actibyti] a netmind social project

JASMINE

© 2017, ACTIBYTI PROJECT SLU, Barcelona
Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

ÍNDICE DE CONTENIDOS

1 CASO PRÁCTICO

Caso Práctico: BananaTube Calidad

"BananaTube" es el proyecto estrella de Banana Apps.

BananaTube será el próximo boom! de las redes sociales; permitirá a sus usuarios gestionar videos, exponerlos en su muro, comentar videos propios y de sus amigos, calificarlos y compartirlos en varios canales.

En esta etapa del proyecto se garantizar que el desarrollo del frontend tiene la calidad necesaria y que futuros desarrollos no estropean el trabajo previo realizado.

Discutamos

- Cómo podemos garantizar la validez y calidad de nuestro software?
- Y en versiones futuras, ¿Cómo podemos garantizar que lo anterior sigue funcionando?
- ¿Qué metodología y qué herramientas necesitaremos adoptar?

1 Introducción

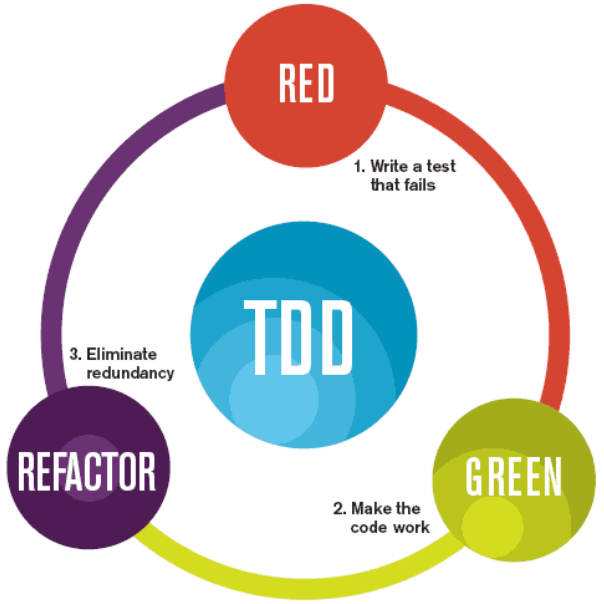
Jasmine

- Jasmine es un framework de pruebas orientado al comportamiento para JavaScript.
  - https://jasmine.github.io/

- Permite automatizar pruebas unitarias que se utilizan en la metodología BDD (Desarrollo Guiado por Comportamiento)
- Jasmine trata de hacer la vida mas fácil al desarrollador JavaScript. Posee las siguientes características:
  - Tiene una sintaxis muy clara.
  - Funciones que permiten la escritura sencilla.
  - Cada test creado por el usuario se convierte en función.

TDD (Test-Driven Development)

- El desarrollo basado en pruebas (TDD) es un proceso de desarrollo de software que se basa en la repetición de un ciclo de desarrollo muy corto:
  - Escribir las pruebas.
  - Observar los fallos.
  - Hacer que sucedan.
  - Restructurar el código.
  - Repetir



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

BDD (Behavior-Driven Development)

- BDD (Desarrollo Guiado por Comportamiento), es un proceso de desarrollo que surgió de TDD; combina técnicas y principios de TDD con ideas de diseño de dominio impulsada y un análisis orientado a objeto.

- BDD se centra en:
  - Establecer los objetivos que los diferentes actores requieren para la implementación de la visión
  - Involucrar a las partes interesadas en el proceso de implementación a través del desarrollo de software de fuera hacia dentro
  - Utilizar ejemplos para describir el comportamiento de la aplicación, o de unidades de código
  - Automatizar las pruebas para proporcionar feedback rápido y pruebas de regresión

2 Sintaxis de Jasmine

Sintaxis de Jasmine

- Las pruebas de Jasmine constan principalmente de dos parte:
  - **Describe** → define el bloque a ser probado
  - **IT** → define un caso de prueba específico para un bloque

- Ambos, **describe** e **it**, toman dos parámetros:
  - una cadena de texto
  - una función de callback

- Dentro del cuerpo del it, se definirán las configuraciones y especificaciones de las pruebas, mediante aserciones o comparaciones.

Sintaxis de Jasmine

```
describe('JavaScript addition operator', function () {
    it('adds two numbers together', function () {
        expect(1 + 2).toEqual(3);
    });
});
```

- La cadena que se pasa a **describe** y a **it**, ambas son sentencias:
  - "JavaScript addition operator adds two numbers together." (Operador de suma JavaScript agrega dos numeros).
- Dentro del bloque **it,** se puede escribir todo el código para la prueba. En el ejemplo:
  - "expect 1 + 2 to equal 3" (Espera que 1+2 sea igual a 3).
- Cualquier valor que se coloque en **expect** se pondrán a prueba y se podrá comparar usando un comparador, en este caso **toEqual**
  - Esto dará un resultado positivo o negativo para la prueba

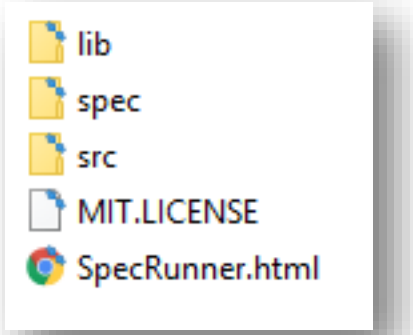
3 Configuración de un proyecto

Configuración de un proyecto

- Jasmine puede utilizarse de manera independiente o, integrarse con otro proyecto, además de poder ejecutarse fuera del navegador (por ejemplo en Node).
- Por defecto ya viene con una plantilla bastante agradable en la descarga.
- Lo primero que debemos hacer es ir a la siguiente url e iniciar la descarga de su ultima versión.
  - https://github.com/jasmine/jasmine/releases

Configuración de un proyecto

- Al descargar y descomprimir el framawork se obtendrá la siguiente estructura de plantilla:



- Los archivos del framework están en la carpeta **lib**.
- El código JavaScript a probar debe ubicarse en la carpeta **src**
- La carpeta **spec** almacena el código de las especificaciones de las pruebas.
- El **SpecRunner.html** es el encargado de **ejecutar las pruebas** en un navegador.

Configuración de un proyecto

- Para probar un módulo es necesario definir el archivo de especificaciones para el módulo.
- Por ejemplo para un módulo c**onvert.js** en la carpeta **src**, crearemos el archivo **ConvertSpec.js** en la carpeta **spec**.
- Modificaciones el **SpecRunner.html** para que se pruebe el módulo:

```
<script src="src/convert.js"></script>
<script src="spec/convertSpec.js"></script>
```

Definiendo las pruebas

Módulo a ser probado

- Usaremos un módulo de conversión de sistemas para distancias y volumen para las pruebas

```
function Convert(number, fromUnit) {
    var conversions = {
            distance : {
                meters : 1,
                cm     : 0.01,
                feet   : 0.3048,
                inches : 0.0254,
                yards  : 0.9144
            },
            volume : {
                liters : 1,
                gallons: 3.785411784,
                cups   : 0.236588236
            }
        },
        betweenUnit = false,
        type, unit;
```

http://jsbin.com/zehurodava/edit?js,console

Definir la estructura global de las pruebas

```
describe( "Convert library", function () {
    describe( "distance converter", function () {

    });

    describe( "volume converter", function () {

    });
});
```

- En el código se anidan los **describe**.
- Esto es perfectamente legal. Es una gran manera de probar los pedazos separados de la funcionalidad de la misma base de código.
- En lugar de dos llamadas separadas para conversiones a distancia y de volumen de la librería Convert, podemos tener un conjunto más descriptivo de pruebas como esta.

Definir los casos de prueba

- Concretamos los casos de prueba de distancia y volumen

```
describe( "distance converter", function () {
    it("converts inches to centimeters", function () {
        expect(Convert(12, "in").to("cm")).toEqual(30.48);
    });

    it("converts centimeters to yards", function () {
        expect(Convert(2000, "cm").to("yards")).toEqual(21.87);
    });
});

describe( "volume converter", function () {
    it("converts litres to gallons", function () {
        expect(Convert(3, "litres").to("gallons")).toEqual(0.79);
    });

    it("converts gallons to cups", function () {
        expect(Convert(2, "gallons").to("cups")).toEqual(32);
    });
});
```

Pruebas extra

- Añadimos un par de casuísticas para conversión de unidades inexistentes.
- Usamos además el matcher **toThrow**, que genera un throw de error.

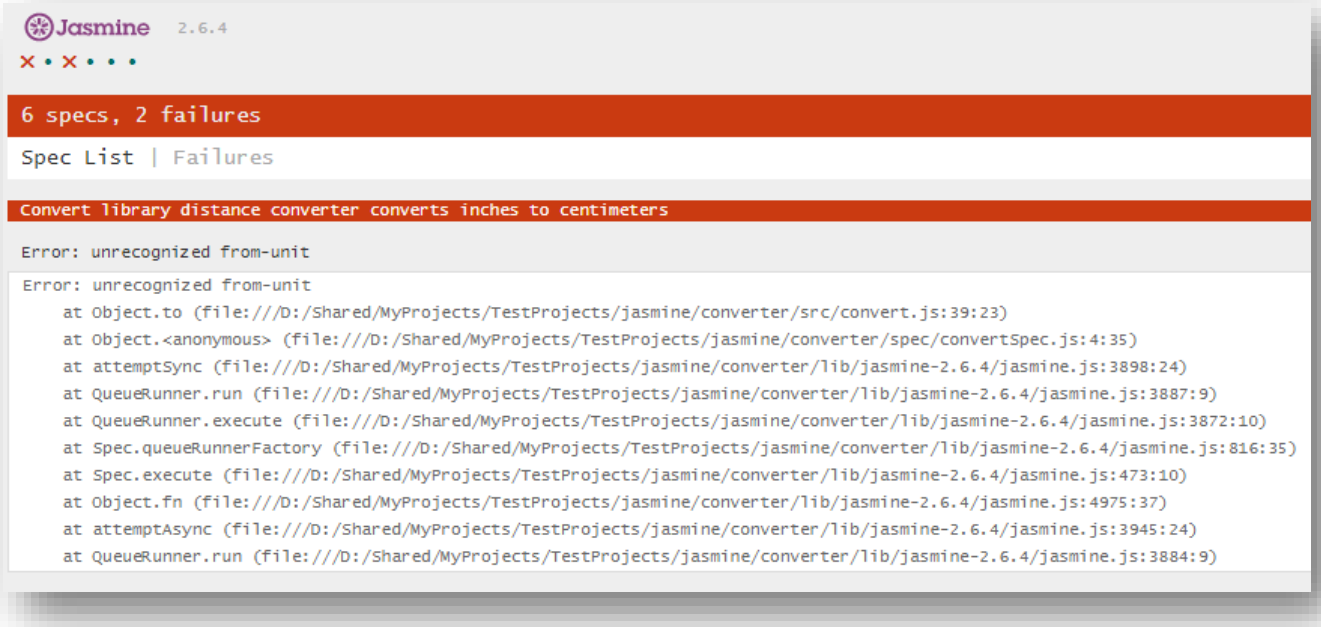
```
describe( "Convert library", function () {
...
it("throws an error when passed an unknown from-unit", function () {
    var testFn = function () {
        Convert(1, "dollar").to("yens");
    }
    expect(testFn).toThrow(new Error("unrecognized from-unit"));
});

it("throws an error when passed an unknown to-unit", function () {
    var testFn = function () {
        Convert(1, "cm").to("furlongs");
    }
    expect(testFn).toThrow(new Error("unrecognized to-unit"));
});

});
```

Pasar las pruebas

- Si abrimos el archivo **SpecRunner.html** en el navegador, obtendremos el resultado de las pruebas.



- Esto nos permitirá visualizar test que fallan y realizar las refactorizaciones necesarias para que funcione.
- Tras las modificaciones volveremos a pasar el **SpecRunner.html**

4 Comparadores

Comparadores

- Hasta el momento solo hemos visto los comparadores **toEqual** y **toThrow**. Sin embargo, hay muchos otros, pero mencionaremos los mas básicos:

  - **toBeDefined / toBeUndefined**: Se utiliza para definir una variable, de igual forma para confirmar que una variable y/o propiedad se encuentra **Undefined**.

```
it("is defined", function () {
    var name = "Andrew";
    expect(name).toBeDefined();
})

it("is not defined", function () {
    var name;
    expect(name).toBeUndefined();
});
```

Comparadores

- **toBeTruthy/ toBeFalsy**: Valida si algo es verdadero o falso.

```
it("is true", function () {
    expect(Lib.isAWeekDay()).toBeTruthy();
});
it("is false", function () {
    expect(Lib.finishedQuiz).toBeFalsy();
});
```

- **toBeLessThan/ toBeGreaterThan**: Valida si un número es mayor a o menor a.

```
it("is less than 10", function () {
    expect(5).toBeLessThan(10);
});
it("is greater than 10", function () {
    expect(20).toBeGreaterThan(10);
});
```

Comparadores

- **toMatch**: Toma parte del texto de salida que debe coincidir con una expresión regular.

```
it("outputs the right text", function () {
    expect(cart.total()).toMatch(/\$\d*.\d\d/);
});
```

- **toContain**: Comprueba si una matriz o cadena contienen un elemento o subcadena.

```
it("should contain oranges", function () {
    expect(["apples", "oranges", "pears"]).toContain("oranges");
});
```

5 BeforeEach y afterEach

beforeEach - afterEach

- A menudo, cuando se prueba un código base, se desea realizar algunas pruebas con líneas de código de la disposición en una serie.
- Sería complicado tener que generar una llamada **it** por cada línea, por lo que Jasmine permite que designemos que un código sea ejecutado antes o después de cada prueba usando las asignaciones **beforeEach** y **afeterEach**

beforeEach - afterEach

- En el siguiente ejemplo, se puede ver cómo antes de ejecutar cada prueba, el estado de **obj** se establece en **“clean”**, si no hiciéramos esto, por defecto los objetos de la prueba anterior persisten en la siguiente prueba.

```
describe("MyObject", function () {
    var obj = new MyObject();

    beforeEach(function () {
        obj.setState("clean");
    });

    it("changes state", function () {
        obj.setState("dirty");
        expect(obj.getState()).toEqual("dirty");
    })
    it("adds states", function () {
        obj.addState("packaged");
        expect(obj.getState()).toEqual(["clean", "packaged"]);
    })
});
```

Cubrir Antes y Después

- Podemos hacer algo similar con la fuction **AfterEach:**

```
describe("MyObject", function () {
    var obj = new MyObject("clean"); // sets initial state

    afterEach(function () {
        obj.setState("clean");
    });

    it("changes state", function () {
        obj.setState("dirty");
        expect(obj.getState()).toEqual("dirty");
    })
    it("adds states", function () {
        obj.addState("packaged");
        expect(obj.getState()).toEqual(["clean", "packaged"]);
    })
});
```

6 Comparadores personalizados

Comparadores personalizados

- Muchas veces resulta útil el construir un comparador personalizado.
- Podemos añadir un matcher pesonalizado en cada llamada a **beforEach** o **it** de una manera sencilla usando **this.addMatchers**

```
beforeEach(function () {
    this.addMatchers({

    });
});
```

- Cada clave en el objeto **this** se considerará un nombre de Matcher y su función asociada, aquello que se ejecutará.

Comparadores personalizados

- Por ejemplo, queremos crear un matcher que compruebe si un número está entre otros dos.

```
beforeEach(function () {
    this.addMatchers({
        toBeBetween: function (rangeFloor, rangeCeiling) {
            if (rangeFloor > rangeCeiling) {
                var temp = rangeFloor;
                rangeFloor = rangeCeiling;
                rangeCeiling = temp;
            }
            return this.actual > rangeFloor && this.actual < rangeCeiling;
        }
    });
});
```

- Hemos tomado dos parámetros, nos aseguramos que el primero es menor que el segundo, y devolveos una declaración booleana que se evalúa como verdadera si cumple nuestra condición.
- Lo importante a notar aquí es cómo recuperamos el valor que fue pasado a la función de **expect**: usando **this.actual**.

Comparadores personalizados

- Podemos usar nuestro matcher personalizado como otro cualquiera

```
it("is between 5 and 30", function () {
    expect(10).toBeBetween(5, 30);
});

it("is between 30 and 500", function () {
    expect(100).toBeBetween(500, 30);
});
```

7 Espias

Espías

- En Jasmine, un espía hace mas o menos lo que su nombre indica, permite espiar piezas de un programa.
- En Jasmine existe la función **spyOn()**, la cual no se puede configurar ni alterar.
- En el siguiente ejemplo, tenemos una clase que llama a **Person**, se puede decir “hello” en general o saludar a alguien.

```
var Person = function () {};

Person.prototype.helloSomeone = function (toGreet) {
    return this.sayHello() + ' ' + toGreet;
};

Person.prototype.sayHello = function () {
    return 'Hello';
};
```

Espías

- Tenemos una clase que hace varias cosas; sin embargo lo que queremos saber es si se llama la función **sayHello()** cuando llamamos a la función **helloSomeone()**.
- Para eso configuramos la siguiente prueba

```
describe('Person', function () {
    it('calls the sayHello() function', function () {
        var fakePerson = new Person();
        spyOn(fakePerson, 'sayHello');
        fakePerson.helloSomeone('world');
        expect(fakePerson.sayHello).toHaveBeenCalled();
  });
});
```

- Al hacer las pruebas dará resultados positivos, debido a que efectivamente se llama **fakePerson.sayHello()** desde **helloSomeone()**.
- En este caso, no importa si algo funciona correctamente, solo necesitamos saber si se ha llamado o no a una función

Asegurando la calidad de BananaTube

- Define los tests funcionales y de comportamiento para la primera historia de BananaTube
- Implementa los tests con Jasmine

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"