IMPLEMENTACIÓN

Se debe comentar tanto el MVC como el Observer que incorpora Angular, que te suscribes a los servicios que quieres, la vista se renderiza y cuando tienes los datos los saca por pantalla

Comentar que hubo problemas a la hora de pasar entre componentes información, ya que la vista del componente hijo se renderizaba antes de tener la información necesaria del padre para renderizar dicha vista (el meter del contador), se solucionó al pasarle el meter desde el listado de contadores al componente padre y buscando con ese meter las medidas necesarias del contador al api de itosens (creo que fue así, mirar el código). El problema de ello es las dependencias que se generaron, aunque cierto es que siempre que se acceda a esa url, lo normal es recoger el id y el meter del contador.

ESTRUCTURA

Librerías externas que se usaron: transloco, ngx-charts (no se usó al final), leaflet (mapa), chart.js, fortawesome-angular

Decir que angular utiliza más un MVVM porque la vista y el modelo están fuertemente ligados

Para la implementación del ..

La figura X muestra la estructura básica de la aplicación, la cuafksjfsd l frontend y backend

Para el frontend, como se ha mencionado en la sección X, se utilizó el framework de Angular. Esto fue debido a su comodidad para assas

El framework Angular permite una separación muy cómoda entre la parte más cercana al usuario, que es una vista pasiva (

4.2 ESTRUCTURA DEL FRONTEND

La estructura del *frontend* de la aplicación está dividida principalmente en componentes, servicios y modelos, aunque antes de entrar en profundidad a explicarlos, se deben mencionar otros elementos que también la conforman.

Como se observa en la figura X, hay varios archivos de configuración situados en el directorio raíz del *frontend* que contienen metadatos del proyecto, así como las dependencias que utiliza. Estos archivos van ligados con la carpeta *node\_modules*, en la cual se encuentran las librerías instaladas, incluyendo las externas. A su vez, se pueden visualizar los ficheros de inicio de la aplicación y el fichero *styles.scss,* en el que se especifican los estilos comunes de la interfaz de usuario, como la tipografía, el color y forma de los botones, etc.

Otro directorio que destacar es *assets*, en el cual están almacenados los archivos que contienen las traducciones de las vistas. Para acceder a ellos se utiliza un servicio de la librería *Transloco*, el cual se puede llamar tanto desde la vista como del controlador de un componente. Hay un ejemplo de su uso al principio del código mostrado en la figura X.

Por otra parte, encontramos dentro del directorio *app* los ficheros de enrutado de los componentes y su jerarquía. Esto es importante para poder especificar, por ejemplo, que la barra de navegación debe de estar presente en todas las vistas a excepción del *Login*. Este directorio también contiene otros directorios, como *validators*, el cual almacena los validadores que se usan para verificar los datos de los formularios, o *interceptors*, para mostrar las vistas en función de los permisos que tengan los usuarios.

Con respecto a los componentes, cada uno de ellos está formado por dos ficheros. El primero (véase figura X) es un fichero de *html* que contiene únicamente lo elementos que se muestran al usuario y su formato, es decir, se trata de una vista completamente pasiva. El segundo fichero (véase figura X) tiene la extensión *ts* y contiene el código que se encarga de pedir la información necesaria que debe mostrarse por pantalla. Este fichero contiene una clase en *Typescript* con la anotación *@Component*. A diferencia de otros *frameworks* como *React*, *Angular* permite hacer este tipo de separación entre vista y controlador para facilitar su implementación, ya que así se evita juntar ambas partes y, por tanto, ambos lenguajes en un mismo fichero, lo que podría resultar un tanto lioso.

Asimismo, se debe tener en cuenta que una determinada vista de la interfaz del usuario puede estar formada por varios componentes. En la aplicación, el componente que muestra una tabla con el listado de alarmas que tiene un contador lo engloba un componente padre que contiene los datos básicos de dicho contador y sus medidas. Al final del código mostrado en la figura X se puede ver un ejemplo de ello, incluyendo el paso de parámetros y propiedades necesarios para que ambos componentes puedan comunicarse. Este tema está mejor desarrollado en el apartado X, el cual explica los problemas que hubo durante la implementación.

Continuando con la descripción del componente, en el fichero *blabla.ts* (véase figura X), se puede observar que implementa la interfaz *OnInit*, que sirve para indicar qué servicios o métodos se quieren ejecutar al renderizar la vista del componente. También se debe considerar cómo realiza la llamada al método de un servicio, ya que realmente se suscribe a él para ser notificado cuando le lleguen los datos y, mientras tanto, se ejecute paralelamente el resto del programa. Esto es así para optimizar su ejecución, ya que si se quiere hacer más de una consulta a la base de datos sería tedioso tener que esperar a que acabaran todas ellas para cargar la vista. Además, como se muestra en la figura X, se puede indicar cuándo visualizar un determinado elemento usando *\*ngIf*, puesto que consultará una variable booleana que cambiará una vez se reciban los datos. En la sección X se explica con más detenimiento los *observables* de Angular y el patrón que implementan.

Por otro lado, están los servicios, que son los que se comunican directamente con el *backend* de la aplicación y en ellos están definidas las llamadas necesarias para tal propósito. Pueden ser utilizados por diferentes componentes y cada servicio se encarga de la gestión de un determinado tipo de datos, como las medidas de un contador, las alarmas o la información del usuario. La figura X hace referencia un fragmento de la clase *AdditionalDataWatermeterService.ts*, en el cual se puede observar los dos tipos de llamadas que realiza para obtener la información adicional de un contador y para modificarla. Como se puede observar, no es necesario indicar la *url(poner un pie de página si no se ha explicado antes qué es una url)* completa, ésta se puede especificar en los archivos de configuración.

En referencia a los modelos, éstos son clases en *Typscript* que dan forma a los datos provenientes del *backend*. La figura X muestra la clase *Watermeter*, correspondiente a los datos básicos que tiene un contador. Para que la información se almacene correctamente, es necesario que la clase tenga los mismos atributos y se llamen igual que las clases definidas en el modelo del *backend* (véase figura X del apartado X).

PATRONES

Observer y MVC

PROBLEMAS

Lo de pasar de padre a hijo la info que venía del backend

Además, puede darse el caso de que varios componentes compartan algún fichero con la extensión *scss*, el cual indica que dicho grupo de componentes tienen un determinado tipo de características estéticas comunes, como podría ser el color y tamaño de ciertos paneles. No obstante, como ya se ha comentado en el apartado X, el estilo general usado en la aplicación se encuentra en el archivo *styles.scss*, el cual es importado por todos los componentes.

Para entender mejor el funcionamiento de cada componente y de cómo

Otras características del componente que se encuentran en el fichero de Typescript son

No obstante, no es algo común, ya que lo más conveniente es extrapolar a un lugar común todos los elementos y estilos que sean iguales, los cuales se vayan a usar por varios componentes. Esto tiene como finalidad evitar la producción de código repetido, el cual es susceptible a errores en caso de que se produzca algún cambio que obligue a una modificación del código, ya que probablemente se tenga que modificar en más de un fichero. Pasa lo mismo con los servicios, puesto que realmente las llamadas al *backend* podrían hacerse desde el propio componente.