ENUNCIADO PEC 4

Testing en Angular

Desarrollo front-end avanzado

Máster Universitario en Desarrollo de sitios y aplicaciones web

Universitat Oberta de Catalunya

Contenido

Introducción

Esta práctica se centra en presentar una introducción a los test utilizando **Angular**. Para esta práctica no se utilizará nuestro proyecto debido a que para los primeros pasos con test es mejor utilizar una aplicación que no utilice el patrón de datos **Redux**.

Para la realización de esta práctica utilizaremos el proyecto denominado "**Heroes Tour**" que es utilizado por **Angular** como proyecto para enseñar sus características.

Este proyecto se puede encontrar en la siguiente dirección:

https://angular.io/tutorial

Stackblitz: https://stackblitz.com/angular/vvqadvblooo

Código fuente: https://angular.io/generated/zips/toh-pt6/toh-pt6.zip

Objetivos

Los objetivos que se deben conseguir con esta práctica son:

- Entender la pirámide de test de cualquier aplicación software
- Configura un entorno de test con **Jasmine/Karma**
- Introducción a los test con **Angular**

Formato y fecha de entrega

Se entregará todo el proyecto comprimido en formato **.zip** sin incluir el directorio **"node_modules"**. La fecha de entrega se mostrará en el aula de la asignatura.

Un poco de teoría

Una vez hemos hecho el desarrollo de cualquier aplicación, se pasa a la etapa de testing/pruebas (a menos que estemos en una metodología ágil en la cual los test son parte de cada **sprint**, posteriormente haremos un breve comentario de qué es un **sprint**).

Hacemos un apunte comentando que la metodología más extendida a nivel de desarrollo del software podríamos decir que es **SCRUM**, aquí podéis encontrar la guía en su última versión actualizada en español:

https://www.scrumguides.org/download.html

Si trabajáis en equipos de desarrollo, seguramente utilizaréis esta metodología ágil, en caso contrario, os animo a que os documentéis un poco sobre esta manera de trabajar ya que las metodologías ágiles son más productivas que las clásicas.

Pero como ya hemos dicho anteriormente, durante el desarrollo también se hacen pruebas locales (y no tan locales).

¿Qué tipos de test nos podemos encontrar?

Los test los podemos ver desde dos perspectivas:

a) funcionalidad del test.
b) quién ejecuta el test.

A) Funcionalidad del test

En esta categoría veremos dos tipos: los test funcionales y los no funcionales.

- **Test funcional**: en este tipo de pruebas se comprueba que el software cumple las funciones por las que se ha construido. Por ejemplo, en una calculadora con la operación de sumar, se debe comprobar que la operación sumar se lleva a cabo correctamente.

- **Test no-funcional**: este tipo de test comprueba otros elementos importantes en un software, p.ej. seguridad, escalabilidad, volumen de datos a manejar o a cargar, niveles mínimos aceptables de rendimiento, etc. La comprobación de algunos de estos elementos da nombre al test, como, por ejemplo:

    - **Test de estrés**: busca forzar un fallo del software a través del consumo excesivo de sus recursos (p.ej. espacio en disco, memoria, etc.).

    - **Test de carga**: mira la concurrencia máxima, es decir, la capacidad máxima que tiene el software para atender a un conjunto de usuarios de manera simultánea. Por ejemplo, así como posteriormente veremos cómo implementaremos nuestros propios test unitarios, en este caso de los test de **carga**, existen herramientas que nos facilitan como hacerlos. Por ejemplo, estamos desarrollando una **webapp** de reservas de actividades turísticas, vamos a salir a producción y queremos validar que nuestra plataforma nos 'aguante' sin caerse y queremos saber qué número de usuarios concurrentes nos aguanta nuestro servidor. Pues existen plataformas donde a grandes rasgos, tú les pasas tu **url** y te genera un

pequeño **token** que simplemente tienes que añadir a tu fichero **index.html** y luego des de un panel de administración de dicha plataforma puedes simular la conexión concurrente de 100, 1000, 10000, … usuarios y ver los tiempos de respuesta. Esto sería un ejemplo.

- **Test de rendimiento (performance)**: comprueba que el tiempo de respuesta del software a una petición sea aceptable.

B) Quién ejecuta el test

En esta categoría podemos clasificar los test en si los ejecuta el cliente o el equipo de desarrollador. En el primer caso, al test se le llama test de **aceptación** o de cliente, mientras que, en el segundo caso, existen tres tipos de test:

- test unitario.
- test de integración.
- test de sistemas.

Test realizado por el cliente (conocido como test de aceptación/cliente)

En una metodología en cascada (clásica), a grandes rasgos, al cliente se le ve al principio, para saber los requisitos, y al final, es decir, para entregarle el producto.

Esta no es la mejor manera de trabajar, y de ahí que salieron las metodologías ágiles.

En otras metodologías, como las ágiles (donde se incluye **SCRUM**), al cliente se le muestra el estado del producto cada cierto tiempo, p.ej. cada 15 días o cada mes.

Normalmente se hacen **sprints** de **SCRUM** de 15 días, donde lo que se implemente en estas dos semanas es totalmente funcional y está en producción.

Un **sprint** es un período de tiempo que suele ser de dos semanas donde se define un objetivo (un conjunto de tareas a desarrollar) y se deben tener implementadas, probadas y desplegadas a producción al final del **sprint** para enseñar los avances al cliente.

Cuando el cliente define el test (que lo puede hacer antes de que el equipo desarrollador escriba una sola línea de código), éste se usa para que el cliente valide el software. Por ello a dicho test se le llama test de aceptación o de cliente. Así pues, quien ejecuta el test es el cliente.

El test de aceptación/cliente es un test que permite comprobar que el software cumple con un requisito de negocio. Así pues, un test de aceptación describe, en lenguaje natural, un escenario/ejemplo que debe cumplir el software desde la perspectiva del cliente. Estos ejemplos/test son consensuados con el cliente. Esto implica que al final del proyecto se le presenta al cliente el resultado de dichas pruebas que, si son favorables, obligan al cliente a dar por válido el software y, por ende, por cerrado el proyecto.

Ejemplos de test de aceptación:

- El producto 045 con precio 35€ al aplicarle el IVA del 21% su precio final es de 42,35€.
- Cuando la facturación llegue a 21.000€ se debe enviar un e-mail al encargado de facturas.

- La web debe ser compatible con tabletas.
- La web realizada debe cargarse en menos de 1 segundo.
- La web debe permitir 1.000.000 de usuarios concurrentes.

Como se puede ver, un test de aceptación puede ser un test funcional (un ejemplo de este tipo está en verde) o no funcional (un ejemplo de test de carga está en rojo, otros ejemplos serían: seguridad, escalabilidad, volumen de datos a manejar, niveles mínimos aceptables de rendimiento, etc.).

Test realizado por el equipo desarrollador

Durante el desarrollo los programadores hacen, principalmente, tres tipos de test:

- **Test unitarios (unit test)**: son las pruebas de más bajo nivel en la programación. Con ellas se comprueba el funcionamiento de las unidades lógicas independientes, normalmente los métodos. Son pruebas muy rápidas de llevar a cabo y son las que los desarrolladores están constantemente pasando para verificar que su software funciona adecuadamente.
- **Test de integración**: con estas pruebas se llevan a cabo comprobaciones de varias unidades de software diferentes. En este tipo de pruebas se hacen comprobaciones de que diferentes unidades lógicas funcionan adecuadamente entre sí. Normalmente se llevan a cabo comprobaciones del paso de mensajes entre estas unidades lógicas. Por ejemplo, un método que llama a otro; un objeto que interactúa con otro, etc.
- **Test de sistema**: una vez que se han comprobado las unidades lógicas de software mediante test unitarios y de integración, se comprueba la aplicación completa en un entorno de desarrollo similar al de producción.

Test unitario

En esta Práctica nos centraremos en los test unitarios. Estos son los test más importantes a nivel de desarrollo. Cada test unitario es un paso que andamos en el camino de la implementación correcta del software. Los desarrolladores utilizan los test unitarios para asegurarse de que el código funciona como esperan que funcione, al igual que el cliente usa los test de aceptación/cliente para asegurarse de que los requisitos de negocio se cumplan con el software como se espera que lo haga. Todo test unitario debe ser:

- **Atómico**: significa que el test unitario prueba la mínima cantidad de funcionalidad posible. Esto es, probará un único comportamiento de un método de una clase. Como ya sabes, un método puede presentar distintas respuestas ante distintas entradas o distinto contexto. El test unitario se ocupará exclusivamente de uno de esos comportamientos, es decir, de un único camino de ejecución.
- **Independiente**: significa que un test no puede depender de otros para producir un resultado satisfactorio. No puede ser parte de una secuencia de test que se deba ejecutar en un determinado orden. Debe funcionar siempre igual independientemente de que se ejecuten otros test o no.

- **Inocuo**: esta propiedad significa que no altera el estado del sistema. Al ejecutarlo una vez, produce exactamente el mismo resultado que al ejecutarlo veinte veces. Además, no altera la base de datos, ni envía e-mails ni borra ficheros, ni los crea (y si los tiene que crear, los debe borrar). Un test, una vez ejecutado, debe comportarse como si no se hubiera ejecutado.

- **Rápido**: puesto que ejecutamos un gran número de test cada pocos minutos, resultaría muy poco productivo tener que esperar unos cuantos segundos cada vez. Un único test tiene que ejecutarse en una pequeña fracción de segundo.

Enunciado

Ejercicio 1 – Repaso de los tipos de test

Lee el artículo publicado en la Web de Martin Fowler (https://martinfowler.com/articles/practical-test-pyramid.html) y junto a la explicación previa debes contestar a las siguientes preguntas:

a) Realiza un esquema de los diferentes tipos de test que existen, y explica en una frase corta qué testean según la categoría. (1 punto)

b) Explica qué es un "test double" y por qué son importantes a la hora de desarrollar test. (0.5 puntos)

c) Según nuestro proyecto qué deberían testear (diagramas de nodos, igual que en el artículo) nuestros test según las categorías: (1 punto)

- Unitarios.
- Integración.
- UI Test.
- E2E Test.
- Aceptación.

Ejercicio 2 – Configuración

Siguiendo los pasos en la web oficial de **Angular**:

https://angular.io/guide/testing#setup

se muestra como configurar el entorno de test. Sin necesidad de realizar ningún cambio ejecuta el entorno de test de **Angular** con un test básico que podría ser:

expect(true).toBe(true);

Para poder comprobar que el entorno está satisfactoriamente configurado.

Para realizar la validación del entorno de test anterior podríamos crearnos un **app.component.spect.ts** y en este fichero añadir la instrucción anterior dentro de un **describe/it**. Con esto sería suficiente.

Lo que buscamos es que con el proyecto podamos hacer un **ng test** y nos valide el test anterior.

NOTA: Primer consejo es comentar o eliminar todos los ficheros .spec que tengas en el proyecto puesto que al no haber realizado ningún test previamente y haberse generado utilizando Angular-CLI o copiando de diferentes carpetas pueden no superarse los test y complicar el comienzo de la práctica.

Realizar cambios en la configuración del entorno de test de modo que los reportes sean al estilo de Mocha (otro test runner) que permite una visualización del flujo del test más amigable para las personas. Para realizar esta configuración sólo debes seguir el siguiente manual (https://juristr.com/blog/2018/02/add-mocha-reporter-angular-cli-tests/)

Ejercicio 3 – Primer test sobre servicios

Siguiendo la documentación oficial podemos ver que se construyen test que permiten comprobar la información que existe en los servicios.

En el proyecto del **Tour de Heroes** tenemos un servicio muy sencillo denominado "**message.service.ts**" el cual contiene una lista de mensajes.

Este servicio está compuesto por dos métodos: **add** y **clear**.

Debemos crear nuestro fichero **message.services.spec.ts** para realizar las siguientes comprobaciones sobre los métodos **add** y **clear**

- **add**
  - Se agrega un nuevo **message** al array cuando se invoca (es decir, la longitud del array crece y el valor enviado como parámetro es el último de la lista).
- **Clear**
  - Se borra el contenido del array cuando no contiene mensajes la lista.
  - Se borra el contenido del array cuando contiene varios mensajes.

Ejercicio 4 – Test de HeroService

Siguiendo la documentación oficial observa que hay un ejemplo de cómo hacer test sobre **HeroService.ts** y **observables**. Amplia dicho fichero de test con los test a los siguientes métodos:

- getHeroNo404
- getHero
- addHero
- deleteHero
- updateHero

Para realizar este ejercicio nos crearemos el fichero **hero.service.spect.ts**.

Ejercicio 5 – Primer test de un componente

Siguiendo la documentación oficial podemos ver que se construyen test que nos permiten comprobar que una página tiene un título y que se invocan a diferentes eventos, tal y como un **click** de un botón (https://angular.io/guide/testing-components-basics).

En este primer contacto con test de componentes vamos a dirigirnos al componente **heroes.component** y nos crearemos el fichero **heroes.component.spec.ts**.

Realiza los siguientes test en este nuevo fichero:

- Existe un h2 con el texto "**My Heroes**".
- Simula la invocación del botón haciendo **click** y comprueba que se invoca el método **add**.

Ejercicio 6 - Test de componentes de la aplicación

Ampliando el **coverage** de los test que has realizado.

Elige dos clases del proyecto **hero** (servicios o componentes) y realiza test sobre sus métodos o aspecto visual.

Hay que realizar al menos tres validaciones (test) de cada clase.

No es válido escoger una clase que no tiene métodos ni aspecto visual.

Puntuación

A continuación, mostramos cuánto puntúan cada uno de los apartados de la práctica para obtener la nota final de la misma.

- Ejercicio 1**[2,5 puntos]**
- Ejercicio 2**[1 punto]**
- Ejercicio 3**[1,5 puntos]**
- Ejercicio 4**[2,5 puntos]**
- Ejercicio 5**[1 punto]**
- Ejercicio 6**[1,5 puntos]**