Lesson | Monday

Intermediate JavaScript (/intermediate-javascript) / Test-Driven Development and Environments with JavaScript (/intermediate-javascript/test-driven-development-and-environments-with-javascript) / TDD Review

Text

We learned about the basics of Test-Driven Development (https://www.learnhowtoprogram.com/lessons/overview-of-test-driven-development-tdd-with-text-analyzer) in Introduction to Programming. So far, we have used the basic principles of TDD to help organize and plan our code — but we aren't actually executing any tests yet. Well, we're about to start doing that!

At this point, you will no longer need to add pseudocode tests in your READMEs. That was practice for writing actual tests which we can run with software, and that's exactly what we're about to start doing with Jest!

The Benefits of Test-Driven Development

Before we move on to learning more about the testing workflow and using Jest, let's look at all the advantages of TDD and why it's so helpful to the development process.

TDD Helps Us Create a Plan

By taking time to think through a program, identify behaviors, and tackle the simplest first, we're creating a game plan. We are laying out each step we need to take in detail, which leads to the next benefit.

It's Easier to Tackle Complex Problems

Even the most experienced coders can struggle with approaching complex issues. By identifying and testing the smallest possible behavior at a time, we can take baby steps towards solving bigger problems.

It Prevents Errors and Makes Bugs Easier to Locate

When we implement and test a single behavior at a time, we're preventing errors. Testing each behavior before moving on allows us to spot bugs as soon as they're introduced.

Without TDD, we might not spot some of these bugs until they break our application. To make matters worse, if we add more code after introducing a bug, other parts of our application will likely be reliant on our buggy code. To resolve this issue, we'd need to locate the bug, resolve it, and potentially alter all the code that depends on the buggy code. That's a lot of work.

It Allows Us to Create Projects Faster

Starting a complex project can be daunting. Even if we aren't sure how to code the more difficult features of our application, we can hit the ground running by implementing smaller, identifiable functionality. As we work, the bigger picture will come into focus.

New Features Are Built Upon Reliable Code

By testing functionality as it's implemented and ensuring previous tests still pass when new functionality is introduced, new features are always built on a foundation of reliable, tested code.

It Keeps Code DRY

By implementing the least amount of code to pass each test, we keep our code DRY. By approaching each piece of our project's functionality individually, we are likely to write more modular code. Modular code is easier to maintain, update, and debug.

It's Employable

Testing is an important part of the tech industry and many tech careers rely on it — ranging from developers and QA testers to devops specialists and site reliability engineers.

Writing Coded Tests

Up to this point, we've written pseudocode tests and then manually checked to see if our code works. This is a time-consuming and unrealistic way to test our code — and it's not what professionals do.

Now we're ready to level up our TDD skills by writing automated tests for each behavior. While writing coded tests may take longer at first, we'll be able to test all of our program's behaviors with a single command.

Previous (/intermediate-javascript/test-driven-development-and-environments-with-javascript/webpack-and-npm-practice)
Next (/intermediate-javascript/test-driven-development-and-environments-with-javascript/red-green-refactor-workflow)

disable dark mode

(http://www.epicodus.com)