

## Ejercicios Tema 9

### Ejercicio 1

Vamos a definir una serie de pruebas sobre el proyecto creado en el tema anterior.

El primer paso será instalar los módulos que vamos a utilizar para la creación y la ejecución de las diferentes pruebas:

```
npm install -g mocha
```

```
npm install chai
```

```
npm install assert
```

Las pruebas las generaremos dentro del script **/test/test.js**

Lo primero que vamos a definir son las pruebas para posteriormente realizar el código que cumpla esas pruebas. Las premisas con las que partimos son las siguientes:

- Vamos a desarrollar una clase Operaciones que tenga una función que nos permita sumar dos números
- Ese mismo script dispondrá de otra función que nos permita obtener el siguiente número primo de otro que le pasemos por parámetro.
- Crearemos otro script que nos permita realizar comprobaciones sobre direcciones de email, como por ejemplo, comprobar si el mail con el que trabajamos contiene el caracter @ y el punto.

En nuestro script **test.js** lo primero que tenemos que definir son los require que vamos a utilizar durante todo el script. Necesitamos el require de **assert** para poder evaluar las comprobaciones que vamos a hacer y los dos controladores que crearemos a posteriori con las funciones (Operaciones y MailFunctions)

```
var assert = require('assert');
var Operaciones = require('../controllers/Operaciones');
var MailFunctions = require('../controllers/MailFunctions');
```

El siguiente paso será crear los conjuntos de pruebas, los cuales dividiremos en función del script que estemos probando a través del método **describe**.

```
describe('Conjunto de pruebas #1: Operaciones', function() {

});
```

```
describe('Conjunto de pruebas #2: MailFunctions', function(){
});
```

Vamos a empezar a definir las diferentes pruebas dentro de cada uno de los conjuntos. Comenzamos con las pruebas para la función de sumar.

```
describe('Conjunto de pruebas #1: Operaciones', function(){
  it('suma(n,m)', function(){
    assert.equal(22, Operaciones.suma(11,11));
    assert.equal(22, Operaciones.suma(20,2));
    assert.equal(22, Operaciones.suma(18,4));
  });
});
```

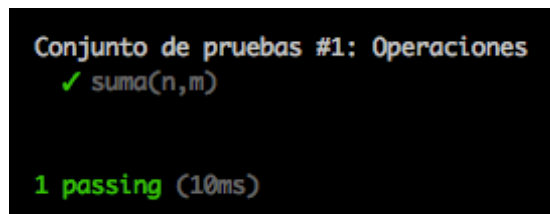
Se define una prueba dentro de la cual, especificamos varias pruebas sobre la función suma dentro del script Operaciones. De esta manera queda claro que, evaluando las diferentes posibilidades que tenemos para alcanzar el resultado, nos aseguramos que el método funciona correctamente.

Vamos a definir el método suma con la premisa que hemos creado a la hora de definir la prueba. En el script **controllers/Operaciones.js** podemos desarrollar algo parecido a esto:

```
function suma(n, m){
  return n+m;
}

module.exports = {
  suma: suma
}
```

Para ejecutar las pruebas, simplemente tenemos que ejecutar el comando **mocha** desde nuestro terminal, obteniendo un resultado como el siguiente:



```
Conjunto de pruebas #1: Operaciones
✓ suma(n,m)

1 passing (10ms)
```

El siguiente paso es definir la prueba para evaluar la función que nos obtiene el siguiente número primo, para ello, crearemos una nueva prueba dentro del grupo de pruebas que hemos creado para el script de Operaciones:

```
describe('Conjunto de pruebas #1: Operaciones', function(){
```

```
it('suma(n,m)', function(){
  assert.equal(22, Operaciones.suma(11,11));
  assert.equal(22, Operaciones.suma(20,2));
  assert.equal(22, Operaciones.suma(18,4));
});

it('nextPrimo(n)', function(){
  assert.equal(11, Operaciones.nextPrimo(7));
});
});
```

Y como hemos hecho anteriormente, a partir de la prueba que hemos generado, creamos el método **nextPrimo**. Podría ser algo así:

```
function nextPrimo(n){
  var smaller;
  n = Math.floor(n);

  if (n >= 2) {
    smaller = 1;
    while (smaller * smaller <= n) {
      n++;
      smaller = 2;
      while ((n % smaller > 0) && (smaller * smaller <= n)) {
        smaller++;
      }
    }
    return n;
  } else {
    return 2;
  }
}

function suma(n, m){
  return n+m;
}

module.exports = {
  nextPrimo: nextPrimo,
  suma: suma
}
```

Si, como antes, ejecutamos el comando **mocha** desde nuestro terminal obtenemos el siguiente resultado

Conjunto de pruebas #1: Operaciones

✓ suma(n,m)  
✓ nextPrimo(n)

2 passing (35ms)

## Ejercicio 2

General las pruebas y el código que compruebe si una dirección de mail\* que pasamos es válida. (puedes usar la función **ok** de **assert**)

\*Puedes utilizar la siguiente expresión regular para validar que el correo es válido según la RFC 5322:

```
var re = /^((\[^\(\)[\]\\\.,;:\s@" ]+(\.[^\(\)[\]\\\.,;:\s@" ]+)*|(".+"))@((\[ [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\]|(( [a-zA-Z\ -0-9 ]+\.)+[a-zA-Z]{2,}))$)/;
```

## Ejercicio 3

Hemos terminado hace poco una API, y no podemos dormir tranquilos sin hacerle unas pruebas de test. ¿Pero como hacer un test no a los métodos, si no al servicio RESTful?

Vuelve al proyecto ToDo y asegurarte de que existe nuestra carpeta /test

Instala los siguientes paquetes: mocha, should y supertest

```
npm install --save-dev mocha
npm install --save-dev should
npm install --save-dev supertest
```

Crea un archivo test.js dentro de /test:

```

var supertest = require("supertest");
var should = require("should");

// This agent refers to PORT where program is running.

var server = supertest.agent("http://localhost:3000");

// UNIT test begin

describe("Ejemplo de test", function() {

  it("Deberia de devolverme un listado", function(done) {

    // llamada a la api
    server
    .get("/api")
    .expect("Content-type", /json/)
    .expect(200) // La respuesta HTTP
    .end(function(err, res) {
      // El status HTTP debería ser (should) 200
      res.status.should.equal(200);
      done();
    });
  });

  it("me tiene que dar 404", function(done) {
    server
    // Esta url no existe. Me merezco un 404
    .get("/tareass")
    .expect(404)
    .end(function(err, res) {
      res.status.should.equal(404);
      done();
    });
  });

  it("add todo", function(done) {

    //calling ADD api
    server
    .post('/api')
    .send({ "titulo": "Test Tdd", "completa": 0 })
    .expect("Content-type", /json/)
    .expect(201)
    .end(function(err, res) {
      // 201 Created: The request has been fulfilled and resulted in a
      new resource being created.
      res.status.should.equal(201);
      res.body.message.should.equal('Salvado');
    });
  });
});

```

```

        done();
    });
});

it("creamos ToDo y lo editamos", function(done) {
    var tarea = new ToDo({
        'titulo': 'mocha test put',
        'completa': 0
    });
    tarea.save(function(err, data) {

        console.log('#PUT /api/' + data.id);
        server
            .put('/api/' + data.id)
            .send({ "titulo": "mocha test put OK", "completa": 1 })
            .end(function(err, res) {
                res.status.should.equal(200);
                res.should.be.json;
                res.body.should.have.property('message');
                res.body.message.should.equal('Actualizada');
                done();
            });
    }); //save
});
});

```

Recuerda, este es solo un ejemplo y deberás editar lo necesario en tu código del proyecto ToDo para ajustar los "message" a lo que desarrollaste, y revisar que los códigos http que utilizaste son los correctos.

#### Ejercicio 4

A partir del ejemplo anterior, crea una prueba que cree una tarea, la edite, y la elimine.