Backend Testing

Steps And Instructions For Unit Testing using Mocha and Chai.

SE Boot camp

Agenda

1 **Testing Assertion Libraries and Automation**
Why using Mocha with Chai and Should?

2 **Environment Setup**
Installation and Setup needed for Testing “ Mocha - Chai – Should”

3 **Files Structure**
How to develop your Files Hierarchy in a clean way

4 **Imports and Requires**
Now!! Start Coding

5 **Mocha Divisions**
Describe – Before – it – After

Testing Assertion Libraries and Automation

You need to First Understand Differences

Testing Framework & Assertion Libraries

Mocha

Chai

Automation

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun.

Tests are written in

Chai is an assertion library, similar to Node's built-in assert. It makes testing much easier by giving you lots of assertions you can run against your code.

Environment Setup & Installation

You need to Prepare for your work

Dependencies Installation

You have to include the testing libraries in your project before you start

Main Testing Library “Mocha”

```
$ npm install --global mocha
```

Our Assertion Library “Should”

```
$ npm install --global should
```

Testing Writing Platform “Chai”

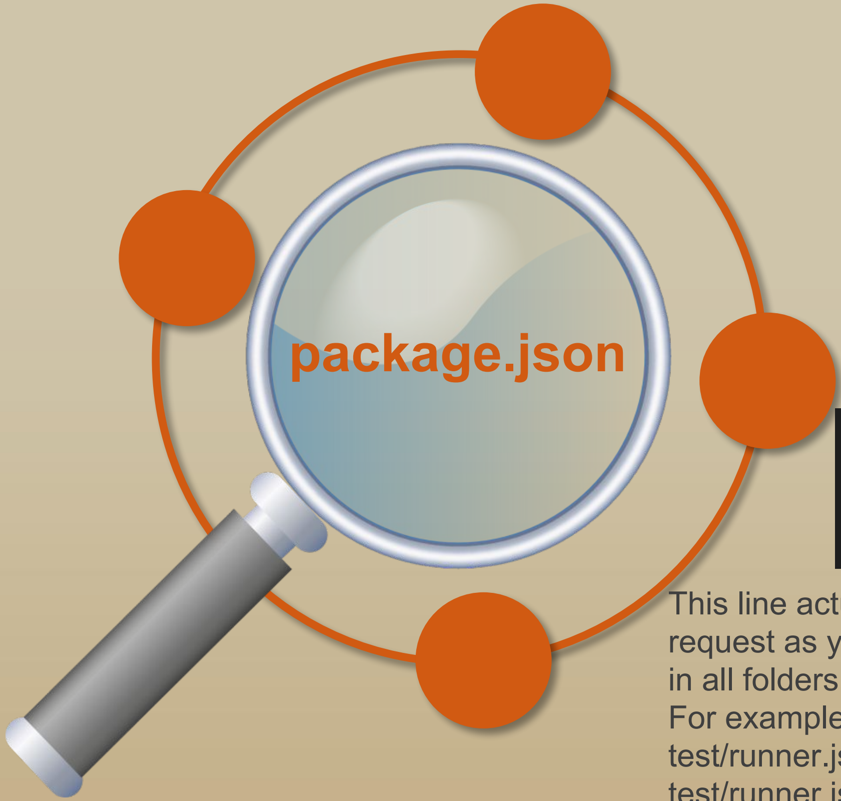
```
$ npm install --global chai
```

Http Requests using Chai

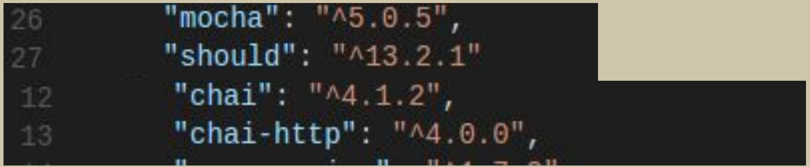
```
$ npm install --global chai-http
```

Environment Completion

Final Act to complete your Testing Environment

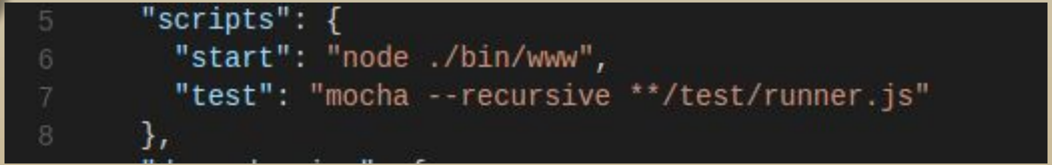


1- Make Sure Every Dependency is defined



```
        "mocha": "^5.0.5",
        "should": "^13.2.1"
         "chai": "^4.1.2",
         "chai-http": "^4.0.0",
```

2- You have to specify where are your Test files that will run When actually Testing.

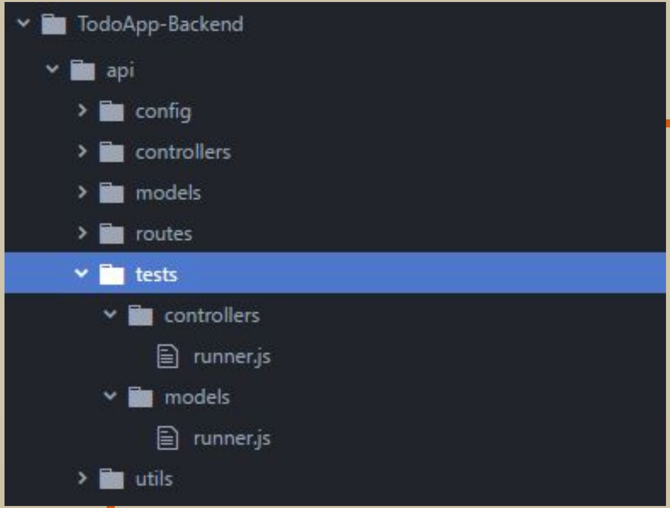


```
  "scripts": {
    "start": "node ./bin/www",
    "test": "mocha --recursive **/test/runner.js"
  },
```

This line actually means: When you run npm test, Mocha will get the request as your main Testing Library. And then will do a recursive test in all folders that have a test folder inside it and runner.js file.
For example: Mocha will recursively enter Models Folder and search for test/runner.js. Then enter Controller Folder and search for test/runner.js. And so on..

Files Structure & Hierarchy

You need to stick to the conventions of the programmers hierarchy

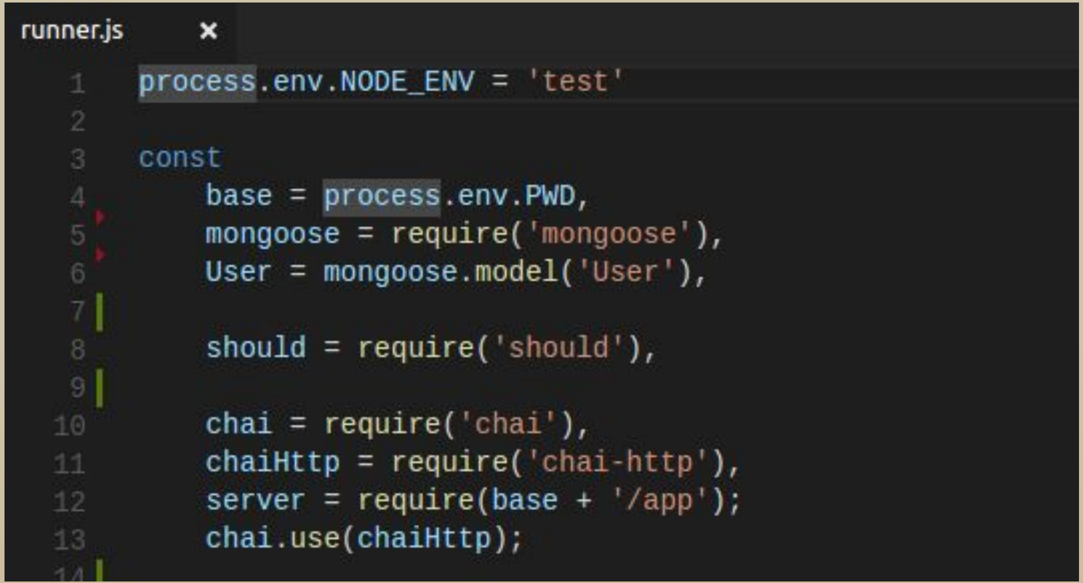

```
TodoApp-Backend
  api
    config
    controllers
    models
    routes
    tests
      controllers
        runner.js
      models
        runner.js
    utils
```

Files Hierarchy

- We have defined the testing folder.
- So we need to first test the controllers "Backend Functions".
- Therefore We have created a test folder inside test directory, containing runner.js file.
- That runner.js file will only be responsible for testing controllers files "auth.controller, and list.controller".
- Later on, if we need to test models for example.
- WHAT WILL WE DO??

Imports and require variables

Now we have started to code :D


runner.js

```
process.env.NODE_ENV = 'test'

const
    base = process.env.PWD,
    mongoose = require('mongoose'),
    User = mongoose.model('User'),

    should = require('should'),

    chai = require('chai'),
    chaiHttp = require('chai-http'),
    server = require(base + '/app');
    chai.use(chaiHttp);
```

First Line, You have to define that this is a test file

You will need them to initiate your database connection, disconnect from database, and inserting or removing from the User database.

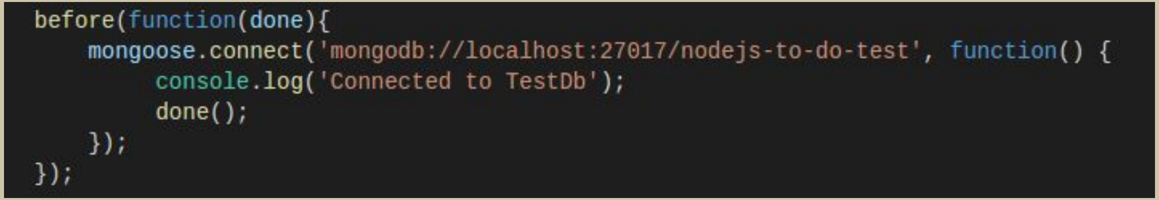
Require the assertion library “should”.

Require the testing library “chai”
You will need chai-http in order to send request to your backend
Server imports your app in the http request

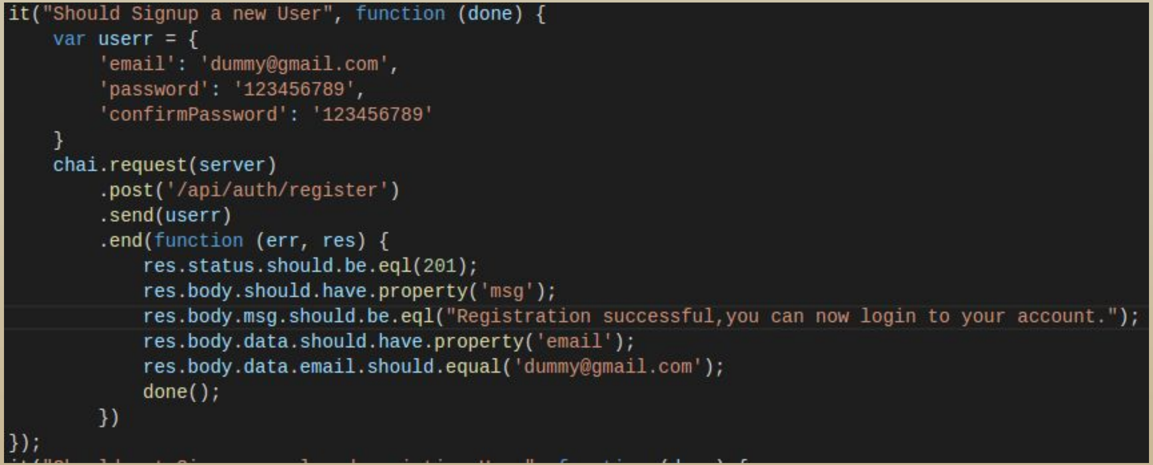
Mocha Divisions

Your code will be divided with mocha in a simple way

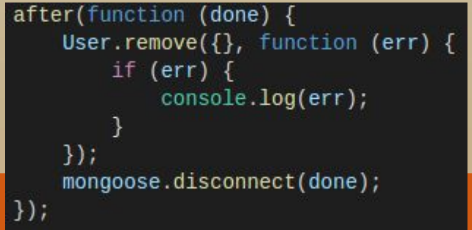
More on Mocha Divisions



```
before(function(done){
    mongoose.connect('mongodb://localhost:27017/nodejs-to-do-test', function() {
        console.log('Connected to TestDb');
        done();
    });
});
```



```
it("Should Signup a new User", function (done) {
    var userr = {
        'email': 'dummy@gmail.com',
        'password': '123456789',
        'confirmPassword': '123456789'
    }
    chai.request(server)
        .post('/api/auth/register')
        .send(userr)
        .end(function (err, res) {
            res.status.should.be.eql(201);
            res.body.should.have.property('msg');
            res.body.msg.should.be.eql("Registration successful,you can now login to your account.");
            res.body.data.should.have.property('email');
            res.body.data.email.should.equal('dummy@gmail.com');
            done();
        })
});
```



```
after(function (done) {
    User.remove({}, function (err) {
        if (err) {
            console.log(err);
        }
    });
    mongoose.disconnect(done);
});
```

Before block

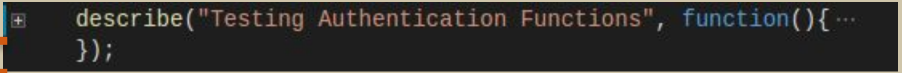
Here we are creating a test database and connecting to it. Before function takes a callback done().

It block

1st define any variables you will use or send in the chai request.
2nd chai will request a "post, get, patch, delete, etc.." on t=your app server.
3rd The response should have specifications depending on the request you have send.

After block

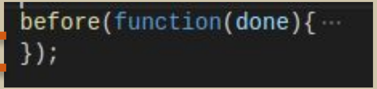
Here you empty the dummy values inside your database, and then close the connection with your database.



```
describe("Testing Authentication Functions", function(){ ···
});
```

Describe block

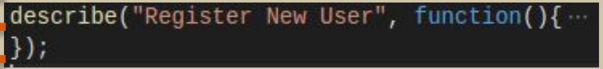
First Thing you write in the code.
It always refers to what generally You are testing.
Now we are referring to testing the whole Authentication file.



```
before(function(done){ ···
});
```

Before block

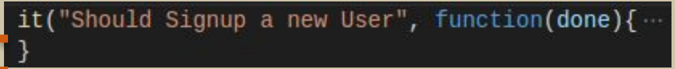
Inside describe block. There is before, usually you set you db connection and setup any other dependency here.



```
describe("Register New User", function(){ ···
});
```

Mini describe block

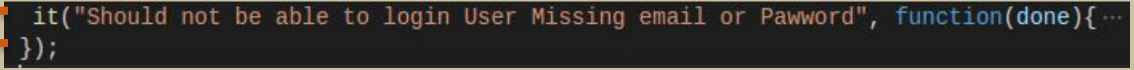
Inside the larger describe block. Usually defines what exactly inside the file Authentication you will test



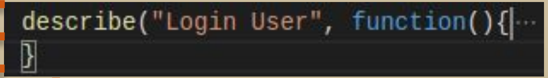
```
it("Should Signup a new User", function(done){ ···
}
```

It block

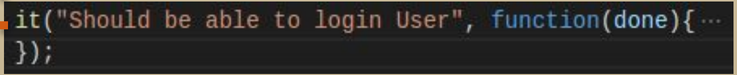
The main test firing event.
What are you expecting from that event.



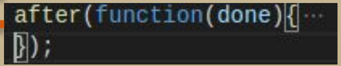
```
it("Should not be able to login User Missing email or Pawword", function(done){ ···
});
```



```
describe("Login User", function(){ ···
}
```



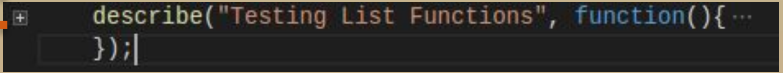
```
it("Should be able to login User", function(done){ ···
});
```



```
after(function(done){ ···
});
```

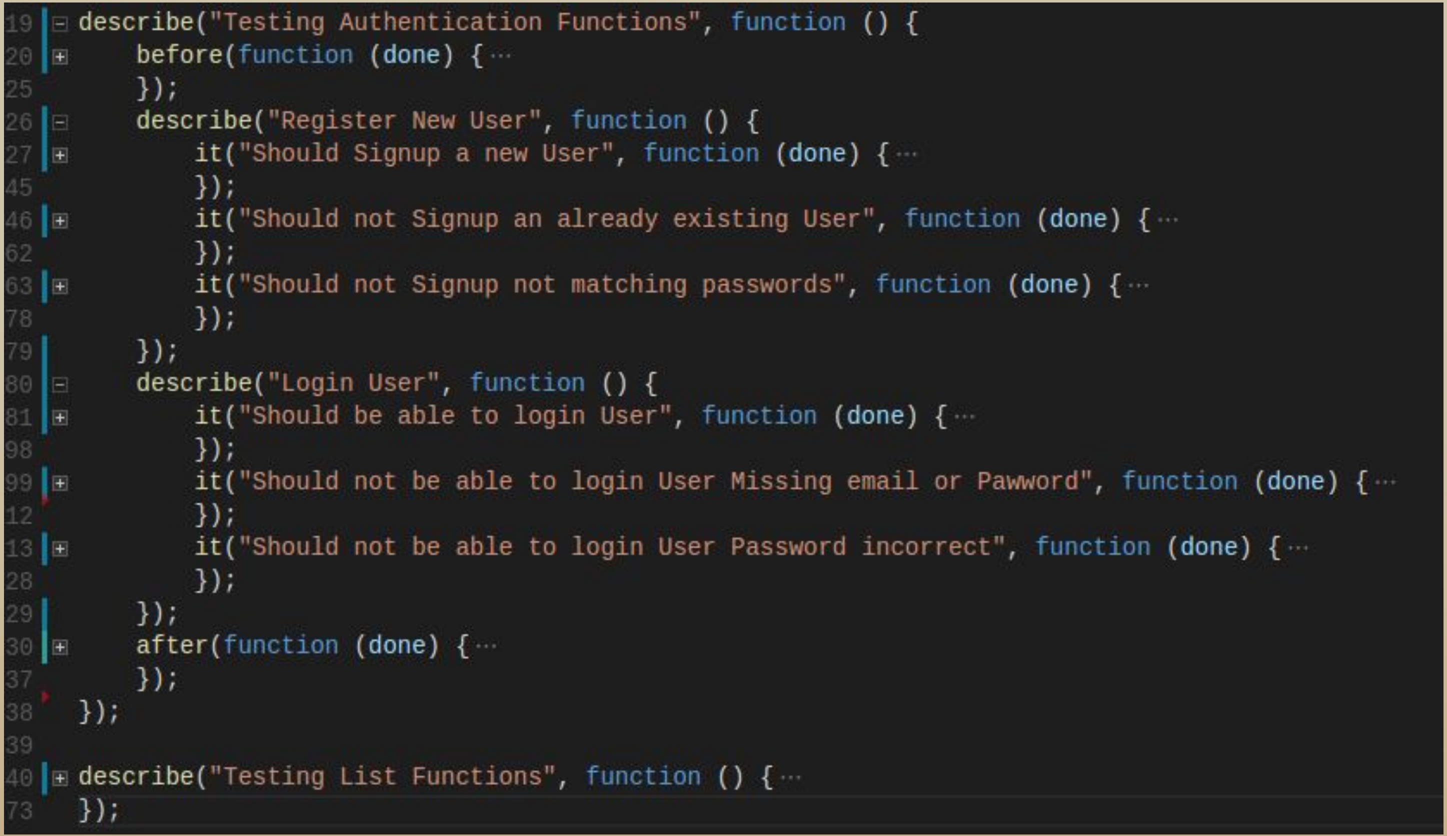
After block

This will be the last thing. You have to terminate the connection with your db, and delete all dummy data you have inserted while testing



```
describe("Testing List Functions", function(){ ···
});
```

More on Mocha Divisions



```
describe("Testing Authentication Functions", function () {
    before(function (done) {⋯
    });
    describe("Register New User", function () {
        it("Should Signup a new User", function (done) {⋯
        });
        it("Should not Signup an already existing User", function (done) {⋯
        });
        it("Should not Signup not matching passwords", function (done) {⋯
        });
    });
    describe("Login User", function () {
        it("Should be able to login User", function (done) {⋯
        });
        it("Should not be able to login User Missing email or Pawword", function (done) {⋯
        });
        it("Should not be able to login User Password incorrect", function (done) {⋯
        });
    });
    after(function (done) {⋯
    });
});

describe("Testing List Functions", function () {⋯
});
```

Thank you

Author: Amr Ahmed Ali
You can find me at:
Github: https://github.com/MoAgamia/SE-Boot-Camp