Introduction to Testing with Mocha and Chai

This article provides a high-level overview of unit testing, why tests are important, and what the Mocha and Chai frameworks provide.

Introduction to Unit Testing with Mocha and Chai

This article provides a high-level overview of tests using the Mocha and Chai frameworks.

A familiar pattern on the Codecademy platform is that when you finish an exercise, your code runs, and instructive error messages might pop up if anything is wrong.

When you move off of the platform to get started with more independent projects, you will still be able to get helpful feedback from tests! To do this, you will be reading, running, and eventually even writing your own test scripts directly.

Why write tests for code?

Tests are common in software engineering because they help to document the core functionality of the code and make sure that new features do not introduce breaking changes.

In the software industry, code is generally written, maintained, and refactored by many different software engineers over many years. Having comprehensive tests enables engineers to confidently change or add to existing code knowing that their changes haven't broken other features or had unintended side effects elsewhere in the app.

Getting used to reading tests and following the error messages that failed tests will log for you in the terminal is a skill that takes practice. Cultivating it now will serve you well throughout your coding journey!

What is Unit Testing?

Unit testing means testing the behavior of code in small, independent units. Units are typically designed to be the smallest meaningful chunk of independently testable code. This is in comparison of integration testing, in

which a set of modules are tested as a group. Reading and writing unit tests is an important essential skill for most software engineers, whereas the way that integration tests are done varies widely from product to product.

Mocha and Chai, Test Suites and Test Cases

Mocha and Chai are two JavaScript frameworks commonly used together for unit testing.

Mocha is a testing framework that provides functions that are executed according in a specific order, and that logs their results to the terminal window.

When you read tests written in Mocha, you'll see regular use of the keywords `describe` and `it`. These keywords, provided by Mocha, provide structure to the tests by batching them into test suites and test cases.

A *test suite* is a collection of tests all relating to a single functionality or behavior. A *test case* or a *unit test* is a single description about the desired behavior of the code that either passes or fails. Test suites are batched underneath the `describe` keyword, and test cases are batched under the `it` keyword.

Additionally, Mocha provides tools for cleaning the state of the software being tested in order to insure that test cases are being run independently of each other. You might end up using other tools, to *stub* or *mock* the desired behaviors of other units that a given unit of code might interact with. The *independence of test cases* is a key principle of unit testing, as it allows the cause of errors to be pinpointed more specifically if a test case fails, thereby speeding up the debugging process.

Assertions

The base component of test cases are *assertions*. Assertions are tied to particular values (whereas test cases are descriptions of behavior) and they will fail if the expected value does not match the actual value.

Every assertion in a test case must be met in order for the test case to pass.

Chai is an assertion library that is often used alongside Mocha. It provides functions and methods that help you compare the output of a certain test

with its expected value. Chai provides clean syntax that almost reads like English!

Example of a Chai assertion: `expect(exampleArray).to.have.lengthOf(3);`

This code will check whether that the variable `exampleArray` has a length of three or not.

Failing and Passing Tests

Robust tests are accurate for both failing and passing conditions! When writing tests, you need to make sure that the test fails if the feature that it is testing was not implemented properly, as well as making sure that the test passes if it is. Tests that will erroneously pass can be enormously misleading, and might lead to broken code getting merged and deployed.

Continue on to Reading Tests and Running Tests and Interpreting Test Output when you are ready to learn more!