Unit testing

Unit testing

- JavaScript es un lenguaje de tipado dinámico con gran poder de expresión, pero sin casi ninguna ayuda del compilador

Unit testing

- Cualquier código escrito en JavaScript tiene que venir con un sólido conjunto de tests

Unit testing

- Los de Angular se lo han currado para que testear aplicaciones sea fácil: No tienes excusa para no testear tu código

Unit testing

- Los test unitarios tratan de testear unidades individuales de código

Unit testing

- Responden a preguntas como "¿Funciona la lógica correctamente?" o "¿La función ordenarLista() lo hace bien y en el orden correcto?"

Unit testing

- Es muy importante poder aislar la unidad de código a probar

Unit testing

- No queremos tener que currarnos piezas relacionadas con nuestro código, como los elementos DOM, o hacer cualquier llamada XHR para recuperar datos

Unit testing

- Angular nos ayuda a separar y delegar las responsabilidades de código correctamente
- Ofrece inyección de dependencias para el mockeo de peticiones XHR

Unit testing

- Proporciona abstracciones que permiten testear el modelo sin tener que recurrir a la manipulación del DOM

Unit testing

- Permite testear componentes mucho más fácilmente, ya que se pueden inyectar mockeos de sus dependencias

Herramientas

- Para probar aplicaciones Angular hay ciertas herramientas a utilizar que harán que los tests sean mucho más fácil de instalar y ejecutar

Herramientas: Karma

- Karma es una herramienta de línea de comandos JavaScript que se puede utilizar para lanzar un servidor web que carga el código fuente de la aplicación y ejecuta los tests

Herramientas: Karma

- Es una aplicación Nodejs
- Configurable para funcionar contra diferentes navegadores

Herramientas: Jasmine

- Jasmine es un **framework** de desarrollo basado en pruebas (**test-driven**) de JavaScript

Herramientas: Jasmine

- Se ha convertido en la opción más popular para probar aplicaciones angular

Herramientas: Jasmine

- Ofrece funciones para ayudar en la estructuración de tests y hacer “**assertions**”

Herramientas: Jasmine

- Usamos la función “**describe**” para agrupar nuestros tests

```
describe('generating random numbers', function () {
    // Tests van aqui dentro
});
```

Herramientas: Jasmine

- Y luego cada test individual se define dentro de una llamada a la función `it`:

```
describe('generating random numbers', function () {
    it('should limit the range of numbers to MOD'),
function(){ // Test assertion goes here });
});
```

Herramientas: Jasmine

- Jasmine viene con un número de comparadores que permiten hacer diferentes “**assertions**”

Herramientas: Jasmine

- El la documentación se pueden ver todos los tipos de assertions

Jasmine & Karma

- Para utilizar Jasmine con Karma, utilizamos el test runner karma-jasmine

Angular mocks

- El módulo **ngMock** se utiliza para inyectar y servicios Angular simulados dentro de las pruebas unitarias

Angular mocks

- Una parte muy útile de ngMock es **$httpBackend**, que nos permite simular peticiones XHR y devolver datos que decidamos nosotros en su lugar

Ejercicios

- Abre la carpeta tests y vamos a ver qué tests tenemos

Ejercicios

- Crea 2 nuevos tests unitarios:
  - Uno para testear el servicio del **módulo service**
  - Otro para testear la **directiva foot**