EJERCICIO 5 ANGULAR EVENTOS

PARTE 1.

Una aplicación web puede dividirse en dos grandes bloques: **Front-end** y **Back-end**. El Front-end es la parte que se muestra a través del navegador web, o sea, la parte que se encarga de interactuar con el usuario para recoger datos e información. El Back-end, por su parte, se ejecuta en un servidor y recoge a través de una API de servicios toda esa información para gestionarla (guardarla en base de datos, realizar comprobaciones…).

Angular, por su parte, nos ofrece el servicio HttpClient. **HttpClient** es un servicio que proporciona una API simplificada, construida sobre la interfaz XMLHttpRequest, para realizar peticiones HTTP y procesar sus respuestas. En este sentido, el servicio ofrece toda una serie de métodos para realizar los distintos tipos de peticiones HTTP que existen:

Get, Post, Put, Delete, Patch, Head y Json.

HttpClient se integra en el **módulo HttpClientModule** del paquete **@angular/ common/http.**

Los **observables de las librerías RxJS** (reactive ex- tensions for JavaScript) son otra herramienta que nos permitirá programación asíncrona. Veamos cuáles son las principales diferencias respecto a las **promesas**:

- Una promesa siempre devolverá un valor o un error, mientras que un observable puede emitir múltiples valores en el tiempo. Por este motivo, a veces se le define como un flujo de datos o “stream”.

- Una promesa no puede cancelarse, en cambio un **observable** sí. Si el valor esperado por una promesa (el resultado de una llamada a un servicio) ya no es necesario, no hay manera de cancelar esa promesa. Siempre acabará llamándose a su “callback” de éxito o rechazo. En un observable no ocurre lo mismo, ya que siempre podemos anularlo.

La función empieza a ejecutarse de forma asíncrona cuando un observador realiza una **suscripción** en el observable. Esta suscripción se traduce en una ejecución al método "**subscribe**(observer)" del observable. En el caso que se realizara otra suscripción al mismo observable, se pondría en marcha otro hilo de ejecución asíncrona distinta.

Funcionamiento de los operadores intermedios:

=> **Map** → Obtiene el dato devuelto por el Observable (o el operador anterior si concatenamos varios), aplicamos alguna transformación al dato y la devolvemos (el método devolverá Observable<DatoTransformado>).

=> **Tap** → Se utiliza normalmente para operaciones de depuración (mostrar datos por consola). Devuelve un observable con el dato que recibe sin modificar nada.

=> **Filter** → Filtra los datos que devuelve el observable y solo deja pasar aquellos que cumplan la condición.

Funcionamiento de los parámetros de **subscribe**:

Estos 3 parámetros deben ser funciones.

(result) => Recibe los datos del observable.

(error) => Esta función es llamada si se produce algún error en el observable y se encarga de recibir el error.

() => Esta función se ejecuta siempre al final y no recibe ningún parámetro.

Siempre debemos **suscribirnos a un observable** para que se ejecute su código interno y así pueda empezar a emitir los datos.