

Node.js Testing with Mocha, Chai, Sinon

MAY 05, 2019 BY ROBIN WIERUCH - [EDIT THIS POST](#)

 Follow on Twitter

17k

[Follow on Facebook](#)



This tutorial is part 2 of 2 in this series.

[Part 1: The minimal Node.js with Babel Setup](#)

This tutorial demonstrates how to setup testing with Mocha, Chai, and Sinon in Node.js. Whereas the previous tutorial has already shown you how to setup your Node.js application, this tutorial sets up a testing environment for your Node.js project. Let's dive into it by setting up Mocha with Chai and Sinon as testing framework.

NODE.JS WITH MOCHA

Mocha will be our test runner which is responsible for encapsulating our tests in test suites (describe-block) and test cases (it-block). Furthermore test runners like Mocha offer an [API](#) to run all tests from a command line interface. Let's dive into it: First, install

an additional Babel package for your Node.js application which makes our tests understand Babel enabled JavaScript code:

```
npm install @babel/register --save-dev
```

Second, install our test runner Mocha as node package:

```
npm install mocha --save-dev
```

And third, in your *package.json* include a test script which uses Mocha to execute our tests and the new Babel package to run all executed test files through Babel enabled JavaScript code:

```
{
  ...
  "main": "index.js",
  "scripts": {
    "start": "nodemon --exec babel-node src/index.js",
    "test": "mocha --require @babel/register 'src/**/*.spec.js'"
  },
  "keywords": [],
  ...
}
```



The last configuration tells Mocha which files should be identified as test files via a file name pattern matching mechanism. Since we used `**` in between, Mocha will run recursively through the *src/* folder to find all files in your application. In this case, all files with the ending *spec.js* are identified as test files. It's up to you to choose a different name matching (e.g. *test.js*) for your test files.

NODE.JS WITH CHAI

Chai will be our assertion library to run equality checks or other test related scenarios. It enables you to compare expected results against actual results (e.g. expect X to be true). First, let's install it on the command line as dependency for our project:

```
npm install chai --save-dev
```

Next, let's define our first test suite and test case in a *src/spec.js* file:

```
describe('test suite', () => {
```

```
describe('test suite', () => {  
  it('test case', () => {  
    ...  
  });  
});
```

So far, there is nothing related to Chai yet. The describe-blocks and it-blocks are provided from our test runner Mocha. You can have multiple test cases in a test suite and a test file can contain multiple test suites as well. Often one test suite tests one function's different outcomes with multiple test cases. When we run our test runner, all test cases will be checked for their assertion(s).

```
import { expect } from 'chai';  
  
describe('true or false', () => {  
  it('true is true', () => {  
    expect(true).to.eql(true);  
  });  
  
  it('false is false', () => {  
    expect(false).to.eql(false);  
  });  
});
```



These test cases don't test any specific logic from our application, but only demonstrate how an equality check is performed with booleans. You can run both tests from the command line with *npm test*, because we have defined this command as npm script in our *package.json* file.

NODE.JS TESTING

So far, we didn't test any implementation logic yet. Our previous test was standalone without any external dependencies of business logic from our application. However, in a real application you would want to test logic of your actual Node.js application. Let's say we have a function which sums up two integers in a *src/sum.js* file which needs to be tested:

```
function sum(a, b) {  
  return a + b;  
}  
  
export default sum;
```

The utility function gets exported, because it is used in other parts of our application.

However, even though if it would be only used in this one file without an export

statement, you can still export it for the sake of testing. Now, in our *src/spec.js* -- or maybe more specific *src/sum.spec.js* test file --, we could import the function and test it:

```
import { expect } from 'chai';

import sum from './sum.js';

describe('sum function', () => {
  it('sums up two integers', () => {
    expect(sum(1, 2)).to.eql(3);
  });
});
```

Congratulations, you have set up your first unit test in Node.js. When you run your tests again with `npm test`, you should see a successful test on the command line. If the test turns red because it failed, you need to check whether your business logic (or test) is set up correctly.



NODE.JS WITH SINON

Testing JavaScript primitives, complex objects and arrays with Chai in Mocha is a great start. Eventually you will run also in the case of testing functions to be called. Therefore you need a utility to spy, stub, or mock functions. Sinon is a powerful library that helps you with it. Let's first dive into the use case which we want to test and then how to test it with Sinon in Mocha and Chai. In a new *src/call-my-function.js* file implement the following function:

```
function callMyFunction(callback) {
  callback();
}

export default callMyFunction;
```

The function only takes another function as argument -- it is a higher-order function -- and simply calls this function. Let's use it in our *src/index.js* file:

```
import sum from './sum.js';
import callMyFunction from './call-my-function.js';

console.log(sum(1, 2));
```

```
callMyFunction(function() {  
  console.log('Hello world');  
});
```

How would we test this function to be called within the other function? Let's install Sinon on the command line as node package for our application and see how we can test it:

```
npm install sinon --save-dev
```

In a new *src/call-my-function.spec.js* file, let's write our test for this new higher-order function:

```
import { expect } from 'chai';  
  
import callMyFunction from './call-my-function.js';  
  
describe('callMyFunction function', () => {  
  it('calls the passed function', () => {  
    callMyFunction(callback);  
  
    expect(callback ???).to.eql(true);  
  });  
});
```

Now we can test it with a Sinon spy which is used instead of the empty function:

```
import { expect } from 'chai';  
import { spy } from 'sinon';  
  
import callMyFunction from './call-my-function.js';  
  
describe('callMyFunction function', () => {  
  it('calls the passed function', () => {  
    const callback = spy();  
  
    callMyFunction(callback);  
  
    expect(callback.called).to.eql(true);  
  });  
});
```

That's it. The test should be successful, because the function within our function to be tested is called. The Sinon spy switches the internal boolean flag for `called` from `false` to `true` after it has been called. You can find out more about Spies, Mocks, and Stubs from [Sinon's documentation](https://sinonjs.github.io/docs/spies).

Mocha and Chai are a popular combination of test runner and assertion library for Node.js applications. Sinon comes in as bonus if you need to make assertions on functions. You can find a ready to go setup Node.js application in this [GitHub repository](#). If you want to dive deeper into testing, you may want to check out this [GitHub repository](#) with a few tests where we are testing [reducer functions](#). The concept of reducers is a popular pattern in JavaScript which is a great candidate for unit testing.

Show Comments

KEEP READING ABOUT [DOCKER](#) >

HOW TO DOCKER WITH NODE.JS

Just recently I had to use Docker for my Node.js web application development. Here I want to give you a brief walkthrough on how to achieve it. First of all, we need a Node.js application. Either take...



NODE.JS TESTING WITH JEST

This tutorial demonstrates how to setup testing with Jest in Node.js. Whereas the previous tutorial has already shown you how to setup your Node.js application, this tutorial sets up a testing...





THE ROAD TO REACT

Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like **50.000+ readers.**

GET THE BOOK >

Get it on Amazon.



TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✓ Join 50.000+ Developers
- ✓ Learn Web Development
- ✓ Learn JavaScript
- ✓ Access Tutorials, eBooks and Courses
- ✓ Personal Development as a Software Engineer

SUBSCRIBE >

View our [Privacy Policy](#).

PORTFOLIO

[Online Courses](#)

[Open Source](#)

[Tutorials](#)

ABOUT

[About me](#)

[What I use](#)

[How to work with me](#)

[How to support me](#)

© Robin Wieruch



[Contact Me](#)

[Privacy & Terms](#)