Week 7 assignment regarding testing.

I was a C developer a long time ago, and then did a bunch of Perl after that. However, I haven't developed in a language where I also had to write unit tests. I don't remember how we did unit tests in C, that was too long ago... It is interesting to think now about having to create this bookstore app and also have to consider the testing efforts needed. I have been in the build space for a long time, and as such have had to execute tests, fix broken tests, enable test reporting, fix test issues in builds, install/create/maintain testing tools and environments, but haven’t had to actually WRITE any tests. I know with most everything concerning coding, the team chooses and implements strategies for how to get their work done. So generically, a testing team chooses a strategy, then a framework and implements automation and that solves ~75% of the problems, but there is still the writing of tests and managing test data.

The last project I worked on, I specifically asked the testing team to write their tests to be executed as Gradle tasks, as our builds were in gradle and that would provide the easiest testing integration into our CICD pipeline. I didn’t care what testing tools they used, but for sure wanted them as gradle tasks and include input parameters to control what tests ran against what environments.

You had directed us towards investigating a handful of tools and I hadn't heard of any of them before. So I will spend some time seeing what it would take to implement one of these tools into my application. I started by looking for a sample GIT project I could clone just to kick the tires and not touch my existing project. I learn by doing, so I have to try something.

I found this project and cloned it locally and followed the provided instructions to execute:

https://github.com/the-road-to-learn-react/react-testing-mocha-chai-enzyme

Following are my reactions/observations towards using this cloned project to test a simple React app.

* Like using any third-party package, including these test tools require that you install and manage more packages, so in addition to keeping your development node\_modules up to date, you also have to update your testing packages.
* It was an easy example to play with. The first thing I had to do was try to make a test fail. The app in this example is just a counter and I switched the increment and decrement functions to force a test failure.
* One thing that I don't really understand is how it knows what button to interact with for a given test. I found this line in the test spec which explains a bit.

wrapper.find('button').at(1).simulate('click');

I then switched the at(1) and at(0), which I guessed would fix the test failure as I had switched the actual code in the App, and that did cause the tests to pass again. I know this is a simple example, but keeping track of widgets that way has to be confusing and there has to be different strategies for that.

* I had remembered testers using xpath, which is another naming convention that I don’t know much about. I can only assume that these test tools allow the querying of different types of elements in different ways. This line of thinking is taking me in a specific direction about how I assume there has to be another naming convention that developers use when naming elements in the UI so that the testing team can more easily identify widgets to be able to interact with them. I am thinking about how testing adds a WHOLE ADDITIONAL layer to development who already manage the format of the code with its naming conventions and will also have to maintain conventions used for the testing team.
* Looking at this project, there are other tools it is using. I see references to Sinon, which seem to handle async type requests. Cypress is the framework, and the project uses expect from chai and also other tools from enzyme. So there are a lot of moving parts when it comes to testing. Just like development.
* Having a common framework, and specific guidelines about testing would make this effort easier, and that would also include developer discipline for how to implement features. There is always discipline needed, and this adds more.
* So being a tester is a whole different skill, with best practices and common methodologies.

I had an anecdote to share about a previous work project from a number of years ago. I was responsible for the infrastructure that was used to run tests against a large suite of tools in a number of datacenters. I didn’t manage the actual tests, but the Jenkins jobs that ran the tests. Our team also used the tests to decide if a particular tool would be promoted through our environments. At different times, we would have test failures which would cause the deployment process to stop and would then require us to reach out to the tester and then possibly the developer of the tool. At the time I thought: “Why can’t these tests just work!!”? But now having more exposure in this class and in my previous work project, testing is a whole other project in and of itself. There are SOO many moving parts to keep track of. I see now that the rare test failures we had were a testament to the quality of QA and Development staff.

In my last job, I managed a small suite of API tests that ran in Postman to do post-deployment testing. There I learned that managing test data is a little project all by itself. Managing test users and credentials in addition to any additional data is yet more to keep track of.

So, this assignment and React class has opened my eyes a bit more to the added complexity that testing adds to any software project, and this class has been great for me to expand my horizons about what UI development is and to fill in some knowledge gaps about programming APIs and creating a front end project.