MOCHA + CHAI

Óscar Moreira Estévez - alu0101209067
Adal Díaz Fariña - alu0101112251

INDEX

01

INTRODUCTION TO TESTING

- Definition
- Importance
- Testing Pyramid

02

MOCHA AND CHAI

- Definitions
- Directory structure
- Test structure

03

VSCODE EXTENSION

INDEX

INTRODUCTION TO TESTING

01

DEFINITION

Software Testing is the conduct of tests on it, in order to obtain information about its quality. It is one more activity in the quality control process.

QUESTION

Why is testing important?

IMPORTANCE OF THE TESTS

Refactor on insurance

If there are bugs, we catch them

Documentary

Empirical evidence that it works

Continuous deployment

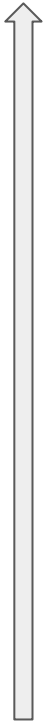
IMPORTANCE TDD

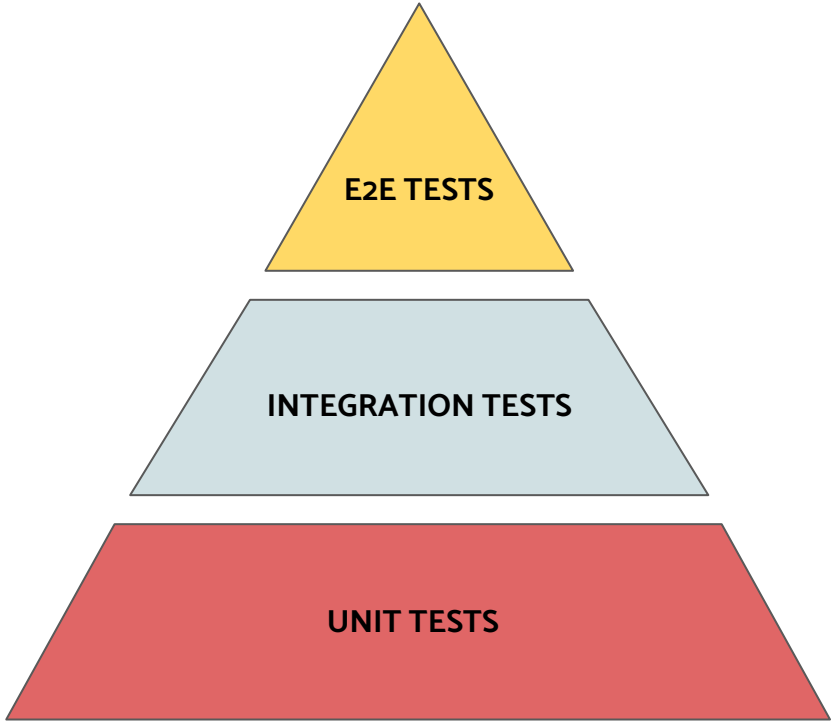
Faster development

We focus on the target

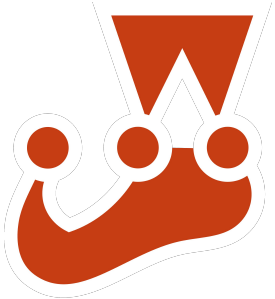
The code is better

TESTING PYRAMID





TESTING FRAMEWORKS FOR JAVASCRIPT



MOCHA AND CHAI

02

WHAT IS MOCHA?

Mocha.org

- Javascript Testing Framework
- Node.js
- Browsers

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

AND CHAI?

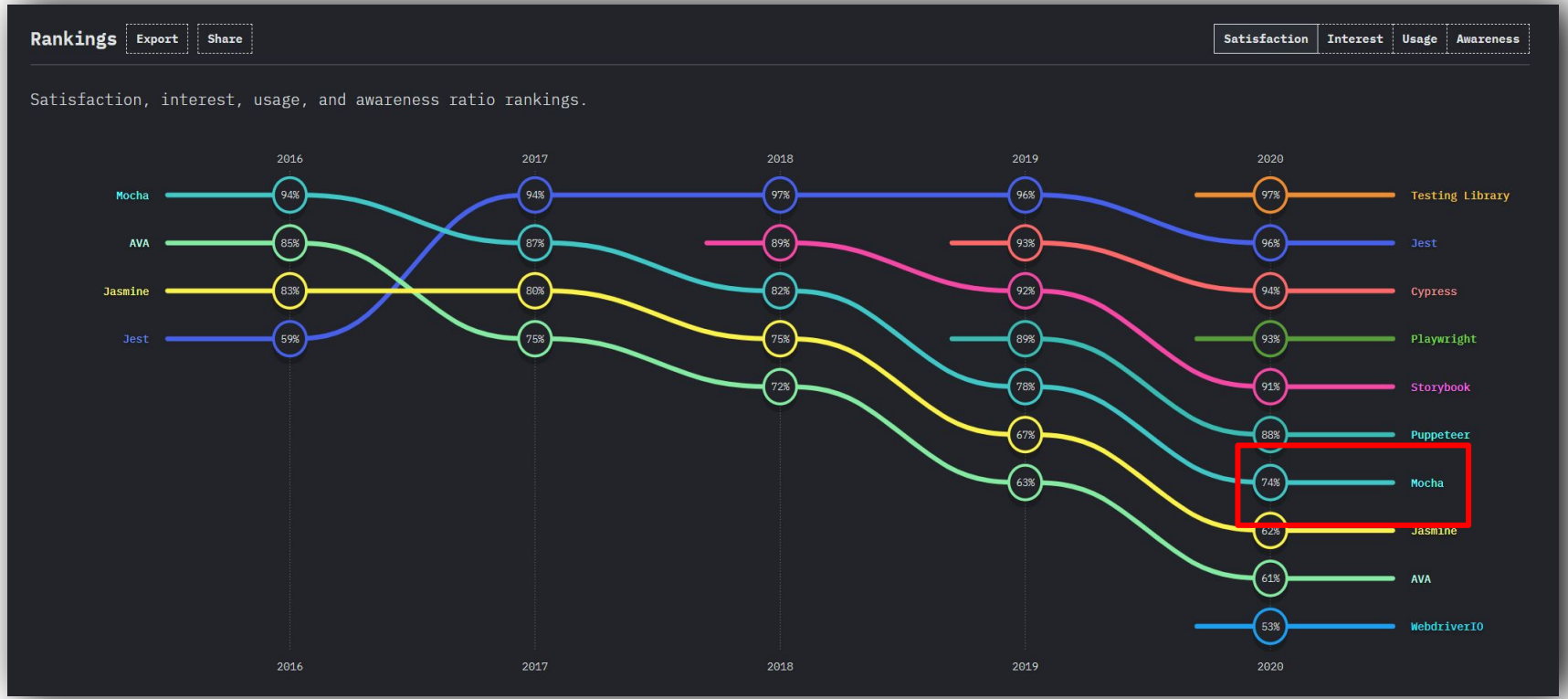
Chai.com

- BDD / TDD assertion library
- Node.js
- Browsers
- Can be paired with any javascript testing framework

It is an assertion library, which can be paired with any Javascript testing framework. Chai has several interfaces: assert, expect and should, which allow the developer to choose the style that is most readable and comfortable for him when developing his tests.

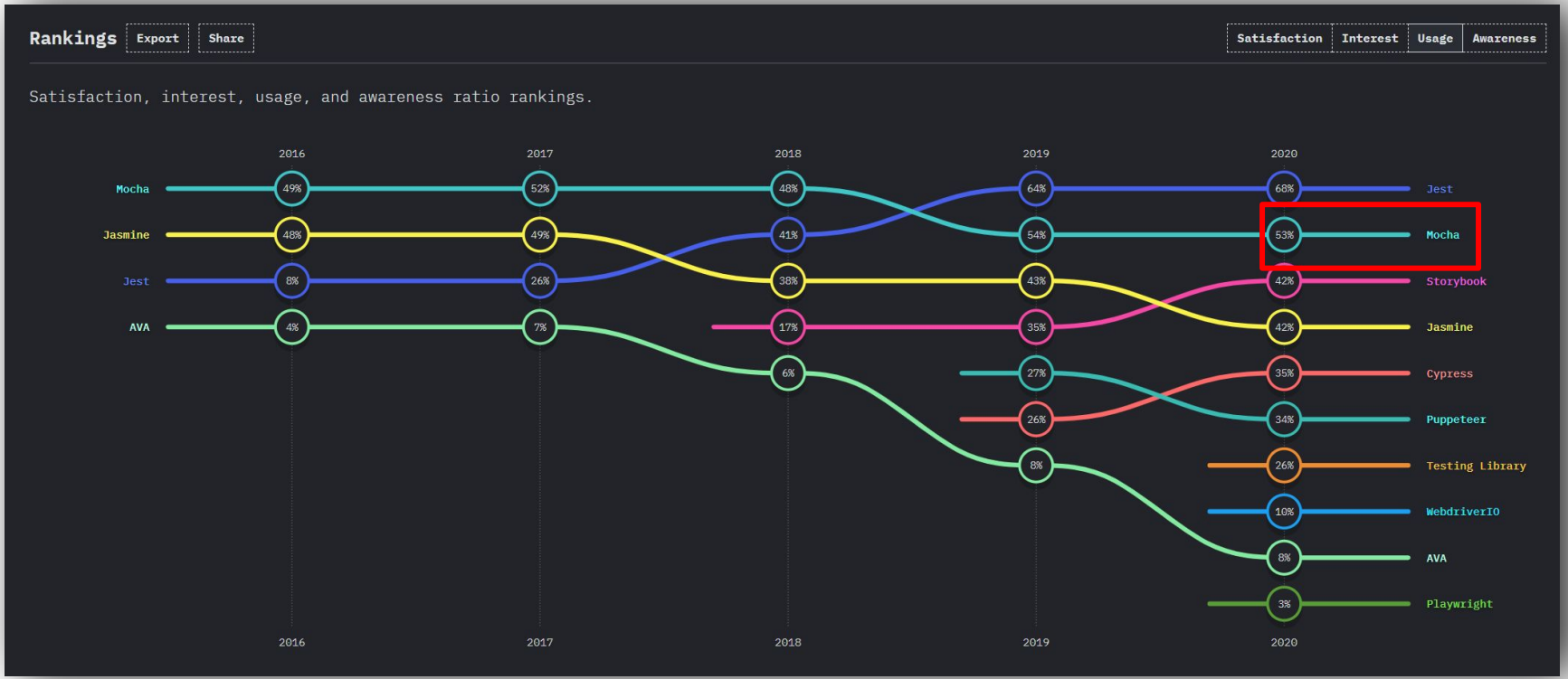
SATISFACTION RANKING

StateOfJS



USE RANKING

StateOfJS



INSTALL MOCHA AND CHAI

```
$ npm install -D mocha
```

```
$ npm install -D chai
```

```
$ npm install -D mocha chai
```

MOCHA PACKAGE.JSON

Only test directory

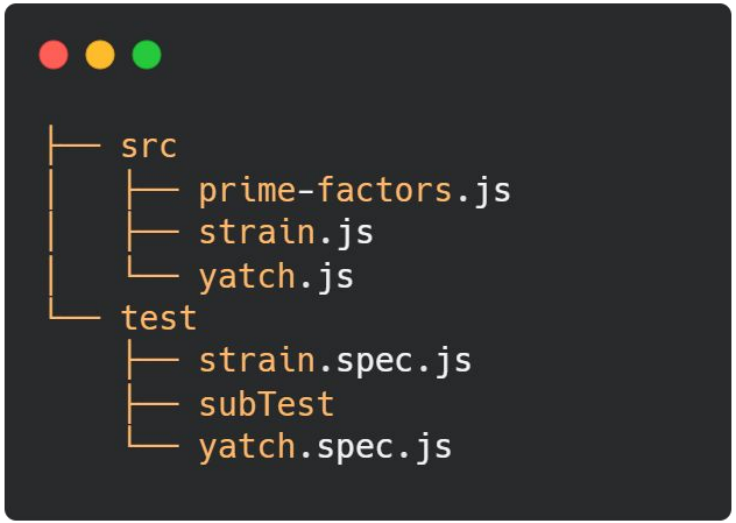
```
"scripts": {
  "test": "mocha"
},
```

Include subdirectories

```
"scripts": {
  "test": "mocha --recursive"
},
```

DIRECTORY STRUCTURE

- **src:** The src directory contains the source codes for our project.
- **test:** The test directory contains the test code about our project



TEST STRUCTURE

HEADER

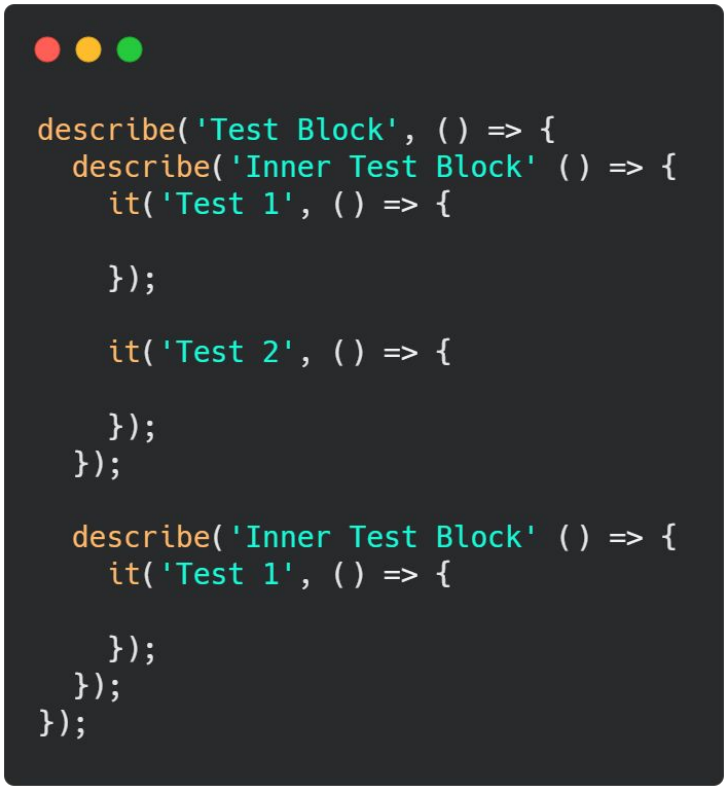
An example header can be the one we see below, where we load the module of the chai library that we are going to use and our program

```
const {expect, assert} = require('chai');
const should = require('chai').should();
const chaiHttp = require('chai-http');
const score = require('../src/nth-prime.js');
```

```
import {expect} from 'chai';
import {sum} from '../src/sum.js';
```

DESCRIBE

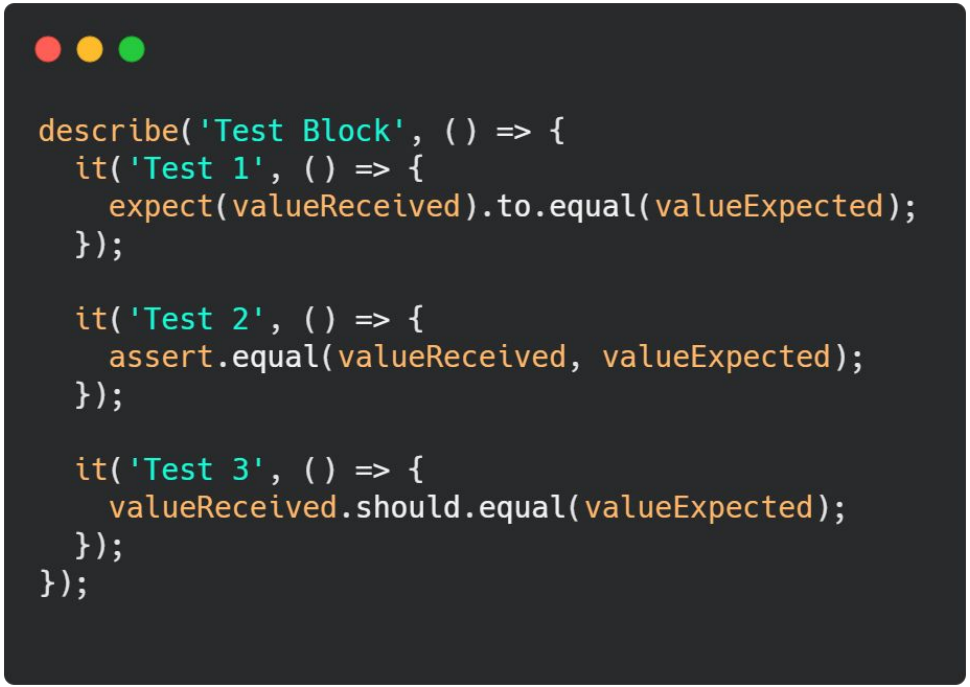
Describe is a hook that allows us to create related test blocks. We can go nesting describes if we consider it necessary for the structure of our tests.



```
describe('Test Block', () => {
  describe('Inner Test Block' () => {
    it('Test 1', () => {

    });

    it('Test 2', () => {

    });
  });

  describe('Inner Test Block' () => {
    it('Test 1', () => {

    });
  });
});
```

IT (TEST)

It is a test that we will perform. We can add as many tests as we want inside a description.

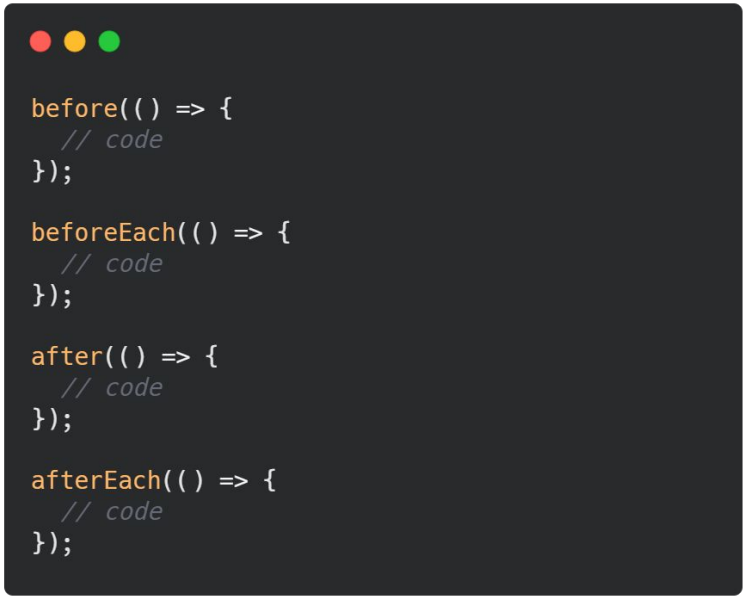


```
describe('Test Block', () => {
  it('Test 1', () => {
    expect(valueReceived).to.equal(valueExpected);
  });

  it('Test 2', () => {
    assert.equal(valueReceived, valueExpected);
  });

  it('Test 3', () => {
    valueReceived.should.equal(valueExpected);
  });
});
```

HOOKS

This would be the simplest structure to test in Mocha, but Mocha allows us to make use of many more hooks

- **Before():** Before is a hook that runs before the first it() or describe()
- **BeforeEach():** BeforeEach is a hook that run before each it() or describe()
- **After():** Afert is a hook that runs after all it() or describe()
- **AfterEach():** Afert is a hook that runs after each it() or describe()



```
before(() => {
  // code
});

beforeEach(() => {
  // code
});

after(() => {
  // code
});

afterEach(() => {
  // code
});
```

SELECT TEST TO RUN

This would be the simplest structure to test in Mocha, but Mocha allows us to make use of many more hooks

Function only:
The function only allow us to execute only the test or block, that we want. For example:

```
describe.only("Test Block 1", function() {
  it("Test 1", function() {});

  it("Test 2", function() {});
});

describe("Test Block 2", function() {
  it("Test 3", function() {});
});
```

```
describe("Test Block", function() {
  it.only("Test 1", function() {});

  it("Test 2", function() {});

  it("Test 3", function() {});
});
```

Function skip:

The function skip allow us to pass the tests or block, that we want

```
describe("Test Block", function() {
  it.skip("Test 1", function() {});

  it("Test 2", function() {});

  it("Test 3", function() {});
});
```

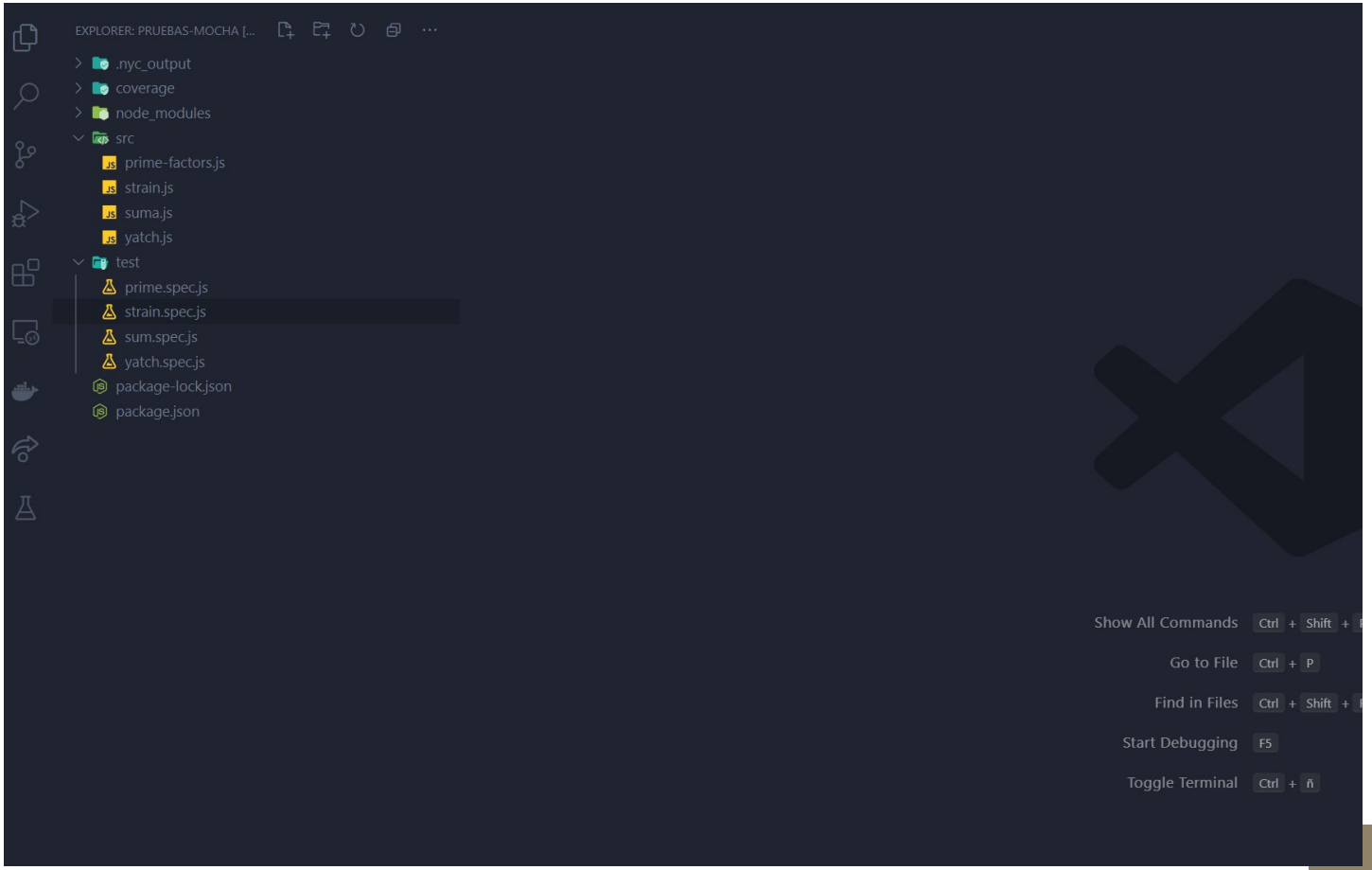
```
describe.skip("Test Block 1", function() {
  it("Test", function() {});
});

describe("Test Block 2", function() {
  it("Test", function() {});
});
```


VSCODE
EXTENSION

MOCHA SIDEBAR

Mocha sidebar

MOCHA SIDE BAR



TESTS

03

CHAI MODULES

```
const should = require('chai').should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

```
const {expect} = require('chai');

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

```
const { asserts } = require('chai');

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3)
```

EXAMPLES

EXERCISM TEST

Link to examples

JEST

```
describe('returns prime factors for the given input number', () => {
  test('no factors', () =>
    expect(primeFactors(1)).toEqual([]));

  test('prime number', () =>
    expect(primeFactors(2)).toEqual([2]));

  test('another prime number', () =>
    expect(primeFactors(3)).toEqual([3]));

  test('square of a prime', () =>
    expect(primeFactors(9)).toEqual([3, 3]));

  test('product of first prime', () =>
    expect(primeFactors(4)).toEqual([2, 2]));

  test('cube of a prime', () =>
    expect(primeFactors(8)).toEqual([2, 2, 2]));

  test('product of second prime', () =>
    expect(primeFactors(27)).toEqual([3, 3, 3]));
 });
```

Mocha + Chai

```
const { expect } = require('chai');

describe('returns prime factors for the given input number', () => {
  it('no factors', () =>
    expect(primeFactors(1)).to.deep.equal([]));

  it('prime number', () =>
    expect(primeFactors(2)).to.deep.equal([2]));

  it('another prime number', () =>
    expect(primeFactors(3)).to.deep.equal([3]));

  it('square of a prime', () =>
    expect(primeFactors(9)).to.deep.equal([3, 3]));

  it('product of first prime', () =>
    expect(primeFactors(4)).to.deep.equal([2, 2]));

  it('cube of a prime', () =>
    expect(primeFactors(8)).to.deep.equal([2, 2, 2]));

  it('product of second prime', () =>
    expect(primeFactors(27)).to.deep.equal([3, 3, 3]));
 });
```

EQUALS

Assert

```
assert.equal(3, '3');
```

```
assert.strictEqual(3, '3');
```

```
assert.strictEqual(3, 3);
```

Expect

```
expect(20).to.equal('20');
```

```
expect(20).to.equal(20);
```

DEEP EQUALS

deep-equal

What is Deep-Eql?

Deep Eql is a module which you can use to determine if two objects are "deeply" equal - that is, rather than having referential equality (`a === b`), this module checks an object's keys recursively, until it finds primitives to check for referential equality. For more on equality in JavaScript, read the comparison operators article on mdn.

```
> { a: 1 } === { a: 1 }
false
> { a: 1 } == { a: 1 }
false
> const deepEql = require('deep-eql');
> deepEql({ a: 1 }, { a: 1 }) === true
true
```

DEEP EQUALS

Asserts

```
assert.equal([3, 4 ,5], [3, 4 ,5]);
```

```
assert.deepEqual([3, 4 ,5], [3, 4 ,5]);
```

Expects

```
expect([1, 2]).to.equal([2, 1])
```

```
expect([1, 2]).to.deep.equal([2, 1])
```

```
expect([1, 2]).to.eql([2, 1])
```

INCLUDES

Asserts

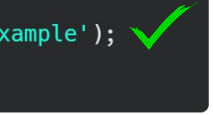
```
assert.include([1,2,3], 2);
```

```
const obj1 = {a: 1},
      obj2 = {b: 2};

assert.include({foo: obj1, bar: obj2}, {foo: obj1});
```

```
const obj1 = {a: 1},
      obj2 = {b: 2};

assert.include({foo: obj1, bar: obj2}, {foo: {a: 1}});
```

Expects

```
expect('This is an example').to.include('example');
```



```
expect([1, 2, 3]).to.include(2);
```

```
expect({x: {a: 1}}).to.include({x: {a: 1}});
```

DEEP INCLUDE

Assert

```
const obj1 = {a: 1},
      obj2 = {b: 2};

assert.deepInclude({foo: obj1, bar: obj2}, {foo: {a: 1}});
```

Expect

```
expect([{a: 1}]).to.deep.include({a: 1});

expect({x: {a: 1}}).to.deep.include({x: {a: 1}});
```

TYPEOF

Returns the type of the ones you pass by parameter

Assert

```
it('assert typeof', () => {
  assert.typeOf({ tea: 'chai' },
                'object',
                'we have an object');
});
```

Expect

```
it('expect typeof', () => {
  expect({a: 1}).to.be.an('object');
});
```

```
it('expect typeof', () => {
  expect('foo').to.be.a('string');
});
```

LENGTH OF

Returns the size of the ones you pass by parameter

Asserts

```
it('Assert lengthOf', () => {
  assert.lengthOf([1,2,3], 3,
         'array has length of 3');
});
```

Expects

```
it('Expect lengthOf', () => {
  expect([1, 2, 3])
    .to.have.lengthOf(3);;
});
```

THROW

Check that the error is returned

Assert

```
it('Assert Throws', () => {
  assert.throws(fn, 'Error message');
});
```

Expect

```
it('Expect Throw', () => {
  expect(func).to.throw('Error message');
});
```

BIND THE CONTEXT

```
expect(function() { func(); }).to.throw('Error message');
expect(() => func()).to.throw('Error message');
```

```
expect(cat.meow.bind(cat)).to.throw();
```

AND MORE...

Chai has a lot of expect and assert that make the tests easier

- isUndefined
- isDefined
- isFunction
- isNotFunction
- isObject
- isNotObject

- isArray
- isNotArray
- isString
- isNotString
- isNumber
- isNotNumber

IN BROWSER

Add libraries and styles

```
<link href="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.css" rel="stylesheet">
<script src="http://chaijs.com/chai.js"></script>
<script src="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.js"></script>
```

Basic configuration

```
<script>
  const expect = chai.expect;
  mocha.setup('bdd');
</script>
```

Load program and test

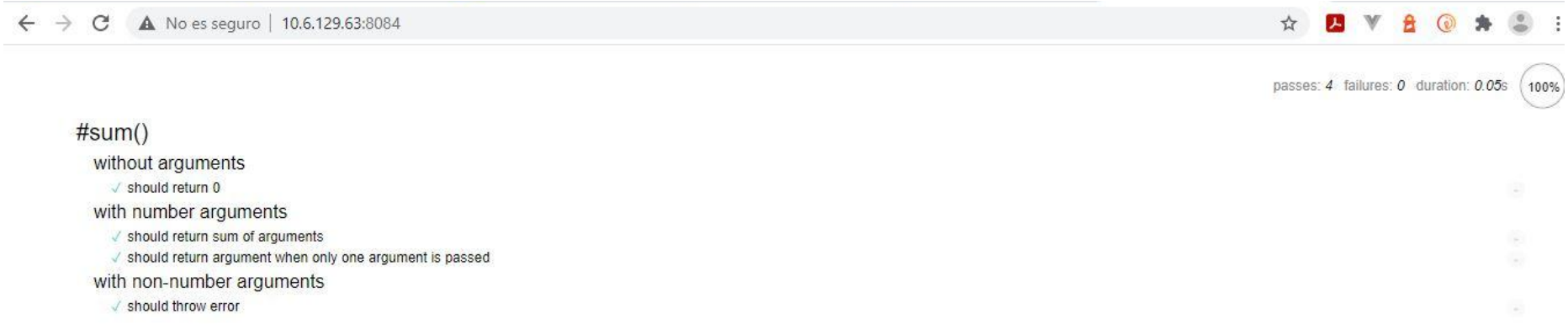
```
<script src="js/sum.js"></script>
<script src="js/sum.spec.js"></script>
```

Run the test with Mocha

```
<div id="mocha"></div>
<script>
  mocha.run();
</script>
```

TEST IN BROWSER

Test



CHAI-HTTP

http response

```
const { expect } = require('chai');
const chai = require('chai');
const chaiHttp = require('chai-http');

chai.use(chaiHttp);
const url= '10.6.129.63:8081';

describe('Working app', () => {
  it('We expect to have a status 200 so the web is deployed', (done) => {
    chai.request(url)
    .get('/')
    .end( function(err,res){
      console.log(res.body)
      expect(res).to.have.status(200);
      done();
    });
  });
});
```

CHAI-HTTP TEST EXECUTE

```
$ npx mocha test/app_spec.js


  Working app
{}
    ✓ We expect to have a status 200 so the web is deployed


  1 passing (36ms)
```

TESTING A CLASS

Class example

```
class Rectangle {
  constructor (height, width) {
    this.height = height;
    this.width = width;
  }

  get area() {
    return this.calcArea();
  }

  calcArea () {
    return this.height * this.width;
  }
}
```

```
const {expect} = require('chai');
const {Rectangle} = require('../src/rectangle');

describe('Rectangle tests', () => {
  const square = new Rectangle(10, 10);

  it('we expected an object', () => {
    expect(square).to.be.a('object');
  });

  it('Square is an object of Rectangle class', () => {
    expect(square instanceof Rectangle).to.equal(true);
  });

  it('Area of a square', () => {
    expect(square.area).to.equal(100)
  });

  it('Area of a rectangle', () => {
    const rectangle = new Rectangle(10, 20);
    expect(rectangle.area).to.equal(200)
  });
});
```

COVERAGE

COVERING WITH MOCHA

module nyc

For this, we need a module npm called **nyc**

- Install nyc

```
$ npm i -D nyc
```

- Package.json

```
"scripts": {
  "test": "mocha",
  "coverage": "nyc npm test"
}
```

COVERING WITH MOCHA


```
$ npm run coverage test/prime.spec.js

  returns prime factors for the given input number
    ✓ no factors
    ✓ prime number
    ✓ another prime number
    ✓ square of a prime
    ✓ product of first prime
    ✓ cube of a prime
    ✓ product of second prime
    ✓ product of third prime
    ✓ product of first prime and second prime
    ✓ product of primes and non-primes
    ✓ product of primes
    ✓ factors include a large prime


  12 passing (23ms)

------------------|---------|----------|---------|---------|-------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|-------------------
All files         |     100 |      100 |     100 |     100 |
 prime-factors.js |     100 |      100 |     100 |     100 |
------------------|---------|----------|---------|---------|-------------------
```

CONCLUSION

04

BAD PRACTICE

1. Write more than one expect, assert or should within an it

```
it('test with more than one', () => {
  expect(valueReceive).to.equal(valueExpected);
  expect(valueReceive).to.equal(valueExpected);
});
```

2. Use negative assertion

```
it('test with negative assertion', () => {
  expect(2).to.equal(2); // Recommended
  expect(2).to.not.equal(1); // Not recommended
});
```

GOOD PRACTICE

Files with .spec

- All tests have extension `.spec.js`.

Folder test/

By default, `mocha` looks for the glob `"./test/*.js"`, so you may want to put your tests in `test/` folder.

Descriptive names

Descriptive tests

Group by similar functionalities

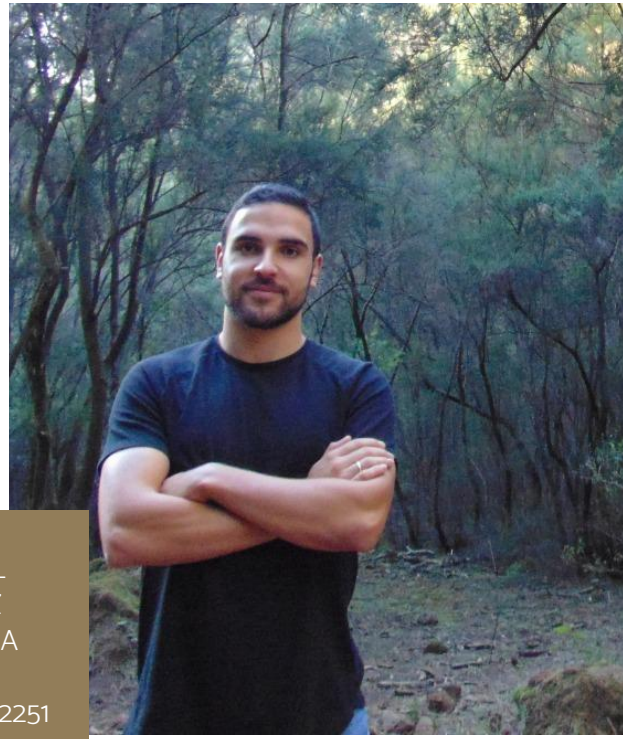
BIBLIOGRAPHY

- Mocha Documentation (link)
- Chai Documentation (link)
- nyc covering (link)
- Testing Javascript with Mocha and Chai (link)
- State Of JS (link)
- Mocha SideBar
- Mocha and Chai repository

OUR TEAM

ÓSCAR
MOREIRA
ESTÉVEZ

alu0101209067



ADAL
DÍAZ
FARIÑA

alu0101112251

THANKS

Questions?

Repository url :
Mocha and Chai