actibyti] a netmind social project

MOCHA & CHAI

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 Introducción

Introducción

- **Chai** es una **librería de aserción BDD/TDD** para Node y el navegador que puede ser emparejado con cualquier marco de prueba javascript.
  - http://chaijs.com/

Chai Assertion Library

- **Mocha** es un framework de pruebas JavaScript con múltiples funciones que se ejecuta en Node.js y en el navegador. Las pruebas Mocha se ejecutan en serie, lo que permite un informe flexible y preciso.
  - https://mochajs.org/

- En otras palabras, con Chai definimos las pruebas y con Mocha las ejecutamos

Pruebas unitarias

- Al probar el código base, se toma una parte del código, normalmente una función, y se verifica su comportamiento en situaciones específicas.
- Las pruebas unitarias es una forma estructurada y automatizada de hacerlo. Como resultado, mientras mas pruebas se realicen, mayor es la confianza obtenida en base al código.
- Por ejemplo:

```
// Given 1 and 10 as inputs...
var result = Math.max(1, 10);

// ...we should receive 10 as the output
if(result !== 10) {
  throw new Error('Failed');
}
```

Pruebas unitarias

- La idea central es evaluar el comportamiento de una función cuando se le da un conjunto de valores de entrada, comprobando el resultado obtenido.
- A veces las pruebas pueden ser mas complejas, por ejemplo, si una función realiza peticiones Ajax, la prueba necesita una configuración más compleja
- En todo caso la definición de "**con ciertas entradas, se espera un resultado específico**" sigue siendo válido.

2 Configuración

Instalar Mocha y Chai

- La manera mas fácil de instalar Mocha es usando **npm** (Node debe estar instalado)
- Abrir un terminal o una línea de comandos en el directorio del proyecto, de este modo, se instalan los paquetes de mocha y chai.

  - Para instalar Mocha
    - En el proyecto

      ```
      npm install --save-dev mocha
      ```

    - De manera global

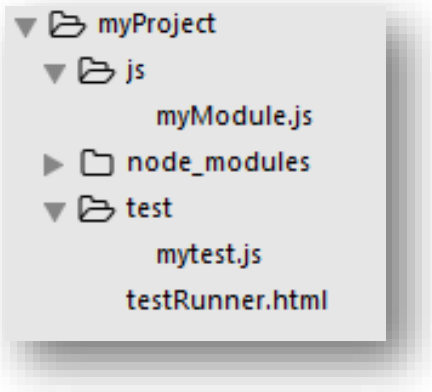
      ```
      npm install -g mocha
      ```

  - Para instalar Chai

    ```
    npm install chai
    ```

Creación de una estructura de directorios

- Se debe realizar las pruebas dentro de un directorio separado de las fuentes de código.
- La práctica mas popular en JavaScript es tener un directorio llamado **test** en el directorio raíz de su proyecto.
- En él se coloca cada archivo de prueba: **test/someModuleTest.js**
- La estructura del proyecto debe ser similar a:



Ejecutor de pruebas

- Para poder ejecutar las prueba en un navegador, es necesario configurar una pagina HTML simple para que sea nuestro ejecutor de pruebas (ver siguiente diapositiva).

Ejecutor de pruebas

```
<!DOCTYPE html>
<html>
<head>
  <title>Mocha Tests</title>
  <link rel="stylesheet" href="node_modules/mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <script src="node_modules/mocha/mocha.js"></script>
  <script src="node_modules/chai/chai.js"></script>
  <script>mocha.setup('bdd')</script>

  <!-- load code you want to test here -->

  <!-- load your test files here -->

  <script>
    mocha.run();
  </script>
</body>
</html>
```

- Descarga de los estilos CSS de Mocha para generar resultados formateados.
- Crear un div con el ID mocha. Aquí se insertarán los resultados de las pruebas.
- Incluye los scripts Mocha y Chai; que están ubicados en subcarpetas de **node_modules**.
- Luego, cargamos el código que queremos poner a prueba y los archivos de prueba.
- Por último, llamamos a ejecutar las pruebas.

Pongámoslo en práctica

- Instala Mocha y Chai
- Crea y configura un proyecto para pruebas

3 Ejecución de pruebas

Bloques de construcción básicos.

- Comenzaremos creando un nuevo archivo de prueba individual (un caso de prueba)
- Cada archivo de caso de prueba sigue el mismo patrón básico.
  - En primer lugar, se debe describir el bloque de la siguiente manera:

```
describe('Array', function() {
  // Further code for tests goes here
});
```

  - La función **describe** se utiliza para pruebas de grupo individuales, el primer parámetro debe indicar lo que se está probando, en este caso, ya se va a probar una función de arrays.

Bloques de construcción básicos.

- En segundo lugar, dentro de la función **describe**, tendremos bloques **it** de la siguiente forma:

```
describe('Array', function() {
  it('should start empty', function() {
    // Test implementation goes here
  });

  // We can have more its here
});
```

- El bloque **it** se utiliza para crear las pruebas específicas.
  - El **primer parámetro** de *it* debe proporcionar una descripción legible de la prueba: “should start empty” (Debería comenzar vacío), lo cual es una buena descripción de como debe comportarse los arrays.
  - El segundo parámetro es una función con la programación de la prueba

Código de prueba

- La prueba completa quedaría de la siguiente manera:

```
var assert = chai.assert;

describe('Array', function() {
  it('should start empty', function() {
    var arr = [];

    assert.equal(arr.length, 0);
  });
});
```

- En la primera línea, se ha creado la variable **assert**, para no tener que repetir **chai.assert**.
- En la función **it** se crea una matriz y se comprueba su longitud.
- Por último se hace la validación usando una aserción que comprueba el resultado.
- La mayoría de las funciones asert toman parámetros en el mismo orden:
  - El **valor real**, es el resultado del código de prueba, en este caso **arr.length**.
  - El **valor esperado**, es el resultado que debería arrojar. Para este caso, que la matriz debe comenzar vacía, el valor esperado es 0 (cero).

Ejecución de la prueba

- Para ejecutar la prueba, debemos añadir la referencia de las misma en el ejecutor de pruebas, la prueba que hemos creado.
  - Se debe hacer después de setup de Mocha

```
<script src="node_modules/mocha/mocha.js"></script>
  <script src="node_modules/chai/chai.js"></script>
  <script>mocha.setup('bdd')</script>

  <script src="test/arrayTest.js"></script>
```

- Una vez agregado al código, puede cargar la página en el navegador.

Resultados de la prueba

- Cuando se ejecuta la prueba, los resultados se verán de la siguiente manera:

Cambios en la prueba

- En la línea de la prueba : `assert.equal(arr.length, 0);`
  - Reemplaza el número 0 con 1, esto hace que la prueba falle, ya que la longitud de la matriz no coincide con el valor esperado.

- Ejecutar la prueba de nuevo. Se verá que la prueba aparecerá en rojo con una descripción del error.

Mensajes de error

- Uno de los beneficios de las pruebas es que ayudan a encontrar errores de manera más rápida, sin embargo este tipo de mensajes de error no es muy útil.
- La mayor parte de la funciones de afirmación, pueden tomar un **mensaje** opcional como parámetro.
- Este es el mensaje que se muestra cuando la aserción falla. De esta manera, el mensaje de error es mas sencillo de comprender.
- Podemos añadir un mensaje a nuestra afirmación, de la siguiente manera:

```
assert.equal(arr.length, 1, 'Array length was not 0');
```

- Si volvermos a ejecutar la prueba este se mostrará

Probando una función

La función

- Vamos a probar una función que añade una clase CSS a un elemento.
- La función irá en el archivo **js/className.js**

```
function addClass(el, newClass) {
  if(el.className.indexOf(newClass) === -1) {
    el.className += newClass;
  }
}
```

- Generaremos asimismo un archivo de pruebas **test/classNameTest.js**

Configurar el ejecutor

- Para ejecutar la prueba desde el navegador, se debe añadir en el corredor **className.js** y **classNameTest.js**.

```
<!-- load code you want to test here -->
<script src="js/className.js"></script>

<!-- load your test files here -->
<script src="test/classNameTest.js"></script>
```

Definición de las pruebas

- Para que sea un poco más interesante, debemos añadir una nueva clase sólo si esa clase no existe en la propiedad className de un elemento. Es decir no queremos que suceda algo como:

```
<div class="hello hello hello hello">
```

- Recordemos la idea básica detrás de las **pruebas unitarias**:
  - Damos a la función ciertas entradas y luego verificamos que la función se comporta como se esperaba.
  - ¿Cuáles son las entradas y los comportamientos para esta función?

- Dado un elemento y un nombre de clase:
  - Si la propiedad className del elemento no contiene el nombre de la clase, debe agregarse.
  - Si la propiedad className del elemento contiene el nombre de la clase, no debería agregarse.

Configuración de las pruebas

- Traducimos la definición de las pruebas en classNameTest.js y ejecutamos el testRunner → OK

```
describe('addClass', function() {
  it('should add class to element');
  it('should not add a class which already exists');
});
```

- Añadimos cuerpo a la segundo prueba y ejecutamos → OK

```
it('should not add a class which already exists', function() {
  var element = { className: 'exists' };

  addClass(element, 'exists');

  var numClasses = element.className.split(' ').length;
  assert.equal(numClasses, 1);
});
```

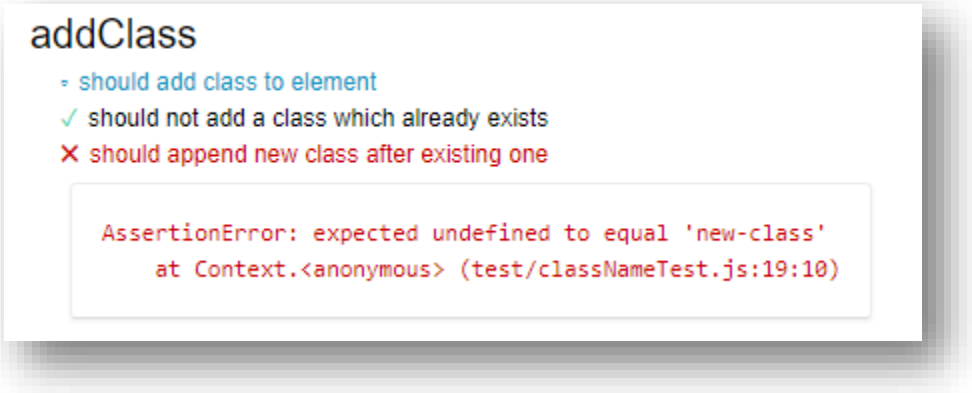
Configuración de las pruebas

- Añadimos más pruebas según los requerimientos funcionales; por ejemplo que añada nuevas clases al final → NOK

```
it('should append new class after existing one', function() {
  var element = { className: 'exists' };

  addClass(element, 'new-class');

  var classes = element.className.split(' ');
  assert.equal(classes[1], 'new-class');
});
```



Modificación del código

- Refactorizamos className.js para que cumpla con la especificación.

```
function addClass(el, newClass) {
  if(el.className.indexOf(newClass) !== -1) {
    return;
  }

  if(el.className !== '') {
    //ensure class names are separated by a space
    newClass = ' ' + newClass;
  }

  el.className += newClass;
}
```

- Ejecutamos las pruebas → OK

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"