**Programming with JavaScript Notes**

**Week 3.4**

**Testing**

What is Testing?

How to developers test their code? First, examine a custom concatStrings function.

(see testing.js)

Adding comments

Downsides:

* Ambiguity
* Doesn’t follow any set structure

JavaScript has testing frameworks, so you don’t always have to rely on comments.

Strengths:

* No comments

The test syntax itself becomes expectation documenting.

Tests as expectation-documenting code

Code syntax that specifies the expected result of passing specific values to your functions.

Tests are better alternatives to comments in the source code because:

* They specify what expectations my source code is trying to satisfy.
* Tests are callable, so you can execute tests to check if expectations are met.

An example of using JEST to test a function can be found in (testing.js)

Here’s an example:

expect(concatStrings(“abc”, “def”)).toBe(“abcdef”);

In short, the use the function expect to pass the call to another function, add the arguments, then add to that, another testing function (.toBe) that should have the value that you expec the function to produce.

Testing code in this way ensures three things:

* Conciseness (there are only two function calls)
* Clarity (you clarify what arguments you’re providing)
* Repeatability (you can call it over and over)

The terminology includes:

Passing - Green

Failing – Red

Changing code structure without changing its functionality is known as refactoring.

Refactoring

Used for updating code, without affecting the results it produces.

It should only make the code run faster, since it is now optimized.

Red-Green Refractor Cycle

It’s likely you will often have both red and green tests. Treat red tests as a guide to how you need to improve your code to cater for unmet expectations. As you continue to refine your code in response to red tests, it becomes a cyclical activity. This is often described as the Red-Green refractor cycle.

This cycle is the basis of Test-Driven-Development (TDD) approach to programming.

First right a failing test, then write a source code so that the test now passes, then you optimize the source code without changing its results.

Testing is how you verify the expectations regarding the behavior of your code.

Types of Testing

Expectations are recorded as software requirements.

How is testing approached by different occupancies:

UX Designer: Making sure website looks and behaves as expectedly.

Project Manager on a Software Project: Making sure a specific piece of software works well with other parts of your system.

Software Engineer: Writing code that doesn’t break the existing functionality, is bug free, and satisfies the preferred requirements.

Based on your motivations, there are various ways of testing.

Three common types of testing:

* e2e
* Integration
* Unit

Example of e2e:  
Laptop manufacture lets employees take laptops off assembly line to test/use themselves.  
More specific to web dev, it imitates how a user would use the app.  
You’re basically testing the final product from the user experience.  
They take the longest and take the most time to setup and run.  
Here are three e2e tests:

1. Webdriver JS
2. Protractor
3. Cypress

Integration Testing

Tests how part of your system interact with one another.

Here are two Integration Tests:

1. React testing library
2. Enzyme

Integration tests are faster and cheaper than e2e tests, but not as fast or as cheap as Unit testing.

Unit Testing

The process of testing the smallest units of source code in isolation.

An example of this is functions or methods.

Unit tests are self-contained.

They’re meant to test code in isolation, preferably to test outside the source code.

Unit Tests > Integration Tests > e2e Tests

Introduction to Jest

Jest is a testing framework that you can use with JavaScript.

JS doesn’t have built-in objects/methods that allow for tests to be written.

Therefore, there are many libraries built for this function. Some examples are:

1. Jasmine
2. Mocha
3. Karma
4. qUnit
5. Jest

Jest can be used to test React, Babel, TypeScript, Node, Angular, View, and more.

It also supports code coverage.

Code Coverage:

A measure of what percentage of code is covered by tests.

If there is an 80% code coverage, that means that only one-fifth of my entire code base is not covered by tests. But even 100% code coverage doesn’t mean you’ve tested for every conceivable expectation.

The higher the code coverage, the less the chance of bugs.

The higher the code coverage, the lower the amount of time required to write new tests.

This depends on whether or not there are incomplete software requirements pending or if you are going to receive more requirements in the future.

Mocking

Mocking allows you to separate code from its related dependencies.

You can use mocking to ensure that your unit testing is stand-alone.

For example:  
You can test the front-end functionality of your web app by mocking the data as if it came back from a server when in fact it came from the client.

Mocking is especially helpful because very often web applications are built by teams of developers.

Some work on the front-end, some on the back-end, this can result in bottlenecks.

Try this example:

A team decides to build a new feature that lists the address book of users of the app on the front-end. The actual user related data for this would come from the server, but what if a back-end developer was late in their part of the feature? A front-end developer would be stuck waiting.

Mocks allow you to pretend that the users are already there. The needed data comes from the mock, rather the back-end. This allows the front-end developers to finish their site of the new feature independently.

Some libraries, such as sign-on, focus specifically on mocking.

The great thing about Jest, is that you use it’s mock functions without any additional installations.

In Jest, you use mocking by employing Jest mock functions.

Jest also allows for easy testing of asynchronous code.

Jest also allows you to perform snapshot testing.

Snapshot testing:

Used by developers to verify that there are no regressions in the DOM of our apps after some changes to the code base are made.

Writing Tests with Jest

Much of this work is done within the folder, *Testing-W-Jest*.  
  
Jest is an open-source testing framework built on JavaScript. Designed majorly to work with React and React Native-based web applications.

After building a Test script with Jest, ensure applying that configuration when running the code to test.

How to install the jest package to a specific directory:

npm install –save-dev jest

Then to run the test, type in the terminal:

npm run test

Additional Resources

MDN: Server-side website programming

<https://developer.mozilla.org/en-US/docs/Learn/Server-side>

Nodejs.org docs website

<https://nodejs.org/api/documentation.html>

Jest testing framework website

<https://jestjs.io/>

Cypress testing framework

<https://www.cypress.io/>

Npm website

<https://www.npmjs.com/>

Unit testing in JavaScript

<https://www.browserstack.com/guide/unit-testing-in-javascript>