**How to run Mocha/Chai unit tests on Node.js apps**

**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:

mkdir myapp && cd myapp

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

npm 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:

//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:

$ 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:

npm install mocha --save

npm install chai --save

**Add a test file**

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

mkdir test

Now, add the first testing file:

touch 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)

npm install request --save

The file should look like this now:

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

var request = require('request');

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

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

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

});

});

Run the file to trigger the tests:

npm test

Test results

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

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

var request = require('request');

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

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

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

});

});

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

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

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

});

});

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

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

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

});

});

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

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:

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

var request = require('request');

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

describe ('Main page', function() {

it('status', function(){

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

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

});

});

it('content', function() {

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

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

});

});

});

describe ('About page', function() {

it('status', function(){

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

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

});

});

});

});

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

Test results with describe()