Tema 10

TDD para Node.js con Mocha

Desarrollo TDD

- El desarrollo TDD se basa en la codificación primero de las pruebas unitarias de nuestro proyecto y a continuación el desarrollo del código que cumpla con esas pruebas que hemos definido.

- En un primer momento, debemos verificar que las pruebas escritas fallan para que después, al pasarlas con el nuevo código desarrollado podamos lograr que su resultado sea satisfactorio.

- El propósito de este tipo de desarrollos es el de lograr un código limpio que funcione.

- Es una manera muy rápida de programar ya que los requisitos de nuestro proyecto directamente se pueden transformar en pruebas a partir de las cuales desarrollar nuestro código.

Mocha

- Mocha se trata de uno de los más potentes y conocidos frameworks para realizar pruebas sobre proyectos creados con Node.js.

- Para instalarlo, tendremos que ejecutar:

npm install -g mocha

- Aparte, usaremos la librería **Chai** para poder hacer las comprobaciones sobre nuestras pruebas.

npm install chai

Mocha

- Mocha es el módulo que vamos a emplear para lanzar las diferentes pruebas que vayamos creando para nuestro proyecto.

- Para poder especificar nuestros grupos de pruebas utilizaremos la cláusula **describe** e **it** para definir cada una de las pruebas que vamos a aplicar.

- Por ejemplo, si tenemos una clase llamada *Persona* y dentro de esa clase queremos probar su método *mostrarPersona*, podríamos plantear el siguiente bloque de pruebas:

```
describe("Persona", function() {
    describe(".mostrarPersona()", function() {
        it("debe detectar si el nombre no es nulo", function(){
            //El código para el test va aquí
        });
    });
});
```

Mocha

- El formato en el cual se definen los casos de prueba a través del método describe es totalmente opcional, pero si os fijáis en el caso expuesto anteriormente, la lectura de esa prueba es muy sencilla y es fácil situarla dentro del contexto en el que se encuentra.

- Sabemos con muy poco esfuerzo, qué clase y qué método estamos probando.

- Por lo tanto, se recomienda siempre seguir una nomenclatura clara a la hora de definir nuestros casos de prueba que nos ayuden a un posterior seguimiento.

Mocha

- Para no repetir código en cada una de nuestras pruebas o poder definir ciertas variables que vamos a utilizar a nivel global, podemos usar los métodos **before** o **beforeEach,** los cuales se ejecutarán antes de todas las pruebas definidas o antes de cada una de las pruebas, respectivamente.

```
var assert = require('assert');
before(function(){
    //Código a ejecutar antes de las pruebas
})
describe('Conjunto de pruebas', function(){
    beforeEach(function(){
        //Código a ejecutar antes de cada prueba
    })
    it('prueba 1', function(){
        //Código prueba 1
    });
    it('prueba 2', function(){
        //Código prueba 2
    })
})
```

Mocha - Chai

- Podemos utilizar el framework Chai, el cual nos permite trabajar con diferentes métodos más elegantes a la hora de definir nuestras pruebas.

- Algunos de los métodos que podemos usar son los siguientes:

- **assert (expressions, message).** Lanza una excepción si *expressions* es falso.

- **assert.fail(actual, expected, [message], [operator]).** lanza un error con valores de actual, expected y operator.

- **assert.ok (object, [message]).** lanza un error cuando el objeto no es igual a true.

- **assert.notOk (object, [message]).** lanza un error cuando el objeto es false

Mocha - Chai

- **assert.equal (actual, expected, [message]).** lanza un error cuando actual no es igual a expected

- **assert.notEqual(actual, expected, [message]).** lanza un error cuando actual es igual a expected

Expect.js

- Expect.js se trata de un módulo que nos aporta una nueva sintaxis para poder especificar nuestros casos de prueba.

- Nos permite pasarle como parámetro el objeto que queremos evaluar y compararlo con el resultado esperado.

expect(3-1).equal(2);

- En este caso, procedemos a evaluar la resta entre los números 3 y 1 y lo comparamos con su resultado 2.

- Es una prueba muy sencilla para poder ver el concepto del método ya que a la hora de plantear nuestras pruebas va a ser poco común encontrar comprobaciones como esta.

Expect

- Expect nos ofrece una serie de operadores que nos permiten interactuar con el resultado esperado dentro de nuestra prueba.
- Las diferentes funciones disponibles son las siguientes:
- **to, be, been, is, that, and, have, with, at, of, same, a, an**
- El primer ejemplo que hemos puesto se podría modificar de la siguiente manera para obtener la comprobación contraria:

expect(3-1).to.not.equal(5)