



Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

Desarrollo e implantación de Sistemas de Software (Grupo 501)

Actividad de Pruebas con Jasmine en Equipo

Alejandro Daniel Moctezuma Cruz A01736353

Daniela Lozada Bracamontes A01736594

José Luis Zago Guevara A01736278

César Guerra Martínez A01656774

Hugo Muñoz Rodríguez A01736149

Christian Flores Albert A01734997

Profesor: Juan Manuel Gonzalez Calleros

Miércoles 15 de Mayo del 2024

REPOSITORIO

El contenido del repositorio en el cual se trabajaron los archivos se encuentra en el siguiente enlace:

https://github.com/AlejandroMoc/E1_JasmineTests

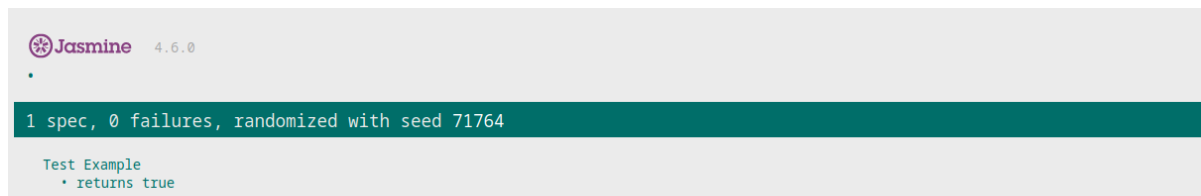
PRUEBAS REALIZADAS Y RESULTADOS

1: Verificación básica de la funcionalidad de Jasmine.

Para la primera parte de esta prueba, es necesario copiar el siguiente código en el archivo main.js del proyecto:

```
describe('Test Example', () => {  
  it('returns true', () => {  
    expect(true).toBe(true);  
  
  });  
});
```

Tras ello, al abrir el archivo html obtendremos el siguiente resultado, el cual refleja que se ha logrado obtener el resultado correspondiente:



Una vez hecho esto, podemos continuar con la segunda parte del funcionamiento de Jasmine. Para esto, copiamos ahora el siguiente trozo de código en el archivo main.js.

```
describe('Test Example', () => {  
  it('returns true', () => {  
    expect(false).toBe(true);  
  
  });  
});
```

Esto ocasiona que exista un fallo en la comprobación de la prueba, dado que el valor brindado no es igual al esperado. Esto da el siguiente resultado:

```
Jasmine 4.6.0
x

1 spec, 1 failure, randomized with seed 34521
Spec List | Failures

Test Example > returns true
Expected false to be true.
<Jasmine>
@file:///home/daniel/Documentos/Escuela%2056/jasmine-standalone-4.6.0/spec/main.js:3:16
<Jasmine>
```

2: Implementación y prueba de la función insertDashes.

Para comenzar esta prueba, es necesario introducir el siguiente código en el archivo main.js:

```
function insertDashes(str) {
  // write code here
}
/**
 * Test Suite
 */
describe('insertDashes()', () => {
  it('insert dashes in between chars', () => {
    // arrange
    const value = "aba caba";

    // act
    const result = insertDashes(value);
    // log
    console.log("result: ", result);

    // assert
    expect(result).toBe("a-b-a c-a-b-a");
  });
});
```

Esto ocasiona que se despliegue el siguiente error al ejecutar la prueba:

```
Jasmine 4.6.0
x

1 spec, 1 failure, randomized with seed 11835
Spec List | Failures

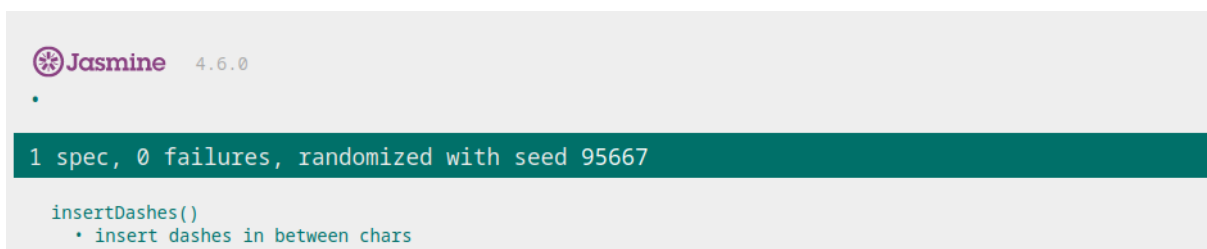
insertDashes() > insert dashes in between chars
Expected undefined to be 'a-b-a c-a-b-a'.
<Jasmine>
@file:///home/daniel/Documentos/Escuela%2056/jasmine-standalone-4.6.0/spec/main.js:18:17
<Jasmine>
```

¿Por qué falla la prueba? Esta falla debido a que si bien se encuentra bien definido el valor esperado, no existe un valor definido en la variable “result”. Esto sucede así puesto que no existe un código definido para la función “insertDashes”, lo que hace que la variable “result” jamás tenga un valor.

Para la siguiente parte de la prueba es necesario ingresar este trozo de código:

```
function insertDashes(str) {  
  // write code here  
}  
/**  
 * Test Suite  
 */  
describe('insertDashes()', () => {  
  it('insert dashes in between chars', () => {  
    // arrange  
    const value = "aba caba";  
  
    // act  
    const result = insertDashes(value);  
    // log  
    console.log("result: ", result);  
  
    // assert  
    expect(result).toBe(undefined);  
  });  
});
```

Al esperarse ahora un resultado no definido, el resultado arrojado es que ahora la prueba se ha comprobado satisfactoriamente.

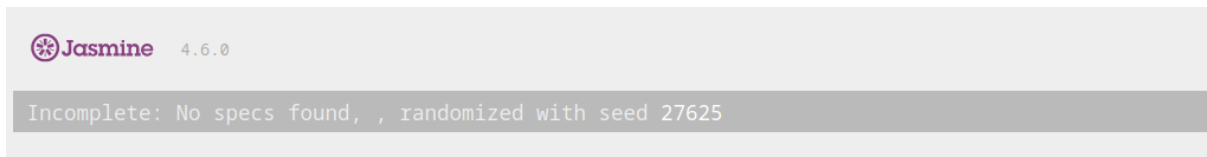


The image shows the Jasmine test runner interface. At the top, the Jasmine logo and version 4.6.0 are displayed. Below this, a green bar indicates the test results: "1 spec, 0 failures, randomized with seed 95667". Underneath the green bar, the test suite "insertDashes()" is listed, followed by the specific test "insert dashes in between chars".

```
Jasmine 4.6.0  
.  
  
1 spec, 0 failures, randomized with seed 95667  
  
insertDashes()  
  • insert dashes in between chars
```

3: Implementación y pruebas para la clase User.

Para la primera parte de esta prueba es necesario eliminar el código ya existente en el archivo main.js, lo que deja el siguiente resultado en el navegador:



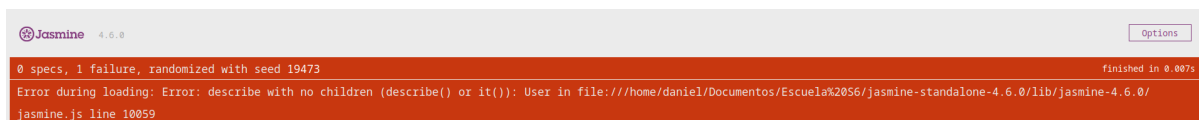
Para el primer paso de la prueba, es necesario pegar el siguiente código en main.js, el cual es diferente al de las primeras pruebas.

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data = {}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName;
  }
}

/**
 * Test Suite
 */
describe('User', () => {
});
```

Al abrir esto en el navegador, se mostrará un error, puesto que no existe prueba alguna que verificar en el código. Es decir, no existe valor alguno que esté siendo mandado a llamar como parámetro del describe para la clase User.



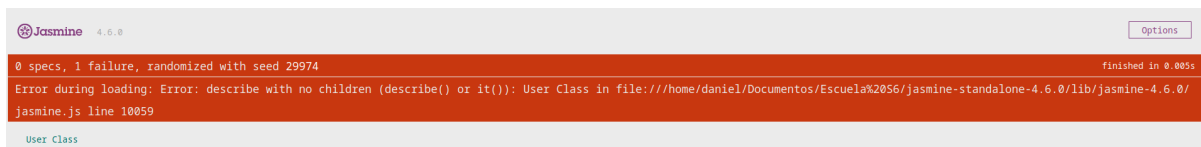
Tras ello es necesario escribir el siguiente código, mismo que ahora posee una referencia a la clase “User”.

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data = {}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName;
  }
}

/**
 * Test Suite
 */
describe(`${User.name} Class`, () => {
});
```

Esto arroja la siguiente respuesta en el navegador, siendo que aún no existen atributos en el “describe”:



Tras ello, podemos cambiar el nombre de la clase de “User” a “Admin”, quedando el código de la siguiente manera:

```
class Admin {
  firstName;
  lastName;
  middleName;

  constructor(data = {}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName;
  }
}

describe(`${User.name} Class`, () => {
});
```

Al no existir ahora una clase “User”, esto arroja el siguiente error en el navegador, mencionando que ocurrió un error durante la carga:



Como siguiente paso se utiliza el siguiente código, en donde se renombra nuevamente la clase a “User” y se define una prueba:

```
class User {
  firstName;
  lastName;
  middleName;
  constructor(data = {}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName;
  }
}

/**
 * Test Suite
 */
describe(`${User.name} Class`, () => {
  it('first name defaults to empty', () => {
    // arrange
    const data = { firstName: null };
    // act
    const model = new User(data);
    // assert
    expect(model.firstName).toBe('');
  });
});
```

Esto devuelve el siguiente resultado, en donde se muestra que la prueba se realizó correctamente.

```
1 spec, 0 failures, randomized with seed 70404
```

```
User Class
  • first name defaults to empty
```

Para el siguiente paso es necesario usar el siguiente código, donde “firstName” deja de ser vacío y pasa a ser de tipo null:

```
class User {
  firstName;
  lastName;
  middleName;
  constructor(data = {}){
    this.firstName = data.firstName;
    this.lastName = data.lastName || '';
    this.middleName = data.middleName;
  }
}

/**
 * Test Suite
 */
describe(`${User.name} Class`, () => {
  it('first name defaults to empty', () => {
    // arrange
    const data = { firstName: null };
    // act
    const model = new User(data);
    // assert
    expect(model.firstName).toBe('');
  });
});
```

Esto arroja el siguiente resultado, en el cual efectivamente podemos contemplar que las variables de tipo nulo y de tipo vacío no son del mismo tipo, ni tampoco se comparan entre sí de la misma manera:

```
1 spec, 1 failure, randomized with seed 20992
```

```
Spec List | Failures
```

```
User Class > first name defaults to empty
```

```
Expected null to be ''.
```

```
<Jasmine>
```

```
@file:///home/daniel/Documentos/Escuela%20S6/jasmine-standalone-4.6.0/spec/main.js:21:26
```

```
<Jasmine>
```


4:

Para este paso es necesario crear el archivo user.js en la carpeta raíz. Este archivo contiene el siguiente código:

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data={}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName || '';
  }
}
```

La referencia deberá ser agregada al archivo index.html, el cual tendrá el siguiente contenido:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v4.6.0</title>

  <link rel="shortcut icon" type="image/png"
href="lib/jasmine-4.6.0/jasmine_favicon.png">
  <link rel="stylesheet" href="lib/jasmine-4.6.0/jasmine.css">

  <script src="lib/jasmine-4.6.0/jasmine.js"></script>
  <script src="lib/jasmine-4.6.0/jasmine-html.js"></script>
  <script src="lib/jasmine-4.6.0/boot0.js"></script>
  <!-- optional: include a file here that configures the Jasmine env -->
  <script src="lib/jasmine-4.6.0/boot1.js"></script>
  <script type="text/javascript" src="./user.js"></script>

  <!-- include spec files here... -->
  <script src="spec/main.js"></script>

</head>

<body>
</body>

</html>
```

De igual forma se reescribe el archivo main.js, el cual ahora contiene lo siguiente:

```
//Test Suite
describe(`${User.name} Class`, () => {

  describe(`default values`, () => {
    it('first name defaults to empty', () => {
      // arrange
      const data = { firstName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.firstName).toBe('');
    });

    it('last name defaults to empty', () => {
      // arrange
      const data = { lastName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.lastName).toBe('');
    });

    it('middle name defaults to empty', () => {
      // arrange
      const data = { middleName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.middleName).toBe('');
    });
  });
});
```

Al ejecutar esto, se muestra el siguiente resultado en el navegador:

```
3 specs, 0 failures, randomized with seed 30169
```

```
User Class
  default values
    • first name defaults to empty
    • middle name defaults to empty
    • last name defaults to empty
```

Debido a la gran cantidad de código repetido en main.js, podemos reescribir este archivo de la siguiente manera:

```
//Test Suite
describe(`${User.name} Class`, () => {

  let model;

  beforeEach(()=>{
    model = new User();
  });

  describe(`default values`, () => {
    it('first name defaults to empty', () => {
      //assert
      expect(model.firstName).toBe('');
    });

    it('last name defaults to empty', () => {
      // assert
      expect(model.lastName).toBe('');
    });

    it('middle name defaults to empty', () => {
      // assert
      expect(model.middleName).toBe('');
    });
  });
});
```

En este caso el resultado es el mismo.

5:

Como primera parte, es necesario editar el archivo user.js de la siguiente manera:

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data={}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName || '';
  }

  get fullName(){
    if(this.middleName.length>0){
      return `${this.firstName} ${this.middleName[0]}.
${this.lastName}`;
    }
    return `${this.firstName} ${this.lastName}`
  }
}
```

Luego editamos el archivo main.js de la siguiente forma:

```
//Test Suite
describe(`${User.name} Class`, () => {
  describe('default values', () => {
    it('first name defaults to empty', () => {
      // arrange
      const data = { firstName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.firstName).toBe('');
    });

    it('last name defaults to empty', () => {
      // arrange
      const data = { lastName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.lastName).toBe('');
    });
  });
});
```

```

    it('middle name defaults to empty', () => {
      // arrange
      const data = { middleName: null };
      // act
      const model = new User(data);
      // assert
      expect(model.middleName).toBe('');
    });
  });

  describe('full name', ()=>{
    beforeEach(() => {
      model = new User({firstName: 'Dylan', lastName: 'Israel'});
    });

    it('middle initial when middlename is defined with first and last',()
=> {
      //arange
      model.middleName='Christopher';
      //act
      const result = model.fullName;
      //assert
      expect(result).toBe(`${model.firstName} ${model.middleName[0]}.
${model.lastName}`);
    });

    it('when no middle name return just first and last',() => {
      //arange
      model.middleName='';
      //act
      const result = model.fullName;
      //assert
      expect(result).toBe(`${model.firstName} ${model.lastName}`);
    });
  });
});

```

Esto da el siguiente resultado en el navegador:

5 specs, 0 failures, randomized with seed 95839

User Class

full name

- middle initial when middlename is defined with first and last
- when no middle name return just first and last

default values

- first name defaults to empty
- middle name defaults to empty
- last name defaults to empty

Alerts.

Se modifica el archivo user.js, agregando la función sayMyName, la cual retorna un aviso con el nombre.

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data={}){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName || '';
  }
  get fullName(){
    if(this.middleName.length>0){
      return `${this.firstName} ${this.middleName[0]}.
${this.lastName}`;
    }
    return `${this.firstName} ${this.lastName}`
  }
  sayMyName(){
    window.alert(this.fullName);
  }
}
```

Tras ello escribimos el archivo main.js de la siguiente manera:

```
//Test Suite
describe(`${User.name} Class`, () => {

  let model;

  beforeEach(()=>{
    model = new User();
  });

  describe(`say my name`, () => {
    it('alerts the full name of user', () => {
      //arrange
      model.firstName="Dylan";
      model.lastName="Israel";
      spyOn(window, 'alert');

      //act
      model.sayMyName();

      //asert
      expect(window.alert).toHaveBeenCalled();
    });
  });
});
```

Al ejecutarlo esto da como resultado la siguiente interfaz, la cual menciona una prueba satisfactoria:

```
1 spec, 0 failures, randomized with seed 22345
```

```
User Class
  say my name
    • alerts the full name of user
```

Para la siguiente parte de la prueba, añadimos una alerta en la cual se esperan los valores de nombre y apellido correspondientes. Es así que main.js queda de la siguiente forma:

```
//Test Suite

describe(`${User.name} Class`, () => {

  let model;
```

```

beforeEach(() => {

    model = new User();

});

describe(`say my name`, () => {

    it('alerts the full name of user', () => {

        //arrange

        model.firstName="Dylan";

        model.lastName="Israel";

        spyOn(window, 'alert');

        //act

        model.sayMyName();

        //asert

        expect(window.alert).toHaveBeenCalled();

        expect(window.alert).toHaveBeenCalledWith('Dylan Israel');

    });

});

});

```

API Calls - Ejemplo 1.

Para comenzar este ejemplo, tenemos que crear un archivo “user-service.js”, el cual tendrá el siguiente contenido:

```

class UserService {
    getUserById(id){
        return fetch(`https://jsonplaceholder.typicode.com/users/${id}`);
    }
}

```


Si siguiendo con esto, es necesario modificar el archivo user.js con el siguiente código:

```
class User {
  firstName;
  lastName;
  middleName;

  constructor(data,userService){
    this.firstName = data.firstName || '';
    this.lastName = data.lastName || '';
    this.middleName = data.middleName || '';
    this.id = data.id;
    this.userService = userService;
  }

  get fullName(){
    if(this.middleName.length>0){
      return `${this.firstName} ${this.middleName[0]}.
${this.lastName}`;
    }
    return `${this.firstName} ${this.lastName}`
  }

  async getMyFullUserData(){
    return this.userService.getUserById(this.id);
  }
  sayMyName(){
    window.alert(this.fullName);
  }
}
```

Tras ello se modifica un archivo index.html con el siguiente código:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v4.6.0</title>

  <link rel="shortcut icon" type="image/png"
href="lib/jasmine-4.6.0/jasmine_favicon.png">
  <link rel="stylesheet" href="lib/jasmine-4.6.0/jasmine.css">

  <script src="lib/jasmine-4.6.0/jasmine.js"></script>
```

```

<script src="lib/jasmine-4.6.0/jasmine-html.js"></script>
<script src="lib/jasmine-4.6.0/boot0.js"></script>
<!-- optional: include a file here that configures the Jasmine env -->
<script src="lib/jasmine-4.6.0/boot1.js"></script>
<script type="text/javascript" src="./user.js"></script>
<script type="text/javascript" src="./user-service.js"></script>

<!-- include spec files here... -->
<script src="spec/main.js"></script>

</head>
<body>
</body>
</html>

```

Siguiendo con esto modificamos el archivo main.js con el siguiente contenido:

```

//Test Suite
describe(`${User.name} Class`, () => {
  let model;
  let mockUserService;

  beforeEach(()=>{
    mockUserService={
      lastId:null,
      user: {},
      getUserById(id){
        this.lastId=id;
        return this.user;
      }
    };
    const data = { firstName: 'Dylan', middleName: 'Christopher',
lastName: 'Israel', id: 1};
    model = new User(data, mockUserService);
  });

  describe(`getMyFullUserData`, () => {
    it('passes id to get user', async () => {
      //arrange
      mockUserService.lastId=null;

      //act
      await model.getMyFullUserData();

      //assert
      expect(mockUserService.lastId).toBe(1);
    });
  });
});

```

```

    it('gets full user data', async () => {
      //arrange
      mockUserService.user = new User({
        firstName: 'Juan',
        middleName: 'Programa',
        lastName: 'Bien :)',
        id: 2
      });
      //act

    });

  });
});

```

Esto da el siguiente resultado en el navegador:

2 specs, 0 failures, randomized with seed 73488

```

User Class
  getMyFullUserData
    • passes id to get user
    • SPEC HAS NO EXPECTATIONS gets full user data

```

API Calls - Ejemplo 2.

Para esto modificamos el archivo user.js con el siguiente contenido:

```

class User {
  firstName
  lastName
  middleName

  constructor(data = {}){
    this.firstName = data.firstName || ''
    this.lastName = data.lastName || ''
    this.middleName = data.middleName || ''
  }

  get fullName() {
    if(this.middleName.length > 0) {
      return `${this.firstName} ${this.middleName[0]}. ${this.lastName}`
    }
    return `${this.firstName} ${this.lastName}`
  }
}

```

```

    get fullNamePieces() {
      return [this.firstName, this.middleName, this.lastName];
    }
  }
}

```

Luego se modifica el archivo main.js con el siguiente contenido:

```

//Test Suite
describe(`${User.name} Class`, () => {
  it('exists', () => {
    //assert
    expect(User).toBeDefined()
  });

  let model;

  beforeEach(() => {
    model = new User();
  });

  describe('additional matcher examples', () => {
    //toBeDefined(), toEqual()

  });
});

```

Esto da el siguiente resultado en el navegador, debido a que no existe prueba alguna en el describe:

1 spec, 1 failure, randomized with seed 20292

Spec List | Failures

User Class

Error: describe with no children (describe() or it()): User Class additional matcher examples in file:/.

<Jasmine>

@file:///home/daniel/Documentos/Escuela%20S6/jasmine-standalone-4.6.0/spec/main.js:14:10

<Jasmine>

@file:///home/daniel/Documentos/Escuela%20S6/jasmine-standalone-4.6.0/spec/main.js:2:9

Si le quitamos el describe, este es el resultado de la prueba:

1 spec, 0 failures, randomized with seed 38313

User Class
• exists

API Calls - Ejemplo 3.

Para esta prueba se debe modificar el archivo main.js con el siguiente contenido:

```
//Test Suite
describe(`${User.name} Class`, () => {
  it('exists', () => {
    //assert
    expect(User).toBeDefined()
  });

  let model;

  beforeEach(()=>{
    model = new User();
  });

  describe('additional matcher examples', () => {
    //toBeDefined(), toEqual()
    it('gets full name pieces', () => {
      //arrange
      const firstName = 'César';
      const middleName = 'Zaniru';
      const lastName = 'Guerra';

      //act
      model = new User({firstName, middleName, lastName});

      //assert
      expect(model.fullNamePieces).toEqual([firstName, middleName,
lastName]);
    });
  });
});
```

Esto arroja el siguiente resultado en el navegador:

2 specs, 0 failures, randomized with seed 04057

```
User Class
  • exists
  additional matcher examples
    • gets full name pieces
```

Ejercicio.

Modificamos el archivo main.js con el siguiente contenido:

```
//Test Suite
describe(`${User.name} Class`, () => {

  let model;

  beforeEach(()=>{
    model = new User();
  });

  describe('additional matcher testing area', () => {
    //toBeDefined(), toEqual()
    it('has my first name', () => {
      //arrange
      const firstName = 'Dylan';
      const lastName = 'Israel';

      //act
      model = new User({firstName, lastName});

      //assert
      expect(model.fullName).toMatch(/Dylan/);
    });
  });
});
```

Esto nos da el siguiente resultado en el navegador:

```
1 spec, 0 failures, randomized with seed 36965
```

```
User Class
  additional matcher testing area
    • has my first name
```

COMENTARIO CRÍTICO

Para las pruebas realizadas con Jasmine, consideramos que este es un buen inicio para probar el sistema y su funcionamiento, así como las ventajas que tiene la utilización de este sistema para generar pruebas y mostrar los resultados esperados.

Este parece ser un buen sistema para verificar si los contenidos se están enviando en un formato correcto, si se están enviando datos correspondientes y para activar o desactivar distintas pruebas durante el proceso de desarrollo de un sistema. Creemos que el uso de estas herramientas, si bien en el momento fue un tanto poco utilizado, presenta una buena oportunidad para identificar fallos o distinguir cómo es el funcionamiento de un sistema respecto a la lógica esperada.

También fue un buen punto el haber tenido diapositivas las cuales pudiésemos comparar para leer las instrucciones. Finalmente, creemos también que el utilizamiento de estas herramientas puede suponer un buen avance y profesionalismo a nuestro proyecto actual, y esperamos seguirlas utilizando.