesseS_BrieBug

References
To learn more about marbles and the supported syntax, checkout:
https://github.com/ReactiveX/rxjs/blob/master/doc/marble-testing.md

Check out the GitHub repo for this blog:
https://github.com/briebug/ngrx-jest-testing

Check out my talk at ng-Conf on NgRx Jest Testing:
https://www.youtube.com/watch?v=d91uDEmbBUs

Slides from my talk:
http://briebug.github.io/presentations/ng-meetup/2018/02/ngrx-jest-testing

Better way to jest test | Jest CLI | Migrating to Jest | Snapshot testing

390 Union Blvd, Suite 600
Denver, CO 80228

888-679-2201