Testing en javascript

Anxo Fole
afole@plainconcepts.com

Jasmine

Jasmine.js

- Standard de facto en el testeo de Angular
- Toda la documentación oficial lo utiliza
  - Todo ejemplo de código es acompañado con sus test
- Estilo funcional
- Incluye API para crear mocks de dependencias
  - Spy

http://jasmine.github.io/

Creación de tests

- describe
  - Crea ámbitos que agrupan test (specs)
  - Se pueden anidar
  - beforeEach y afterEach: permiten realizar acciones de inicialización limpieza antes o después de que se ejecuten los test de la suite
  - Prefijar con x para deshabilitar (xdescribe)

- it
  - Crea los test propiamente dichos
  - Prefijar con x para deshabilitar (xit)

Expectations

- Verifican una determinada condición

- Matchers (toBe)
  - Comparación entre el valor esperado y el actual

```
describe(“Test ", function() {
it("contains spec with an expectation", function() {
        expect(true).toBe(true);
});
 });
```

Expectations

```
describe(“Test ", function() {
it("contains spec with an expectation", function() {
        expect(true).toBe(true);
});
 });
```

```
describe("Included matchers:", function() {

it("The 'toBe' matcher compares with ===", function() {
        var a = 12;
        var b = a;

        expect(a).toBe(b);
        expect(a).not.toBe(null);
});
```

Expectations

```
describe("The 'toEqual' matcher", function() {

it("works for simple literals and variables", function() {
        var a = 12;
        expect(a).toEqual(12);

});

it("should work for objects", function() {

        var foo = { a: 12, b: 34 };
        var bar = { a: 12, b: 34 };
        expect(foo).toEqual(bar);
 });

});
```

Spy

- Doble funcionalidad
  - Permite crear mocks de funciones
  - Hacer track de las llamadas a estas y sus argumentos
- Matchers propios
  - .toHaveBeenCalled() devuelve true si el spy ha sido llamado.
  - .toHaveBeenCalledWith(arg) devuelve true si ha sido llamada con el mismo argumento.
  - .toHaveBeenCalledTimes(1) devuelve true si ha sido llamada con el mismo argumento.

Spy

```
describe("The 'toHaveBeenCalledWith' spy matcher", function() {

it('should track the arguments of its calls', function() {

        var c = new Circle();

        spyOn(c,'circumference’);
        c.circumference(2);
        expect(c.circumference).toHaveBeenCalledWith(2);

        });
});
```

Spy

- Mockear funciones
  - callThrough(): Trackea lasllamadas pero tiene el funcionamiento normal. Llama a la función real.

```
spyOn(foo, 'getBar').and.callThrough();
```

  - callFake(): Cada vez que llamemos a la función, utiliza la implementación que le pasamos.

```
spyOn(foo, "getBar").and.callFake(function(arguments, can, be, received) {
  return 1001;
});
```

  - returnValue(): Todas las llamadas que realicemos van a tener el mismo valor.

```
spyOn(foo, "getBar").and.returnValue(745);
```

Spy

- Tres formas diferentes de crearlos:
  - SpyOn: Sobre objetos existentes

```
spyOn(foo, "getBar").and.returnValue(745);
```

  - jasmine.createSy: crea un spy cuando no tenemos una función para hacer Spy.

  - Jasmine.createSpyObject: crea un objeto con varios spies.

Spy

```
describe("Multiple spies, when created manually", function() {
  var tape;

  beforeEach(function() {
    tape = jasmine.createSpyObj('tape', ['play', 'pause', 'stop', 'rewind']);

    tape.play();
    tape.pause();
    tape.rewind(0);
  });

  it("creates spies for each requested function", function() {
    expect(tape.play).toBeDefined();
    expect(tape.pause).toBeDefined();
    expect(tape.stop).toBeDefined();
    expect(tape.rewind).toBeDefined();
  });
```

Runners

Karma

- Usa node.js
- Levanta uno o varios navegadores simultáneos que ejecutan los tests.
- Nos proporciona cobertura de testing.
- No está tan integrado en VS

Iniciar Karma

- npm install -g karma-cli (instalar globalmente)
- ~~karma init~~ (crea un fichero de configuración de Karma, karma.conf.js)
- karma start

Test de Angular

Testing de Controladores y Componentes

- Angular incluye un servicio que es una factoría para crear controladores
    - Controladores : $controller
    - Controladores de componentes: $componentController
    - La usamos para personalizar el proceso de creación inyectando las dependencias que necesitemos

Controladores y componentes

- 3 beforeEach
  - Inicialización del módulo
  - Inyección de dependencias de angular con inject
  - Inicialización de los mocks necesarios

- Definición de una función constructora del controlador
  - Que será invocada en cada test
  - Se le inyectan las dependencias que necesite nuestro controlador

Component Controller Test

```
describe('ComponentController Test', function () {
    'use strict';
    //declaramos las variables
    var service,
    service2,
    mock,
    $componentController;

    function createComponentController() {
      //$componentController(componentName, locals, [bindings], [ident]);
      //locals: Injection locals for Controller.
          return $componentController('componentName',
          {
              service: service,
              service2: service2,
              mock: mock
          });
  }

    //1 cargar la app
    beforeEach(function () {
        module('moduleName');
    });
```

```
    //2 Inyeccion de dependencias
    beforeEach(inject(function (_$componentController_, _service_, _service2_) {
        todoService = _todoService_;
        $componentController = _$componentController_;
        service = _service_;
        service2 = _service2_;
    }));

    //3 MOCKS
    beforeEach(function () {
      mock = {id: 1};
  });

    it('should do something', function () {
        var ctrl = createComponentController();

      });

  });
```

Servicios

- Servicios
  - El servicio que probamos lo resuelve Angular
  - Necesitamos acceder a $provide si queremos inyectar mocks

- 2 beforeEach
  - Inicialización del módulo
    - Incluye una función para personalizar $provide
    - Definimos los mocks en este beforeEach
  - Inyección de dependencias de angular con inject

Servicios

```
describe('Test Service', function() {

    //declaramos lasvariables
    var $httpBackend, testService;

    //Inicializamos el modulo
    beforeEach(module('moduleName'));


    //Inyeccion de dependencias
    beforeEach(inject(function(_$httpBackend_, _TestService_) {
        $httpBackend = _$httpBackend_;
        testService = _TestService_;
    }));


    it('should do something', function() {
    });

});
```

$httpBackend

- Si queremos comprobar que una petición ha sido enviada o no por la aplicación.
  - $httpBackend.expectPOST(‘/url/to/post);

- Si queremos crear un backend mock.
  - Creamos respuestas mockeadas a peticiones.
  - $httpBackend.whenPOST(‘/url/to/post’).respond(responseMock);