**Table of contents:**

* [Creating a Hello World website with Node.js](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#creating-a-hello-world-website-with-nodejs)
  + [Install Node.js](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#install-nodejs)
  + [Install NPM and Mocha](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#install-npm-and-mocha)
  + [Create Hello World with Express framework](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#create-hello-world-with-express-framework)
  + [Details of Hello World](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#details-of-hello-world)
  + [Run the app](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#run-the-app)
* [Configuring unit tests with Mocha and Chai](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#configuring-unit-tests-with-mocha-and-chai)
  + [Install Mocha and Chai](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#install-mocha-and-chai)
  + [Add a test file](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#add-a-test-file)
  + [Grouping tests](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#grouping-tests)
* [Automating Mocha and Chai tests](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#automating-mocha-and-chai-tests)
  + [Version control](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#version-control)
  + [Continuous Integration with Buddy](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#continuous-integration-with-buddy)
* [Summary](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#summary)

**Creating a Hello World website with Node.js**

As usual, we shall begin with creating a simple Node.js application. The latter part of this guide will cover writing, running, and automating tests with BuddyWorks.

If you already know how to write Mocha/Chai tests, you can jump forward to to the [**Automation part**](https://buddy.works/guides/how-automate-nodejs-unit-tests-with-mocha-chai#automating-mocha-and-chai-tests).

**Install Node.js**

If you've never worked with Node.js before, start with installing the npm manager: [**nodejs.org/en/download/package-manager**](https://nodejs.org/en/download/package-manager/)

**Install NPM and Mocha**

Create a directory for the application:

Copymkdir myapp && cd myapp

Now initalize npm. We'll use it to create a package.json with the Mocha framework:

Copynpm init

When asked for the details of the application provide the following:

* name: hello-world
* entry point: app.js
* test command: ./node\_modules/.bin/mocha We shall use this framework to test the application

You can confirm the rest of the values with enter.

**Create Hello World with Express framework**

To build the app, we'll use Express Node.js web application framework:

npm install express --save

Using --save will add this package to package.json where all dependencies are stored

**Details of Hello World**

With everything installed, we can create an app.js file with a simple HTTP server that will serve our Hello World website:

Copy//Load express module with `require` directive

var express = require('express')

var app = express()

//Define request response in root URL (/)

app.get('/', function (req, res) {

res.send('Hello World')

})

//Launch listening server on port 8080

app.listen(8080, function () {

console.log('App listening on port 8080!')

})

**Run the app**

The application is ready to launch:

Copy$ node app.js

Go to http://localhost:8080/ in your browser to view it.

**Configuring unit tests with Mocha and Chai**

Every application requires testing before the deployment to the server, especially a welcome site that determines the first impression. In this example we shall use Mocha as the test running framework, and Chai as the assertion library.

**Install Mocha and Chai**

Let's add Mocha and Chai packages to the package.json:

Copynpm install mocha chai --save-dev

**Add a test file**

Time to define our first test. We shall keep all testing files in a separate /test directory (orndung muss sein):

Copymkdir test

Now, add the first testing file:

Copytouch test/test-pages.js

The test will verify the content of the websit. For that, we need an HTTP client: [**https://www.npmjs.com/package/request**](https://www.npmjs.com/package/request)

Copynpm install request --save-dev

The file should look like this now:

Copyvar expect = require('chai').expect;

var request = require('request');

it('Main page content', function(done) {

request('http://localhost:8080' , function(error, response, body) {

expect(body).to.equal('Hello World');

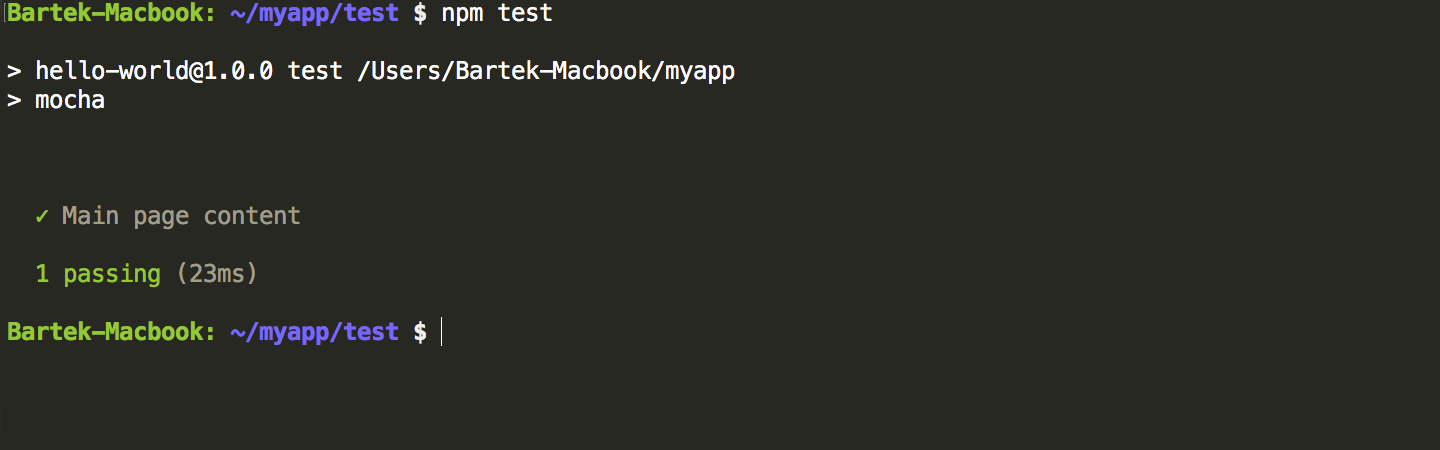
done();

});

});

Run the file to trigger the tests:

Copynpm test

[](https://buddy.works/data/blog/_images/mocha-chai/mocha-chai-1.png)Test results

Let's add some more tests that will check the status of the homepage and /about page:

Copyvar expect = require('chai').expect;

var request = require('request');

it('Main page content', function(done) {

request('http://localhost:8080' , function(error, response, body) {

expect(body).to.equal('Hello World');

done();

});

});

it('Main page status', function(done) {

request('http://localhost:8080' , function(error, response, body) {

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

done();

});

});

it('About page content', function(done) {

request('http://localhost:8080/about' , function(error, response, body) {

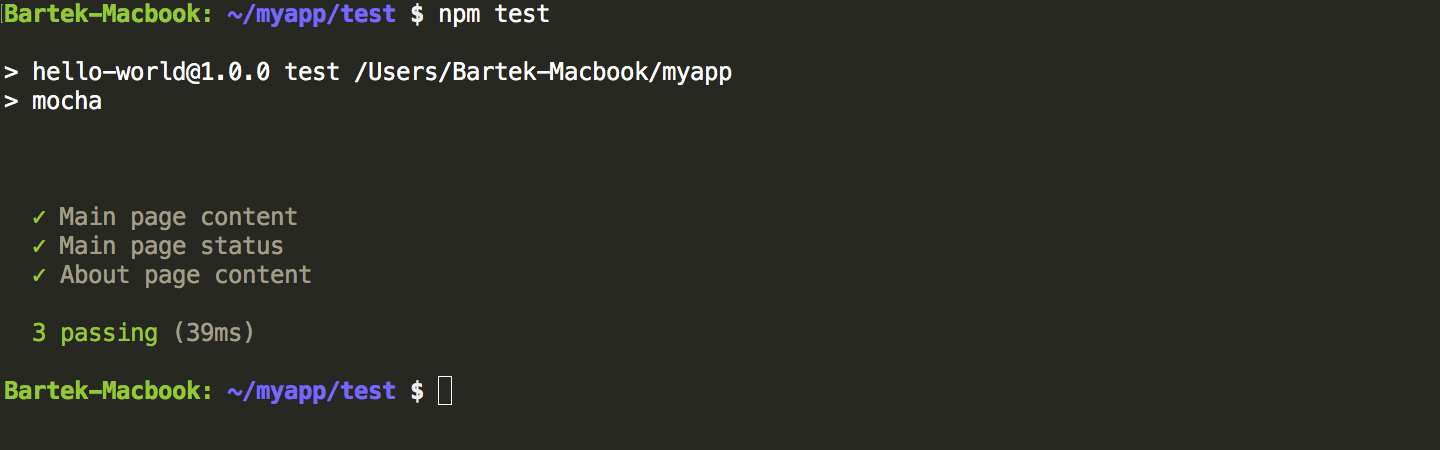
expect(response.statusCode).to.equal(404);

done();

});

});

Run npm test again and see the results. The /about page is not ready yet so the result will be 404:

[](https://buddy.works/data/blog/_images/mocha-chai/mocha-chai-2.png)Expanded tests results

**Grouping tests**

A very useful feature in Mocha is describe(), a function that allows you to better control your tests by grouping them:

Copyvar expect = require('chai').expect;

var request = require('request');

describe('Status and content', function() {

describe ('Main page', function() {

it('status', function(done){

request('http://localhost:8080/', function(error, response, body) {

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

done();

});

});

it('content', function(done) {

request('http://localhost:8080/' , function(error, response, body) {

expect(body).to.equal('Hello World');

done();

});

});

});

describe ('About page', function() {

it('status', function(done){

request('http://localhost:8080/about', function(error, response, body) {

expect(response.statusCode).to.equal(404);

done();

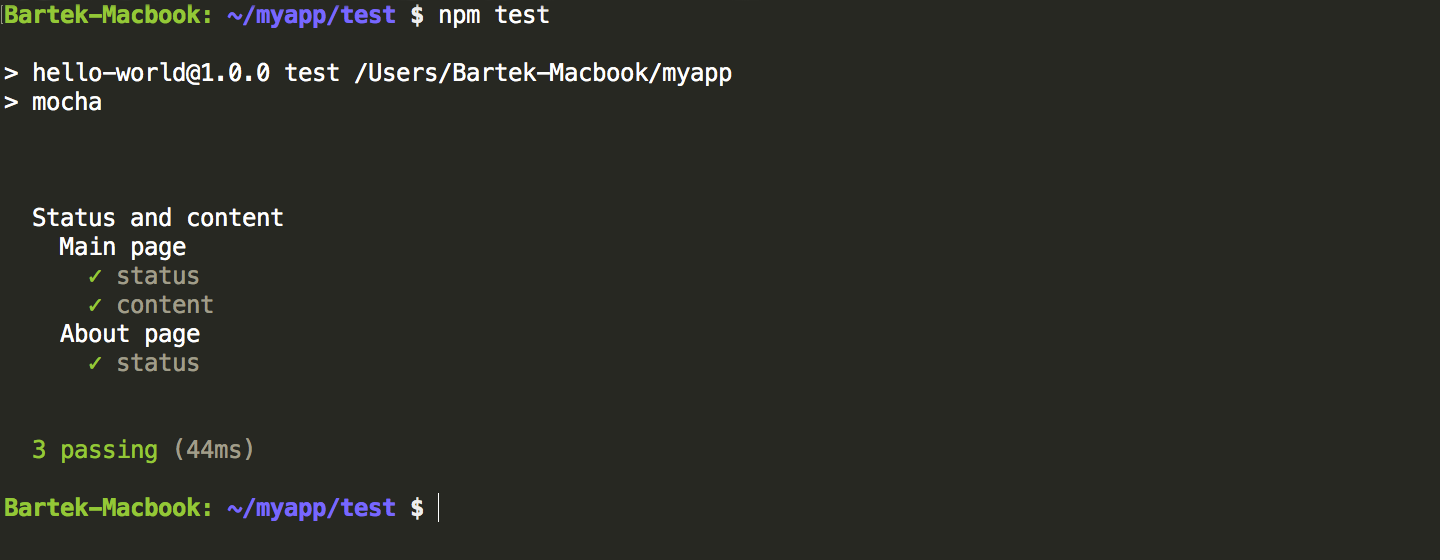
});

});

});

});

Run npm test yet again to see how the results are different:

[](https://buddy.works/data/blog/_images/mocha-chai/mocha-chai-3.png)Test results with describe()

**Automating Mocha and Chai tests**

As we mention in every single guide, all changes to code should be tested for errors. This is what Continuous Integration is about: making sure that your code is verified and prepared for deployment to production.

**Version control**

The application code, including package.json should first be put under version control. You shouldn't, however, add node\_modules under version control because dependencies are, well, independent from the project:

Copygit init

echo 'node\_modules' > .gitignore

git add \*

git commit -m 'my first commit'

Now, if you want to cooperate on a project with other developers (and automate your tests!) you need to add a remote location on Git hosting service and push the code to it. Create a new project in [BuddyWorks](https://buddy.works/), GitHub or Bitbucket, copy the URL of the remote, and push the project:

Copygit remote add origin URL

git push origin --all

**Continuous Integration with Buddy**

With your project under version control, you can use Buddy to streamline the testing process down to a single push to the repository:

1. Sign up at [**https://buddy.works**](https://buddy.works/)
2. Select the Git provider with your Node.js project
3. Create a new pipeline and set it to automatic mode (on push)
4. Add **Node.js** action (npm install and npm test are already entered)
5. (Optional) Add **SFTP** action to automatically upload your app to the server
6. (Optional) Add **Slack** action in the 'On failure' section at the bottom for automatic Slack notification in case some of the tests have failed

From now on, every time you'll make a push to the selected branch, Buddy will automatically test and deploy your application + let you know in case something goes wrong. Of course, you can choose any other type of upload or notification. Here's the whole process summed up in one awesome video:

0