# Getting Started with Node.js and Mocha

Mocha is a simple, extensible and fast testing library for Node.js. This article will walk you through its installation, configuration and usage.

[Mocha](http://mochajs.org/) is a testing library for [Node.js](http://nodejs.org/), created to be a simple, extensible, and fast testing suite. It's used for unit and integration testing, and it's a great candidate for BDD (Behavior Driven Development).

This article will walk you through its installation and configuration, as well as demonstrate its usage by implementing a color converter that can translate from RGB (red, green, blue) to hexadecimal color representation. In addition, we will create a web server that exposes the above converter, and demonstrates how to test asynchronous code.

**Bootstrap**

We will start by creating an empty directory.

mkdir converter

cd converter

Node's official package manager — [npm](https://www.npmjs.com/) — will help us set up an empty project. Npm's initcommand line option will launch a wizard, which creates a package.json for our project.

For the purpose of this tutorial, answer its questions as follows:

npm init

{

**"name"**: "converter",

**"version"**: "0.0.0",

**"description"**: "",

**"main"**: "index.js",

**"scripts"**: {

**"test"**: "echo \"Error: no test specified\" && exit 1"

},

**"author"**: "",

**"license"**: "ISC"

}

Our project's structure will contain an app directory that will contain executable code, and atest directory that is a standard place to put the Mocha test files:

mkdir app

mkdir test

Next, we will install our testing framework, and an expectation library called [Chai](http://chaijs.com/) that serves as a nice replacement for [Node's standard assert function](http://nodejs.org/api/assert.html).

npm install mocha --save

npm install chai --save

Note that we are using the --save option to automatically save these dependencies in ourpackage.json file.

During the server testing phase, we will need a way to send HTTP requests. [Request](https://github.com/request/request) is an excellent choice. Install it as follows:

npm install request --save

Finally, we will also need the [Express](http://expressjs.com/) package that defines a simple DSL (domain specific language) for routing and handling incoming HTTP requests:

npm install express --save

At this point, we are finished with the bootstrap process. However, we will configure one more thing to make running the test suite easier. We will set up the test command inside thepackage.json file, in order to run our tests simply by executing npm test from the command line.

The following command is used to invoke the Mocha binary installed locally in the./node\_modules directory:

./node\_modules/.bin/mocha --reporter spec

Note that we have also changed the standard reporter that Mocha uses, and set up the *spec*reporter that is more verbose, and thus more suitable for beginners. You can explore [other reporters on Mocha's official website](http://mochajs.org/#reporters).

Next, we will update the test command in package.json to contain the above command. That file should now look like this:

{

**"name"**: "converter",

**"version"**: "0.0.0",

**"description"**: "",

**"main"**: "index.js",

**"scripts"**: {

**"test"**: "./node\_modules/.bin/mocha --reporter spec"

},

**"author"**: "",

**"license"**: "ISC"

}

**Describing the Color Converter**

We will follow the TDD (test driven development) practice, and begin our project by creating tests before the implementation. Let's start by creating a test file:

touch test/converter.js

Mocha gives us the ability to *describe* the features that we are implementing by giving us adescribe function that encapsulates our *expectations*. The first argument is a simple string that describes the feature, while the second argument is a function that represents the body of the description.

Our color converter description:

describe("Color Code Converter", **function**() {

});

In the body of that description, we will create fine-grained segments that represent our feature in a little bit more detail:

describe("Color Code Converter", **function**() {

describe("RGB to Hex conversion", **function**() {

});

describe("Hex to RGB conversion", **function**() {

});

});

We will set up a concrete thing we are testing using the it feature. The it function is very similar to the describe function, except that we can only put *expectations* in the body of the itfunction. Let's use it for our color converter:

describe("Color Code Converter", **function**() {

describe("RGB to Hex conversion", **function**() {

it("converts the basic colors", **function**() {

});

});

describe("Hex to RGB conversion", **function**() {

it("converts the basic colors", **function**() {

});

});

});

We can now introduce our first expectation. We will use the Chai library and its expect keyword to compare the result of our feature's implementation and the result we *expect* to get. Of course, we first need to require the Chai library:

**var** expect = require("chai").expect;

**var** converter = require("../app/converter");

describe("Color Code Converter", **function**() {

describe("RGB to Hex conversion", **function**() {

it("converts the basic colors", **function**() {

});

});

describe("Hex to RGB conversion", **function**() {

it("converts the basic colors", **function**() {

});

});

});

Note that our implementation code doesn't exist at this point, but this need not bother us, and we can write our tests as if the implementation already exists.

In the body of our first it function, we will test the rgbToHex color converter. As its name says, it converts from a (red, green, blue) tuple to a hexadecimal color representation. We will test it by converting the basic colors:

**var** expect = require("chai").expect;

**var** converter = require("../app/converter");

describe("Color Code Converter", **function**() {

describe("RGB to Hex conversion", **function**() {

it("converts the basic colors", **function**() {

**var** redHex = converter.rgbToHex(255, 0, 0);

**var** greenHex = converter.rgbToHex(0, 255, 0);

**var** blueHex = converter.rgbToHex(0, 0, 255);

expect(redHex).to.equal("ff0000");

expect(greenHex).to.equal("00ff00");

expect(blueHex).to.equal("0000ff");

});

});

describe("Hex to RGB conversion", **function**() {

it("converts the basic colors", **function**() {

**var** red = converter.hexToRgb("ff0000");

**var** green = converter.hexToRgb("00ff00");

**var** blue = converter.hexToRgb("0000ff");

expect(red).to.deep.equal([255, 0, 0]);

expect(green).to.deep.equal([0, 255, 0]);

expect(blue).to.deep.equal([0, 0, 255]);

});

});

});

Notice the .to.deep.equal part of the above snippet. It is called a *matcher*, and it matches the result of a feature with an expected value. There are of course [many other matchers defined in the Chai libary](http://chaijs.com/api/bdd/), that can match expectations by various criteria. For example, to check the equality of two simple objects, we could write:

expect(1+1).to.equal(2)

In the above example, we have used the deep.equal because we were comparing two nested objects. The .deep part tells Chai to match all the elements of the arrays, one by one.

If we run npm test at this point, it will complain about a missing implementation file for our color converter:

module.js:340

throw err;

^

Error: Cannot find module '../app/converter'

npm ERR! Test failed. See above **for** more details.

npm ERR! not ok code 0

Don't worry, this is perfectly normal.

**Implementing the Color Converter**

We will implement our converter in the following file:

touch app/converter.js

The first description for our color converter describes a RGB to hexadecimal color conversion. It takes three arguments, and returns a string that represents the color in its hexadecimal representation.

We will use the toString method of the incoming numbers, combined with the number 16, to invoke a conversion to a hexadecimal representation.

exports.rgbToHex = **function**(red, green, blue) {

**var** redHex = red.toString(16);

**var** greenHex = green.toString(16);

**var** blueHex = blue.toString(16);

**return** pad(redHex) + pad(greenHex) + pad(blueHex);

};

**function** pad(hex) {

**return** (hex.length === 1 ? "0" + hex : hex);

}

Notice that we have padded the the resulting value with a zero prefix if it is only one character long, because a valid hexadecimal representation must always contain two characters.

To implement the function that converts from hexadecimal to RGB representation, we will use the parseInt function with the base 16 to convert parts of the incoming strings into valid decimal RGB values.

exports.hexToRgb = **function**(hex) {

**var** red = parseInt(hex.substring(0, 2), 16);

**var** green = parseInt(hex.substring(2, 4), 16);

**var** blue = parseInt(hex.substring(4, 6), 16);

**return** [red, green, blue];

};

After this step, we can run our tests with npm test and should see that all our tests are passing:

Color Code Converter

RGB to Hex conversion

✓ converts the basic colors

Hex to RGB conversion

✓ converts the basic colors

2 passing (8ms)

**Implementing the Web Server**

In this step, we will expose the color converter through a HTTP API, and demonstrate writing tests for asynchronous code using Mocha.

First, we will create a test file:

touch test/server.js

The same as in the previous test, we will require chai. To test the HTTP request, we will also require the request package.

**var** expect = require("chai").expect;

**var** request = require("request");

Here is the description of the functionality we want to implement, nicely laid out with Mocha's descriptions:

describe("Color Code Converter API", **function**() {

describe("RGB to Hex conversion", **function**() {

it("returns status 200", **function**() {});

it("returns the color in hex", **function**() {});

});

describe("Hex to RGB conversion", **function**() {

it("returns status 200", **function**() {});

it("returns the color in RGB", **function**() {});

});

});

We will store the full path to the resource we want to test in a variable. Before running the tests, we will run our web server on the localhost port 3000.

describe("Color Code Converter API", **function**() {

describe("RGB to Hex conversion", **function**() {

**var** url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

it("returns status 200", **function**() {});

it("returns the color in hex", **function**() {});

});

describe("Hex to RGB conversion", **function**() {

**var** url = "http://localhost:3000/hexToRgb?hex=00ff00";

it("returns status 200", **function**() {});

it("returns the color in RGB", **function**() {});

});

});

Note that in a bigger test suite, it is probably easier and nicer to put the host part of the URLs in a global constant, and reuse it in all tests.

To make a request, we will use the Request package. We need to pass two arguments to it: a URL to visit, and a function to be invoked when the request is completed. We will set up our expectations inside those callback functions.

describe("Color Code Converter API", **function**() {

describe("RGB to Hex conversion", **function**() {

**var** url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

it("returns status 200", **function**() {

request(url, **function**(error, response, body) {

expect(response.statusCode).to.equal(200);

});

});

it("returns the color in hex", **function**() {

request(url, **function**(error, response, body) {

expect(body).to.equal("ffffff");

});

});

});

describe("Hex to RGB conversion", **function**() {

**var** url = "http://localhost:3000/hexToRgb?hex=00ff00";

it("returns status 200", **function**() {

request(url, **function**(error, response, body) {

expect(response.statusCode).to.equal(200);

});

});

it("returns the color in RGB", **function**() {

request(url, **function**(error, response, body) {

expect(body).to.equal("[0,255,0]");

});

});

});

});

If we run the above code, something strange will happen. Nothing will fail, but, at the same time, no expectation will be checked. This happens because we didn't give Mocha enough time to wait for the requests to finish. In other words, the code inside the request's callback is never actually executed.

Fortunately, Mocha gives us a nice abstraction for this issue. For every it that needs to wait for a response value, we will inject a done callback function and call it only when our expectations were executed. This way, Mocha will know it needs to wait for some of the expectations.

describe("Color Code Converter API", **function**() {

describe("RGB to Hex conversion", **function**() {

**var** url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

it("returns status 200", **function**(done) {

request(url, **function**(error, response, body) {

expect(response.statusCode).to.equal(200);

done();

});

});

it("returns the color in hex", **function**(done) {

request(url, **function**(error, response, body) {

expect(body).to.equal("ffffff");

done();

});

});

});

describe("Hex to RGB conversion", **function**() {

**var** url = "http://localhost:3000/hexToRgb?hex=00ff00";

it("returns status 200", **function**(done) {

request(url, **function**(error, response, body) {

expect(response.statusCode).to.equal(200);

done();

});

});

it("returns the color in RGB", **function**(done) {

request(url, **function**(error, response, body) {

expect(body).to.equal("[0,255,0]");

done();

});

});

});

});

When we run our tests again with npm test, all the expectations will fail. Of course, this happens because we don't have a matching implementation for our color converter yet.

Color Code Converter

RGB to Hex conversion

✓ converts the basic colors

Hex to RGB conversion

✓ converts the basic colors

Color Code Converter API

RGB to Hex conversion

1) returns status 200

2) returns the color in hex

Hex to RGB conversion

3) returns status 200

4) returns the color in RGB

2 passing (29ms)

4 failing

**A Web API for the Color Converter**

Let's create a new file for our API implementation:

touch app/server.js

We will use the Express framework to create a simple web API:

**var** express = require("express");

**var** app = express();

We will also include our previous color converter implementation:

**var** express = require("express");

**var** app = express();

**var** converter = require("./converter");

Express gives us a nice DSL for defining the routes for our API:

**var** express = require("express");

**var** app = express();

**var** converter = require("./converter");

app.get("/rgbToHex", **function**(req, res) {

});

app.get("/hexToRgb", **function**(req, res) {

});

app.listen(3000);

The listen method runs a web server on the provided port (in our case port 3000).

The only thing left is to parse our incoming parameters, pass them to the converter, and send the results back to the client.

**var** express = require("express");

**var** app = express();

**var** converter = require("./converter");

app.get("/rgbToHex", **function**(req, res) {

**var** red = parseInt(req.query.red, 10);

**var** green = parseInt(req.query.green, 10);

**var** blue = parseInt(req.query.blue, 10);

**var** hex = converter.rgbToHex(red, green, blue);

res.send(hex);

});

app.get("/hexToRgb", **function**(req, res) {

**var** hex = req.query.hex;

**var** rgb = converter.hexToRgb(hex);

res.send(JSON.stringify(rgb));

});

app.listen(3000);

To run our tests, we will first need to run our server:

node app/server.js

We can now run our tests with npm test in a separate shell session:

Color Code Converter

RGB to Hex conversion

✓ converts the basic colors

Hex to RGB conversion

✓ converts the basic colors

Color Code Converter API

RGB to Hex conversion

✓ returns status 200

✓ returns the color in hex

Hex to RGB conversion

✓ returns status 200

✓ returns the color in RGB

6 passing (50ms)

**Summary**

Mocha is a nice and easy DSL that makes writing tests a great pleasure. Mocha's strength comes from its modularity, and while in this tutorial we didn't use any other library other thanchai, I encourage you to explore other libraries such as [Sinon](http://sinonjs.org/) and [Should.js](https://github.com/tj/should.js).

* [Testing with Mocha.js](http://brianstoner.com/blog/testing-in-nodejs-with-mocha/)
* [Mocha Documentation](http://mochajs.org/)
* [Chai Documentation](http://chaijs.com/)