

ENUNCIADO PEC 4

Testing, PWA y control flow

Desarrollo front-end avanzado

**Máster Universitario en Desarrollo de
sitios y aplicaciones web**

UOC

Universitat Oberta
de Catalunya

Contenido

- Formato de entrega
- Enunciado
- Puntuación



Formato de entrega

Se entregarán cada uno de los dos proyectos comprimidos en formato **.zip** sin incluir el directorio “**node_modules**”.

IMPORTANTE

El proyecto 1 de TESTING se resolverá a partir de la solución oficial de la práctica 2, concretamente el ejercicio donde tenemos todas las llamadas a la api implementadas con observables.

El proyecto 1 se acompañará de un documento con una captura de pantalla del resultado de la ejecución de los test y cualquier información relevante sobre el desarrollo o estado de la entrega.

El proyecto 2 de PWA es un proyecto nuevo, independiente, que no depende de soluciones anteriores y se deberá desarrollar en la versión 19 de Angular.

El proyecto 2 se acompañará de un documento con la URL de la aplicación desplegada y cualquier información relevante sobre el desarrollo o estado de la entrega.

Enunciado Proyecto 1 de TESTING

Una vez estudiados los ejemplos de testing del documento **Teoria-Tema-4-TESTING_es**, podremos empezar a trabajar en la resolución de este primer proyecto.

Pasos previos:

- Lo primero que tendremos que hacer es eliminar todos los ficheros de test (.spec.ts) que se han creado automáticamente en el proyecto
- Posteriormente, añadir en el fichero **angular.json** en el apartado **test.options** la línea:

```
"codeCoverage": true,
```

Ejercicio 1 – test pipe

Implementar el test del pipe que tenemos en nuestro proyecto el cual formatea el formato de salida de una fecha en función del parámetro que le pasemos.

Tendremos que crear el fichero **format-date.pipe.spec.ts** e implementar los siguientes test:

- Que se crea el pipe correctamente
- Dada una fecha y el argumento 1 devuelve el formato esperado
- Dada una fecha y el argumento 2 devuelve el formato esperado
- Dada una fecha y el argumento 3 devuelve el formato esperado
- Dada una fecha y el argumento 4 devuelve el formato esperado

Ejercicio 2 – test rutas

Implementar los siguientes pasos:

- refactorizar el componente **header.component** de la siguiente manera:
 - En lugar de tener por cada punto de menú un método **home()**, **login()**, **register()**, **adminPosts()**, ... y que cada uno llame a un **router.navigateByUrl** con la url determinada, vamos a implementar un único método **navigationTo** en el que le pasaremos por argumento la ruta destino.

- Adaptaremos la vista **header.component.html** de manera que todos los botones llamarán al método **navigationTo** pero cada uno le pasará por argumento la ruta destino que toque, **home**, **login**, **register**, ...
- Una vez refactorizado el componente, tendremos que implementar los siguientes test:
 - Que el componente se crea correctamente
 - Testear que la navegación es correcta
 - Por cada posible ruta (home, login, register, posts, categories y profile) de las que gestiona el **header.component** validar que se lanza el **navigateByUrl** con el argumento correcto.
 - Ejemplo de ruta **home**:
 - component.navigationTo('home');
 - expect(spy).toHaveBeenCalledWith('home');

Ejercicio 3 – test servicios

Implementar el test de los siguientes servicios:

- CategoryService
- PostService

De cada servicio tendremos que:

- implementar el test de que el servicio se crea correctamente
- implementar el test de cada función que implemente el servicio
 - por cada función, a parte de testear que el resultado es el esperado, deberemos testear que el tipo de llamada es la esperada (get, put, post o delete)

Ejercicio 4 – test componentes

Implementar los siguientes test de componentes:

- en el componente **CategoriesListComponent**
 - un test que valide que se crea el componente correctamente
 - un test que nos asegure dos cosas:

- que cuando se lance el **loadCategories** se lance la llamada **getCategoriesByUserId** del servicio
 - que la respuesta de la llamada sea la esperada
- un test que valide que se lanza el **navigateByUrl** con el argumento correcto cuando creamos una categoría
- un test que valide que se lanza el **navigateByUrl** con el argumento correcto cuando actualicemos una categoría
- en el componente **PostsListComponent**
 - un test que valide que se crea el componente correctamente
 - un test que nos asegure dos cosas:
 - que cuando se lance el **loadPosts** se lance la llamada **getPostsByUserId** del servicio
 - que la respuesta de la llamada sea la esperada
 - un test que valide que se lanza el **navigateByUrl** con el argumento correcto cuando creamos un post
 - un test que valide que se lanza el **navigateByUrl** con el argumento correcto cuando actualicemos un post

Ejercicio 5 – test vista

Implementar los siguientes test de componentes para validar el comportamiento de una vista:

- cuando estamos autenticados que existan los puntos de menú home, admin posts, admins categories, profile y logout
- cuando no estemos autenticados que existan los puntos de menú home, login y register.

Puntuación proyecto 1 testing

A continuación, mostramos cuánto puntúan cada uno de los apartados del proyecto:

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

Enunciado Proyecto 2 de PWA

Una vez estudiado el caso de uso del documento **Teoria-Tema-4-PWA_es**, podremos empezar a trabajar en la resolución de este segundo proyecto.

Ejercicio 1 – Implementar una PWA

En este primer ejercicio deberemos implementar una aplicación completa libre y deberá transformarse a **PWA**.

Podemos seguir los pasos de los materiales del documento **Teoria-Tema-4-PWA_es**. La idea sería buscar una api pública que podamos consultar, que devuelva un listado de alguna entidad, e implementar una aplicación lista-detalle tipo hemos estudiado en la teoría (con la api de imágenes) y una vez hecha esta implementación pasarla a **PWA**.

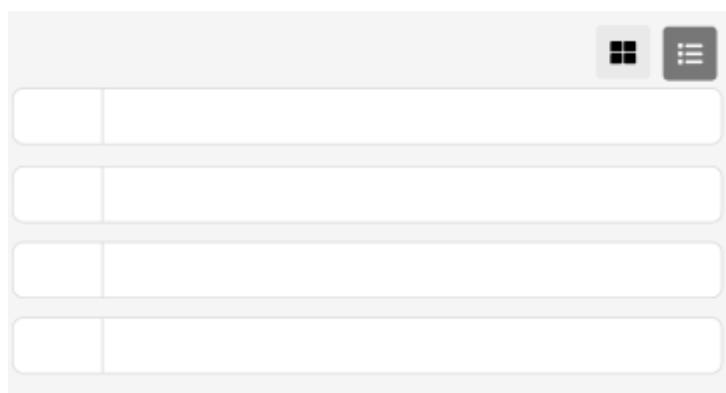
Intentad buscar una api de alguna temática que os resulte interesante, intentad evitar las típicas apis de PokemonAPI o RickAndMortyAPI.

Deberemos configurar el **manifest** y todo lo necesario del **service worker** para que pueda instalarse y funcionar sin internet.

Deberéis pensar en qué estrategias de caché seguir en función de los recursos que tengáis.

La aplicación deberá tener al menos:

- Component-list:
 - Una **home** donde aparezca el listado de la entidad devuelta por la api escogida utilizando una **tabla** o un listado de **cards** de **Angular Material**.
 - Haremos la siguiente implementación:



En la parte superior derecha, implementaremos dos botones para intercambiar la vista “**cards**” de la vista “**tabla**”.

- Mientras esperamos el listado mostraremos el **spinner** de **Angular Material**
- La aparición del listado la haremos mediante alguna animación (elección libre)
- Al hacer clic a un elemento de la lista navegaremos a la página de detalle

- Component-detail:
 - Página donde mostraremos los detalles de la entidad.
 - Esta pantalla va a depender de cada api utilizada. Sería importante que la api devuelva algún tipo de imagen para poder comprobar cómo se cachea.
 - En la parte superior mostraremos solo un campo de la entidad, el nombre o el título (una propiedad que sea representativa de la entidad).
 - A la derecha mostraremos el botón de “back” para poder volver a la **home**
 - Debajo mostraremos un botón “**show all details**”, al pulsar el botón:
 - Se mostrarán el resto de las propiedades de la entidad
 - Para mostrar el resto de las propiedades deberemos utilizar al menos uno de los siguientes elementos de **Angular Material**:
 - **Tree**
 - **Tabs**
 - **Expansion panel**
 - **Progress bar**
 - **Slider**
 - *Si vemos que en la respuesta de la api nos falta información para poder utilizar alguno de los componentes anteriores nos podemos inventar algunas propiedades “añadiéndoles” en el array de respuesta “manualmente”.*

Implementación de componentes base

Tendremos que implementar al menos los siguientes componentes base:

Componente **card** para reaprovechar en el listado de **cards** de la **home**. Al hacer clic en la **card** iríamos al detalle de la entidad.

Componente **grid** para reaprovechar en el listado de la **home** para cuando pulsemos el botón “**modo tabla**”. Al hacer clic a la fila iríamos al detalle de la entidad.

Podéis implementar más componentes base en función de vuestros requisitos.

Importante

En este desarrollo deberéis utilizar el nuevo control flow de Angular, es decir, utilizar:

- **@if** en lugar de ***nflf**
- **@for** en lugar de ***ngFor**

Ejercicio 2 – Desplegar una aplicación Angular en GitHub

Una vez tenemos la aplicación **Angular** finalizada, debemos subirla a un servidor remoto en modo producción.

Seguid los pasos del apartado 5 del documento de teoría **Teoria-Tema-4-PWA_es** para ver como desplegar vuestra aplicación. En el caso la teoría, se propone desplegarlo en Github, es relativamente sencillo.

Hay otras plataformas como Netlify (<https://www.netlify.com/>) que también podéis utilizarlas para subir vuestro proyecto.

Añadid un documento con la **url** resultante para poder revisar la aplicación desplegada.

Puntuación proyecto 2 PWA

A continuación, mostramos cuánto puntuán cada uno de los apartados del proyecto:

- Ejercicio 1[**8 puntos**]
- Ejercicio 2[**2 puntos**]

Nota final de la práctica 4

La nota final de la práctica 4 saldrá como resultado de hacer la media entre la nota del proyecto 1 y 2.