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 DE LA APLICACIÓN

La estructura general de la aplicación se muestra en la figura X-estructura. La parte del \textit{*frontend}* se implementó con el \textit{*framework}* de *Angular*, el cual se explica con más detenimiento en la sección X. Además, se usaron varias librerías externas, como \textit{*Transloco}* para la traducción de los textos mostrados en la interfaz de usuario, \textit{*Chart.js}* para las gráficas, \textit{*Leaflet}* para la visualización de los mapas y \textit{*Fortawsome-Angular}* para los iconos.

Por el contrario, el directorio con el nombre de *java* contiene la parte del \textit{*backend*} de la aplicación. En su implementación se utilizó el \textit{*framework}* \textit{*Spring}*, el cual también se expone en la sección X.

FIGURA ESTRUCTURA GENERAL

Por último, el directorio \textit{*resources}* contiene las sentencias de creación de la base de datos, así como varias inserciones por defecto con el fin de mantener una consistencia de datos mínima. Esto implica que si, por ejemplo, se borrara manualmente de la base de datos la tabla \textit{*AdditionalWatermeterData}* que aparece en la figura X-bbdd, ésta se volvería a crear al ejecutarse la aplicación y añadiría los valores por defecto estipulados para cada contador. En caso de que los valores se fueran actualizando, esta sentencia no se ejecutaría, ya que no sería necesaria.

FIGURA EJEMPLO DE SENTENCIA BBDD

4.2. ESTRUCTURA FRONTEND CORREGIDO

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

FIGURA ESTRUCTURA FRONTEND

Como se observa en la figura X-estructura-front, hay varios archivos de configuración situados en el directorio raíz del \textit{*frontend}* que contienen metadatos del proyecto, así como las dependencias que utiliza. Estos archivos van ligados con la carpeta \textit{*node\_modules}*, en la cual se encuentran las librerías instaladas. A su vez, se pueden visualizar los ficheros de inicio de la aplicación y el fichero \textit{*styles.scss}.* En este último 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 \textit{*assets}*, en que están almacenados los ficheros que contienen las traducciones de los textos que aparecen en las vistas. Para acceder a ellos se utiliza un servicio de la librería \textit{*Transloco}*, el cual se puede llamar tanto desde la vista como desde el controlador de un componente. Hay un ejemplo de su uso al principio del código mostrado en la figura X-html.

IMAGEN DE FICHERO HTML

Por otra parte, encontramos dentro del directorio \textit{*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 \textit{*Login}*. Esta carpeta también contiene otros directorios, como \textit{*validators}*, que almacena los validadores usados para verificar los datos de los formularios, o \textit{*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-html) es un fichero de \textit{*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-ts) tiene la extensión \textit{*ts}* y contiene el código que se encarga de pedir la información necesaria que debe mostrarse por pantalla. Este fichero contiene una clase en \textit{*Typescript}* con la anotación \textit{*@Component}*. A diferencia de otros \textit{*frameworks}* como \textit{*React}*, \textit{*Angular}* permite hacer este tipo de separación entre vista y controlador, facilitando así su implementación.

IMAGEN FICHERO TS DEL COMPONENTE

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-html se puede ver un ejemplo de ello, incluyendo el paso de parámetros y propiedades necesarios para que ambos componentes puedan comunicarse. Además, se puede observar el mecanismo de comunicación entre el componente padre e hijos en la figura X-ts-measures. Este tema está desarrollado en el apartado X, el cual explica los problemas que hubo durante la implementación del \textit{*frontend*}.

Otra característica a destacar del componente es la implementación de la interfaz \textit{*OnInit} (véase figura X-ts-watermeter)*, que sirve para indicar qué 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. En otras palabras, los servicios implementan el patrón \*textit{Observer*} a través del uso de los \textit{*observables}* de \textit{*Angular*}. Además, como se muestra en la figura X-html, es posible indicar cuándo se debe visualizar determinados elementos de la vista usando \textit{*\*ngIf}*, puesto que consulta una variable booleana que cambia una vez se reciben los datos.

Por otro lado, están los servicios, que son los que se comunican directamente con el \textit{*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-service hace referencia un fragmento de la clase \textit{*AdditionalDataWatermeterService.ts}*, en el cual se puede ver los dos tipos de llamadas: la que realiza para obtener la información adicional de un contador y la que actualiza los datos. Como se puede observar, no es necesario indicar la \textit{*URL} (*localizador uniforme de recursos, en inglés\textit{*Uniform Resource Identifier})(poner un pie de página si no se ha explicado antes qué es una url)* completa, ésta se puede especificar en los ficheros de configuración.

IMAGEN SERVICE FRONTEND

En referencia a los modelos, éstos son clases en \textit{*Typscript}* que ayudan a dar forma a los datos provenientes del \textit{*backend*}. La figura X-model.ts muestra la clase \textit{*Watermeter.ts}*, 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 \textit{*backend*} (véase figura X del apartado X, VER SI ES NECESARIO AÑADIR LA OTRA FOTO).

IMAGEN DE WATERMETER

Finalmente, cabe destacar el uso del patrón MVVM (decir las siglas?) en la implementación del *frontend*, el cual es una variación del patrón clásico MVC (modelo-vista-controlador, o en inglés *model-view-controller*). Esto es así debido a la sincronización de datos que tiene el framework Angular, el cual implica una dependencia entre la vista y el modelo \cite{bib:wiki}.

[referencia: <https://openwebinars.net/blog/que-patron-usa-angular-mvc-o-mvvm/>].

**PROBLEMAS – Detalles de la implementación durante el desarrollo (o algo así, mirar qué título vendría bien)**

Entre los problemas que surgieron durante la implementación del \textit{*frontend}*, el más destacable fue el paso de información entre componentes. Esto se debió a que, en algunos casos, se debía estimar si valía la pena crear una dependencia entre componente padre e hijo o hacer varias peticiones al \textit{*backend*}. En el caso de los componentes que mostraban las alarmas y las medidas de un contador, se optó por obtener los datos necesarios desde el componente padre, el cual contenía la información básica de dicho contador. Se llegó a esa conclusión por el hecho de estar fuertemente relacionados entre sí, ya que todos ellos formaban parte de una única vista en la interfaz de usuario.

También se tomó la decisión de no calcular el consumo promedio total de todos los contadores del cliente al no ser un dato útil que consultar, a diferencia de lo que se mostraba en el prototipo inicial (véase figura x del anexo X).

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.

ESTRUCTURA BACKEND

La estructura del \textit{*backend*} de la aplicación se puede contemplar en la figura X-estructura-back, que muestra una serie de directorios entre los que podemos encontrar\textit{*services}* y \textit{*models}*, los cuales tienen el mismo nombre que en el \textit{*frontend}* (véase figura x-estructura-front). Ambos desempeñan funciones similares, aunque en este caso los servicios son más bien los intermediarios que hacen consultas a microservicios externos o usan los métodos de las clases que se encargan de pedir la información a la base de datos. Los servicios también están divididos según sus cometidos, son usados por los controladores y están marcados con la anotación \textit*{@Service*}.

FIGURA MODELO (si se llega a poner), mejor poner FIGURA ESTRUCTURA?

Por el contrario, los modelos, al igual que en \textit{*frontend}*, sirven para representar los datos obtenidos (véase figura x-model-watermeter.java). Si estos datos provienen de las bases de datos, su tratamiento y transformación se realizan en las clases que se encuentran en el directorio \textit{*persistence}*. Estas clases están marcadas con la anotación \textit*{@Repository*} y son las popularmente conocidas como DAOs (Objeto de Acceso a Datos, en inglés \textit{*Data Acess Object*}), ya que implementan este patrón de diseño. Parte de la implementación de una de ellas se puede examinar en la figura X-repository.

FIGURA DAO - repository

Por otro lado, en el directorio \textit{*rest}* se encuentra la implementación los controladores que mapean las peticiones \textit{*http}* al \textit{*backend*} de la aplicación. Estas clases están señaladas con la anotación \textit{*@RestController}* que proporciona el \textit{*framework Spring}* y contienen los servicios que van a utilizar. A su vez, estas clases utilizan otras anotaciones como \textit{*@RequestMapping}* para indicar la *ruta* común o \textit{*@GetMapping}* para especificar que es una petición de tipo \textit{GET}. La figura X-controller muestra un fragmento de la clase \textit{*WatermerController.java}*, donde se observa la estructura de estos controladores.

FIGURA CONTROLLER WATERMETER

Al describir estos elementos que conforman la estructura del \textit{*backend}*, es posible percatarse de la presencia del patrón compuesto \textit{*MVC*}. El \textit{*framework Spring}* facilita su implementación gracias a sus anotaciones, clases y métodos propios.

Finalmente, se debe mencionar el directorio \textit{*config}* que, como su nombre indica, contiene las configuraciones globales del \textit{*backend*} de la aplicación, así como las restricciones de seguridad, el acceso a la base de datos y a los microservicios. La configuración de acceso a estas últimas está representada en la figura X-config. En ella se puede comprobar cómo se obtienen las credenciales necesarias a través de la anotación \textit{*@Value}* con el fin de acceder a los microservicios utilizados para la obtención de medidas y alarmas de los contadores. También se puede observar cómo se crea una instancia de esos microservicios con la anotación \textit{*@Bean}*, que hace uso del patrón \textit{*Singleton}*.

FIGURA CONFIG

**PROBLEMAS -CAMBIAR TÍTULO (como en el front)**

Durante el desarrollo del \textit{*backend*}se optó por hacer uso de las expresiones lambda para simplificar la transformación de datos. Esto estuvo ligado a la decisión de hacer las llamadas a los microservicios directamente en los propios servicios del \textit{*backend*}, sin necesidad de utilizar ninguna clase intermediaria encargada de gestionarlas. Esto se debió a una cuestión de simplicidad, ya que sólo se requería llamar a los métodos del microservicio instanciado y construir el resultado obtenido mediante una expresión lambda sencilla. La figura X-service.java representa un fragmento de esta implementación.

FIGURA SERVICE DE ALARM

NOTA: HACER UN APARTADO QUE MUESTRE LOS RESULTADOS DE LA IMPLEMENTACIÓN, ES DECIR, QUE MUESTRE LAS VISTAS DE LA APLICACIÓN.