COURSEWARE

Mocha

Contents

Overview

JS testing framework specifically designed for use with asynchronous functions.

Installation

```
npm i mocha

npm i -g mocha
```

It is useful to install `mocha` both locally and globally as this makes it possible to use the `mocha` runner at command line.

Running mocha

To use Mocha - simply add `mocha` to a **test** script in the **package.json** file.

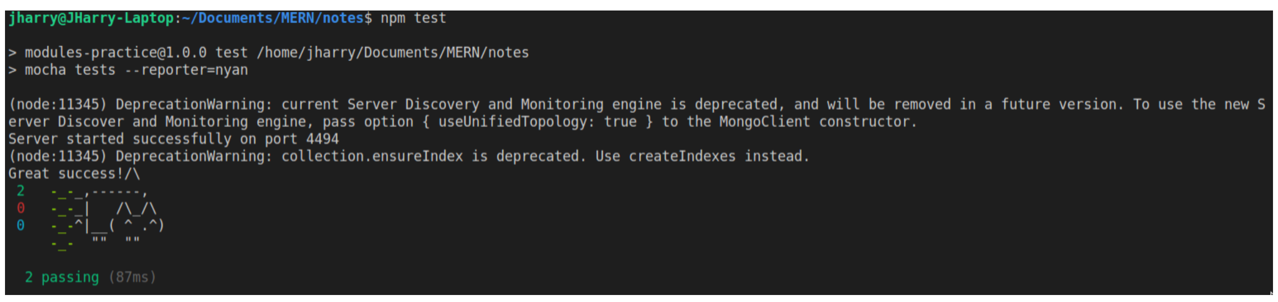
```
  "scripts": {
    "start": "node index.js",
    "test": "mocha"
  }
```

Now, when `npm test` is run Mocha will scan through the current folder and run any test scripts it finds.

If your tests are in a sub-folder (*/tests* for example) then simply include the relative path to that folder in the test script, like so:

```
  "scripts": {
    "start": "node index.js",
    "test": "mocha tests"
  }
```

Mocha will output the test results to the console in this format:



```
jharry@JHarry-Laptop:~/Documents/MERN/notes$ npm test

> modules-practice@1.0.0 test /home/jharry/Documents/MERN/notes
> mocha tests --reporter=nyan

(node:11345) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new S
erver Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
Server started successfully on port 4494
(node:11345) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
Great success!/\
 2   -_-_,------,
 0   -_-_|   /\_/\
 0   -_-^|__( ^ .^)
     -_-  ""  ""

  2 passing (87ms)
```

describe (it)

Mocha tests are designated using `describe()` or `it()`, these functions take in two arguments:

1. A description - as a `string`.
2. The test itself - as a `function`.

For example:

```
describe('Testing a feature', function() {

  it('Should do one thing', function() {
    //test it does one thing
  });

  it('Should do another thing', function() {
    //test it does another thing
  });

});
```

Passing arrow functions to Mocha is discouraged. Lambdas lexically bind `this` and cannot access the Mocha context.

If you do not need to use Mocha's context, lambdas should work.

Hooks

Mocha provides the hooks `before()`, `after()` and `beforeEach()` and `afterEach()`. These should be used to set up preconditions and clean up after your tests.

```
describe('Hooks', function(){
    before(function(){
        // runs once before the first test in this block
    });

    after(function(){
        // runs once after the last test in this block
    });

    beforeEach(function(){
        // runs before each test in this block
    });

    afterEach(function(){
        // runs after each test in this block
    });

    // test cases
});
```

Exclusive tests

Exclusivity allows you to run *only* the specified suite or test-case by appending `.only()` to the function.

```
describe('Exlusive tests', function(){
    descibe.only('This one', function() {
        // some implementation
    });
});
```

All nested suites will still be executed.

Here's an example of executing an individual test case:

```
describe('Exclusive tests', function(){
    describe('Collection', function(){

        it.only('Should run', function(){
            // ...
        });

        it('Will not run', function(){
            // ...
        });
    });
});
```

Skipping tests

This feature is the inverse of `.only()`.
By appending `.skip()` we tell Mocha to ignore tests. Anything skipped will be marked as pending and reported as such.

```
describe('A suite', function(){
    describe('A more segregation', function(){

        it.skip('Should skip',function(){
            // ...
        });

        it.('Should run',function(){
            // ...
        });
    });
})
```

Tutorial

There is no tutorial for this module.

Exercises

There is no exercise for this module.