How to test JavaScript with Mocha— The Basics

Brandon Morelli Follow
Jun 7, 2017 · 6 min read

Mocha is one of the most popular Node.js testing frameworks and while it may seem daunting, it's actually pretty easy to get started with.

testing via upsplash.com

Preface

> *Hey! I'm Brandon. I created codeburst.io and I write JavaScript tutorials and articles to help beginners better understand the inner workings of Web Development. If you have any questions about the article, leave a comment and I'll get back to you, or find me on twitter @brandonmorelli. Lastly,*

> *when you're ready to really dive into Web Development, Check out the Best Courses for Learning Full Stack Web Development*

What exactly is this tutorial?

This tutorial is all about Mocha **basics**. By the end of this tutorial you'll have successfully written your first test with Mocha. You'll understand how to setup Mocha, how to group tests, and how to use an assertion library. Tomorrow I'll be releasing Part 2 which will focus on more advanced testing techniques, as well as actually integrating our tests with real code! ***Edit: Part 2 is live and can be found here.***

. . .

Mocha has great documentation. However, if you're *new-ish* to JavaScript, it can be a bit dense and difficult to understand. I'll be walking you through the first example direct from the Mocha Documentation. I'll provide additional insight and breakdown the concepts more than the documentation does to ensure you understand exactly what is going on.

Ready? *Lets get started!*

Install Mocha Globally

Install Mocha globally by running:

```
$ npm install -g mocha
```

When you install an npm module globally, you aren't limiting its use to your current project. Instead, you're able to access and use the module like a command line tool. Once Mocha is installed globally, we're able to run commands in our command line using the `mocha` keyword.

Create a project

Next, we'll create a project directory named `test` . In our test folder we'll create a file named `test.js` . Finally, we'll initialize our project by running `npm init` .

If you're not familiar, `npm init` is a simple way to interactively create a package.json file. Answer the questions and hit enter. The question that is most important here is '**test command:'**—respond with **'mocha'.** This way we can run mocha by simply typing `npm test` .

When finished, you should have a file structure that looks like this:

```
test
|-- test.js
|-- package.json
```

Your package.json file should also contain the following json:

```
"scripts": {
  "test": "mocha"
},
```

Once you have all of the above, we're ready to test!

Writing our first test

We're going to copy a test straight from the Mocha documentation, then I'll walk you through exactly what is happening. In your `test.js` file, copy the following:

```
var assert = require('assert');
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present',
function(){
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Now run our test in the command line using `npm test` and you should get the following:

```
Array
  #indexOf()
```

```
    ✓ should return -1 when the value is not present

  1 passing (9ms)
```

OUR TEST PASSED! That's cool. But we have no idea why… So lets break it down.

. . .

Remember, **Mocha is a testing frameworks**. That means it's used to organize and execute tests. When writing a test, there are two basic function calls you should be aware of: **`describe()`** and **`it()`**. Both are used in our above example.

- `describe()` is simply a way to group our tests in Mocha. We can nest our tests in groups as deep as we deem necessary. `describe()` takes two arguments, the first is the name of the test group, and the second is a callback function.

```
describe('string name', function(){
  // can nest more describe()'s here, or tests go here
});
```

Recall our earlier example. We have a test group named `Array` and inside of that a test group named `#indexOf()` and finally inside of that is our actual test.

- `it()` is used for an individual test case. `it()` should be written as if you were saying it out loud: "It should equal zero", "It should log the user in", etc. `it()` takes two arguments, a string explaining what the test should do, and a callback function which contains our actual test:

```
it('should blah blah blah', function(){
  // Test case goes here
});
```

. . .

Within our testing framework (Mocha), we can use assertion libraries. **An assertion library is a tool to verify things are correct -** It's what actually verifies the test results.

Note that we don't need to use an assertion library, but they make testing **way** easier. Mocha allows us to use any assertion library we wish. In the above example (and for all of the other examples), we're using Node.js' built-in assert module. Hence this line of code where we require the assert module:

```
var assert = require('assert');
```

There are a number of different *assertion tests* included with `assert`. The one we've already used is `assert.equal(actual, expected);` This tests equality between our actual and expected parameters using double equals (`==`).

Recall one last time our original example:

```
it('should return -1 when the value is not present',
function(){
  assert.equal(-1, [1,2,3].indexOf(4));
});
```

All we're doing here is testing if `[1,2,3].indexOf(4));` is equal to `-1`. If our expected parameter equals our actual parameter, the test passes. If it doesn't, the test fails.

Again, we can go to our command line and run our test with `npm test`

```
Array
  #indexOf()
    ✓ should return -1 when the value is not present

1 passing (9ms)
```

The result broken down line by line is:

1. Our first test group `Array`

2. Our nested test group `indexOf()`
3. A check mark indicating a passing test, along with the description of our test
4. A summary indicating we have 1 passing test and the testing took 9 milliseconds

Putting the pieces together

You have all of the pieces now, so lets put it all together. Here's our original test, commented out to explain every line:

```
// Require the built in 'assertion' library
var assert = require('assert');
// Create a group of tests about Arrays
describe('Array', function() {
  // Within our Array group, Create a group of tests for in
  describe('#indexOf()', function() {
    // A string explanation of what we're testing
    it('should return -1 when the value is not present', fu
      // Our actual test: -1 should equal indexOf(...)
```

Test your learning

Alright, it's time to test your learning. **Without scrolling down to see the answer, write the following test:**

1. Create a test group named Math
2. Create two tests within the group Math.
3. Test one: Should test if ***3*3 = 9***
4. Test two: Should test if ***(3–4)*8 = -8***

***** ***** ***** ***** ***** ***** ***** *****

Don't scroll down until you have two passing tests!

Don't scroll down until you have two passing tests!

Don't scroll down until you have two passing tests!

Don't scroll down until you have two passing tests!

***** ***** ***** ***** ***** ***** ***** *****

Test Answer

We're you able to do it? If not, that's ok! If you were, great work! Lets take a look the commented solution:

```
// Require the built in 'assertion' library
var assert = require('assert');
// Create a test suite (group) called Math
describe('Math', function() {
    // Test One: A string explanation of what we're testing
    it('should test if 3*3 = 9', function(){
      // Our actual test: 3*3 SHOULD EQUAL 9
      assert.equal(9, 3*3);
    });
    // Test Two: A string explanation of what we're testing
```

And when we run `npm test`

```
Math
  √ should test if 3*3 = 9
  √ should test if (3-4)*8 = -8

2 passing (13ms)
```

You did it.

Good work! You can now (ideally) understand what Mocha is, how to set it up, how to group tests, and how to use an assertion library. But what good is testing if you can't actually test your code? In tomorrows article we'll integrate tests into an existing JavaScript file! ***Edit: Part 2 is live. Check it out Here.***

I publish a few articles/tutorials each week, please enter your email here if you'd like to be added to my once-weekly email list.

❤ If this post was helpful, please hit the little blue heart! And don't forget to check out my other recent articles:

- What the heck is an Immediately-Invoked Function Expression?
- Three awesome courses for learning Node.js
- And when you're ready to really dive into Web Development, Check out the 5 Best Courses for Learning Full Stack Web Development