Facultad
de
Ciencias

Sistema web para la consulta interactiva de observaciones meteorológicas

(Web system for the interactive query of meteorological observations)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Fernando Arruti Samperio

Directora: Marta Zorrilla Pantaleón

Co-Director: Daniel San Martín Segura

Junio - 2018

Resumen

La Agencia Estatal de Meteorología (AEMET) dispone de una red de estaciones, atendidas por personal colaborador de la organización, en las que se recogen los datos diarios de temperaturas y precipitaciones.

Estas observaciones, aunque son de carácter público, su acceso no es directo y además el formato en el que se encuentran sus valores no se corresponde a ningún estándar.

Con objeto de resolver estas limitaciones, este proyecto ha diseñado e implementado una aplicación web que facilita la consulta y la visualización de estos datos.

Para desarrollar la solución se ha trabajado con el lenguaje de programación Python y la librería Pandas para adaptar los datos al estándar. Se ha creado un servicio REST que ofrece las funcionalidades de acceso a datos y una aplicación Web con Angular para implementar la interfaz de usuario.

Palabras clave: Meteorología, web interactiva, tratamiento de datos

Abstract

The Spanish State Meteorological Agency (AEMET) has a network of stations, which are attended by collaborators of the organization, who collect daily data of temperatures and rainfalls.

This agency has a historical record of the values collected in these stations available to the public, but its access is not direct and the format in which these values are stored does not conform to any standard.

That is why this project aimed at designing and implementing a web application, which facilitates the query and visualization of this data.

To develop this solution, we worked with the Python programming language and the Pandas library, to adapt the data to the standard. A REST service was created to offer the functionality of data access and a Web application with Angular was produced to implement the user interface.

Keywords: Meteorology, interactive web, data treatment

Índice

Índice de ilustraciones

1. Introducción

1.1. Antecedentes

La empresa PREDICTIA es una spin-off surgida de un grupo de minería de datos de la Universidad de Cantabria, cuyo ámbito de trabajo es el de la meteorología.

La información relativa a los datos meteorológicos es de interés general y existen repositorios con toda la información histórica recogida en todo el territorio nacional, pero su acceso y almacenamiento no está estandarizado.

Es por ello, por lo que la empresa PREDICTIA, con el fin de poner a disposición de la comunidad el acceso a la información histórica recogida por la red secundaria de AEMET [1], desea desarrollar una aplicación web que facilite la consulta y visualización de estos datos.

1.2. Objetivos

El objetivo del proyecto es, por tanto, la creación de una aplicación web interactiva que permita visualizar los datos recogidos por las estaciones de la red secundaria de AEMET.

En concreto, debe proveer las siguientes funcionalidades:

- Visualización de los datos recogidos en todo el territorio nacional sobre un mapa, que nos permita su filtrado según la variable recogida y el día; además de ajustar la resolución de estos datos a nuestro interés, ya sea diario, mensual o anual.

- Creación de gráficas dinámicas que permitan visualizar los datos seleccionados acorde a diferentes criterios.

- Visualización en formato tabla de los datos más significativos recogidos por las estaciones.

- Visualización de valores estadísticos para el periodo especificado.

1.3. Metodología y plan de trabajo

El desarrollo de este proyecto se ha realizado de forma iterativa incremental. Esta metodología consiste en dividir el proyecto en diversos bloques o iteraciones. Cada iteración provoca un resultado sobre el producto final, de manera que el cliente puede ver la evolución del proyecto de manera incremental.

Cada requisito se desarrolla en una única iteración, y en esta iteración se deben realizar todas las fases de desarrollo software (Análisis de requisitos, diseño, implementación, pruebas y documentación).

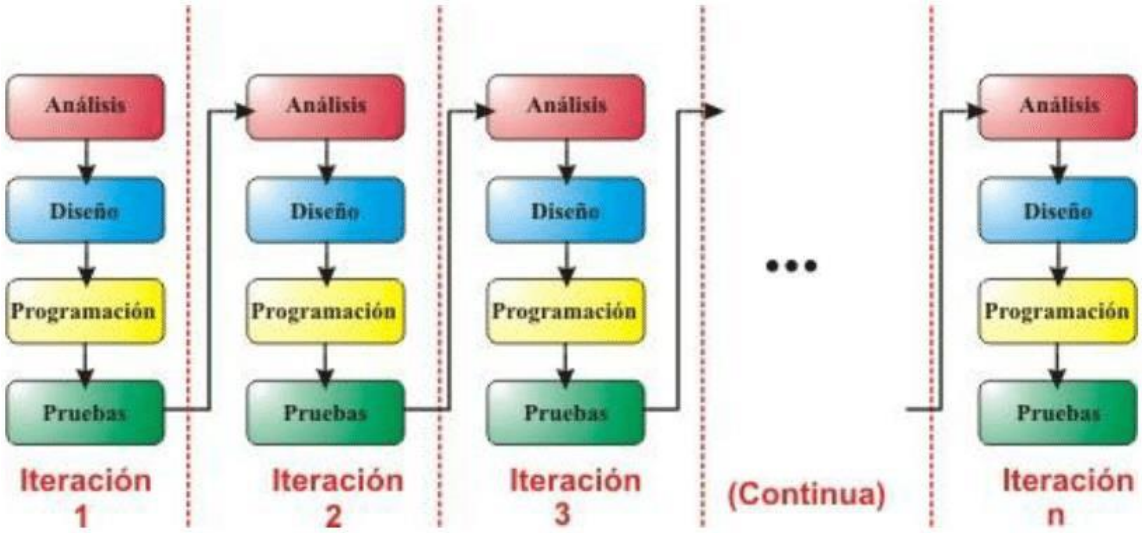

Figura 1: Desarrollo iterativo incremental [2]

Esta metodología nos permite gestionar las expectativas del cliente de manera regular, así como permitirle realizar cambios durante el desarrollo y permitirle ser participe en la toma de decisiones.

A continuación, se indica el contenido de las iteraciones de las que se ha compuesto el proyecto. Este se ha desarrollado únicamente por el autor del proyecto, con la guía del codirector del proyecto:

- Iteración 1: Tratamiento de datos recibidos en formato txt haciendo uso del lenguaje de programación Python. Al acabar esta iteración, se dispuso de cuatro ficheros NetCDF, uno por variable, donde se encontraban almacenadas las observaciones realizadas por las estaciones, así como los metadatos de cada estación.

- Iteración 2: Desarrollo de un servicio, que lea los ficheros creados en la iteración anterior, para así poder hacer uso de ellos.

- Iteración 3: Desarrollo de un controlador, que permita que sea posible el intercambio de información entre los datos obtenidos en el servicio y la capa de presentación.

- Iteración 4: Desarrollo de la vista inicial de la capa de presentación.

- Iteración 5: Desarrollo de un método en el servicio que nos permita la obtención de los datos recogidos en el día requerido para una variable determinada y los ponga a disposición de la capa de presentación en el controlador.

- Iteración 6: Implementación de la operación para mostrar los marcadores de los datos solicitados en el formulario inicial en la capa de presentación, datos que se obtendrán del servicio de la anterior iteración.

- Iteración 7: Implementación de la operación que nos permita la apertura de una ventana emergente sobre la pantalla para visualizar los detalles de la estación seleccionada.

- Iteración 8: Desarrollo de un método en el servicio que obtenga los datos característicos de una estación determinada haciendo una lectura de una base de datos *MySQL*, donde se encuentra almacenada esta información. Además, en el controlador, se implementa un método que proporcione la información obtenida en el servicio al cliente.

- Iteración 9: Implementación de la generación de gráficas en el cliente mediante *Plotly* [3], que nos permitan la generación de histogramas. Uno de estos mostrará los datos mensuales y el otro los datos anuales. Estos datos se obtendrán de métodos implementados en esta iteración en el servicio y el controlador.

- Iteración 10: Implementación de la generación de tablas de los valores máximos y mínimos recogidos por las estaciones acorde a distintos periodos temporales. Estos datos se obtendrán de los métodos generados para ello en el servidor y el controlador, implementados en esta misma iteración.

2. Tecnologías y herramientas

En este apartado se detallan brevemente las tecnologías y herramientas que han sido utilizadas a lo largo del desarrollo del proyecto.

2.1. Lenguajes de programación

Para el desarrollo de este TFG se ha hecho uso de los siguientes lenguajes de programación:

2.1.1. TypeScript

Lenguaje de programación de código abierto desarrollado y mantenido por Microsoft. Escrito sobre JavaScript, añade tipado estático y objetos basados en clases, facilitando su comprensión y mejorando la calidad del código generado.
TypeScript, al convertir su código a JavaScript, permite ser ejecutado en el navegador sin necesidad de que este sepa si fue o no escrito en este lenguaje, al ejecutar únicamente el JavaScript resultante de la compilación del código.

2.1.2. Java

Lenguaje de programación de alto nivel orientado a objetos propiedad de la compañía Oracle. Su objetivo es permitir a los desarrolladores ejecutar el código Java compilado en todas las plataformas que admitan *Java* sin necesidad de un recompilado. Las aplicaciones en este lenguaje son compiladas a bytecode, lo que nos permite ejecutar los programas en cualquier máquina virtual Java sin importar la estructura del computador.

2.1.3. Python

Lenguaje multiparadigma de propósito general que se ha popularizado en los últimos años debido a su sencillez y su facilidad a la hora de crear programas, además de la gran cantidad de librerías y funcionalidades que trae incorporadas. Es un lenguaje gratuito en el que podemos programar en los principales sistemas operativos del mercado.

2.2. Herramientas

Para el desarrollo de este proyecto, se ha hecho uso de las siguientes herramientas:

2.2.1. Angular

Framework de código abierto para el desarrollo de aplicaciones web creado por Google que nos permite realizar y mantener aplicaciones de una sola página. Estas aplicaciones de una sola página se caracterizan por ir cargando los recursos según lo requiera la aplicación, como respuesta a las acciones del usuario, lo que proporciona una navegación más fluida que las aplicaciones tradicionales.
Uno de sus propósitos es facilitar la utilización del patrón Modelo-Vista-Controlador.

2.2.2. Spring

Framework de código abierto para el desarrollo de aplicaciones Java que implementa el patrón de inversión de control. Cuenta con varios módulos que facilitan, entre otros, el desarrollo de servicios web, el acceso a bases de datos o aspectos de seguridad. Dentro de Spring [4], cabe destacar SpringBoot, que facilita la creación y despliegue de proyectos que utilicen este *framework*.

2.2.3. Pandas

Librería software para el análisis y tratamiento de datos para Python. Es útil, en particular, para el tratamiento de series temporales y tablas de datos. Tiene la ventaja de realizar las operaciones de manera rápida y eficiente y además, permitir manipular los datos con gran facilidad.

2.2.4. NetCDF

Formato creado por UNIDATA destinado al almacenamiento de datos científicos multidimensionales. Un archivo en este formato se compone principalmente de dimensiones, variables y atributos. A diferencia de otros formatos, NetCDF [5] no necesita hacer uso de archivos adicionales para determinar qué tipo de datos se encuentran almacenados. Tiene como ventaja su alta portabilidad y su eficiencia a la hora de tratar los datos.

2.2.5. Eclipse

Entorno de código abierto y multiplataforma para el desarrollo software. Es uno de los entornos de programación más empleados debido a su versatilidad y a la gran variedad de lenguajes de programación que soporta.

2.2.6. Leaflet

Librería de código abierto para JavaScript que nos permite la publicación de mapas en entornos web. Destaca por su facilidad de uso frente a la complejidad de otras librerías de mapas, además de por su dinamismo y la amplia variedad de opciones que ofrece.

2.2.7. MySQL

Sistema de gestión de base de datos relacionales de código abierto. En este proyecto se empleará únicamente para almacenar y obtener metadatos de las estaciones para su posterior uso.

2.2.8. WampServer

Herramienta software para el sistema operativo Windows que nos provee de un sistema de gestión de bases de dato MySQL y un servidor web Apache Tomcat.

2.2.9. SonarQube

Plataforma que permite realizar análisis de código con diferentes herramientas de manera automatizada. El resultado de estos análisis puede ayudar a mejorar la calidad de código de un programa.

3. Especificación de requisitos

En este apartado, se detalla la especificación de requisitos del sistema. Se organiza en tres apartados: requisitos funcionales, no funcionales y, casos de uso y mockups.

3.1. Requisitos funcionales

En la tabla 1 se recogen los requisitos funcionales que deberá cumplir el sistema para considerarse completado.

ID	Descripción
RF-001	El usuario podrá visualizar sobre un mapa las estaciones que han recogido valores para una determinada variable y día seleccionados.
RF-002	El usuario podrá visualizar diferentes histogramas de los datos recogidos por la estación acorde a diferentes criterios.
RF-003	El usuario podrá visualizar los metadatos de una estación cuando ésta sea seleccionada.
RF-004	El usuario podrá visualizar en formato tabla los valores máximos y mínimos acorde a diferentes criterios

Tabla 1: Requisitos funcionales

3.2. Requisitos no funcionales

Los requisitos no funcionales nos definen las características requeridas que el sistema debe poseer para certificar que es válido para su uso.

A continuación, en la tabla 2, se especifican los requisitos no funcionales de la aplicación.

ID	Descripción
RNF-001	El sistema deberá disponer de conexión a internet para su funcionamiento.
RNF-002	El sistema deberá disponer de un navegador de internet Chrome, Firefox, Safari o Internet Explorer 10 o superior.
RNF-003	La aplicación deberá tener un tiempo de respuesta en las peticiones realizadas por los usuarios inferior a diez segundos.
RNF-004	La aplicación deberá funcionar en todo tipo de dispositivos con un navegador web de los citados anteriormente.
RNF-005	La aplicación deberá ser intuitiva y fácil de utilizar por todo tipo de usuarios, tanto usuarios expertos como usuarios con competencias digitales básicas.
RNF-006	El sistema deberá ser altamente mantenible y adaptable a nuevos requerimientos, especificaciones o funcionalidades.
RNF-007	El sistema deberá notificar al usuario en los casos de error o en los casos en los que sus peticiones no tengan respuesta.

RNF-008	El usuario podrá visualizar los textos de la aplicación tanto en español como en inglés.

Tabla 2: Requisitos no funcionales

3.3. Casos de uso

A continuación, se relacionan los casos de uso que serán implementados para satisfacer los requisitos funcionales.

Identificador + nombre	CU-001 - Visualización general de datos
Autor	Fernando Arruti
Actor	Usuario
Descripción	El usuario desea visualizar los datos proporcionados por la aplicación
Precondición	
Postcondición	
Garantías de éxito	El usuario podrá visualizar los datos de todas las opciones disponibles.
Garantías mínimas	El sistema se mantiene estable
Escenario principal	1. El usuario introduce los valores requeridos en formulario principal y selecciona el botón de búsqueda. 2. El sistema muestra sobre el mapa marcadores de las estaciones que tienen un valor recogido de la variable introducida en la fecha seleccionada. 3. El usuario selecciona un marcador de los mostrados por pantalla. 4. El sistema muestra por pantalla las características de la estación seleccionada. 5. El usuario introduce en un formulario las dos fechas entre las cuales quiere visualizar un histograma. 6. El sistema muestra un histograma con los datos mensuales recogidos. 7. El usuario cambia a la pestaña “Gráfico anual” 8. El usuario introduce los meses de los que quiere visualizar datos en el formulario habilitado para ello. 9. El sistema muestra al usuario un gráfico con los datos anuales recogidos. 10. El usuario selecciona la pestaña “datos estadísticos”. 11. El usuario accede a la pestaña mensual.

	13. El usuario introduce un mes en el formulario habilitado para ello en la pestaña “Mensuales”. 14. El sistema muestra al usuario dos tablas al usuario con los valores máximos y mínimos mensuales del mes introducido, dependiendo de la pestaña sobre la que se encuentre, pudiendo cambiar de pestaña el usuario siempre que desee. 15. El usuario selecciona la pestaña “Estacionales”. 16. El usuario introduce una de las cuatro estaciones del año. 17. El sistema muestra al usuario dos tablas con los valores máximos y mínimos estacionales de la estación introducida para la pestaña sobre la que se encuentre, pudiendo cambiar esta siempre que desee. 18. El usuario selecciona la pestaña “Totales”. 19. El sistema muestra dos tablas al usuario con los valores máximos y mínimos diarios registrados de la variable de la pestaña sobre la que se encuentre. 20. El usuario selecciona la pestaña “Anuales”. 21. El usuario introduce el año del que quiera obtener datos. 22. El sistema muestra dos tablas al usuario con los valores máximos y mínimos diarios registrados en el año introducido para la pestaña sobre la que se encuentre, pudiendo cambiar esta siempre que desee. 23. El usuario selecciona la pestaña “Diarios”. 24. El usuario introduce el día de un mes de que quiera visualizar datos. 25. El sistema muestra dos tablas al usuario con los valores máximos y mínimos registrados en el día introducido para la pestaña sobre la que se encuentre, pudiendo cambiar esta siempre que desee.
Extensiones	
Casos de error	1.a. El usuario no introduce todos los valores requeridos. El sistema se lo notifica al usuario para que los introduzca.

	2.a., 6.a, 9.a, 14.a, 17.a, 19.a, 22.a, 25.a.: El sistema no obtiene datos de la información requerida. El sistema se lo notifica al usuario, que puede realizar otra búsqueda. 4.a. El sistema no obtiene datos de las características de la estación marcada. El sistema le notifica al usuario el problema y se muestra la estación sin datos.

Tabla 3: Caso de uso general: Visualización de datos por parte del usuario

3.4. Mockups

Los mockups son representaciones de lo que se quiere ver en la futura aplicación a desarrollar y nos permiten tener una primera aproximación a su diseño. Estas representaciones nos aportan una imagen sobre aspectos como son la distribución de los componentes y contenidos en la interfaz o la navegación del usuario a través de las diferentes pantallas del sistema.

Los mockups son utilizados a menudo como demostración del sistema final o para validar si el diseño de la aplicación final se ajusta a lo requerido.

En las figuras 2, 3 y 4 se muestran los mockups de las diferentes pantallas de las que se compone el sistema.

La Figura 2 representa la pantalla de inicio, con la que se encontrará el usuario una vez acceda a la aplicación. En ella podemos ver que la pantalla de inicio de la futura aplicación estará compuesta de varias secciones: Una cabecera, un formulario donde el usuario introducirá los valores a buscar, el mapa *Leaflet* [6] ya inicializado y un pie de página.

Figura 2: Pantalla inicial

En la Figura 3, se representa la pantalla en la que el usuario se encontrará una vez haya introducido los valores requeridos y solicitado su búsqueda mediante la pulsación del botón habilitado para ello. En esta pantalla ya se encuentran representados los marcadores de las estaciones que tienen datos de la variable seleccionada en la fecha requerida.

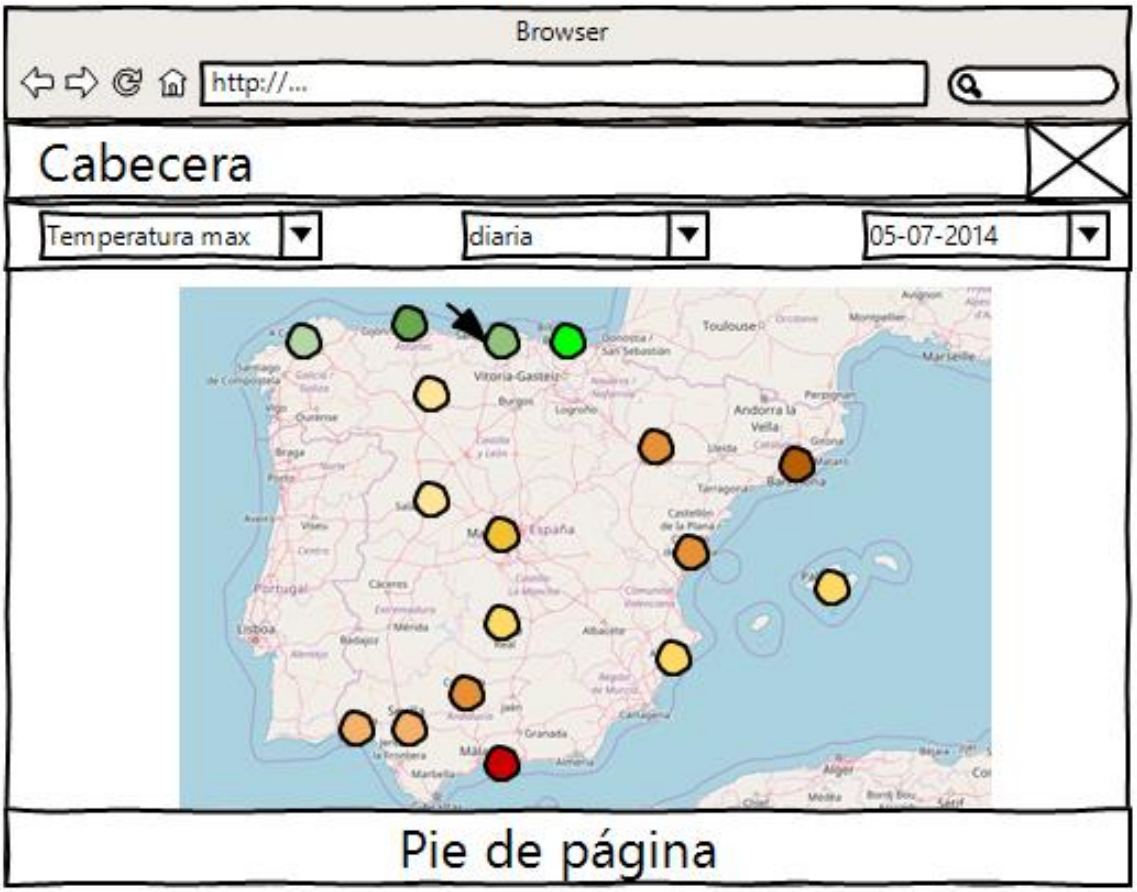

Figura 3: Pantalla con marcadores sobre mapa

La Figura 4 representa la página en la cual el usuario ha clicado sobre uno de los marcadores mostrados en la figura anterior. En ella podemos observar cómo se mostrarían los detalles de la estación seleccionada y cómo nos permitiría la creación de gráficas y tablas a partir de los datos recogidos en esta estación.

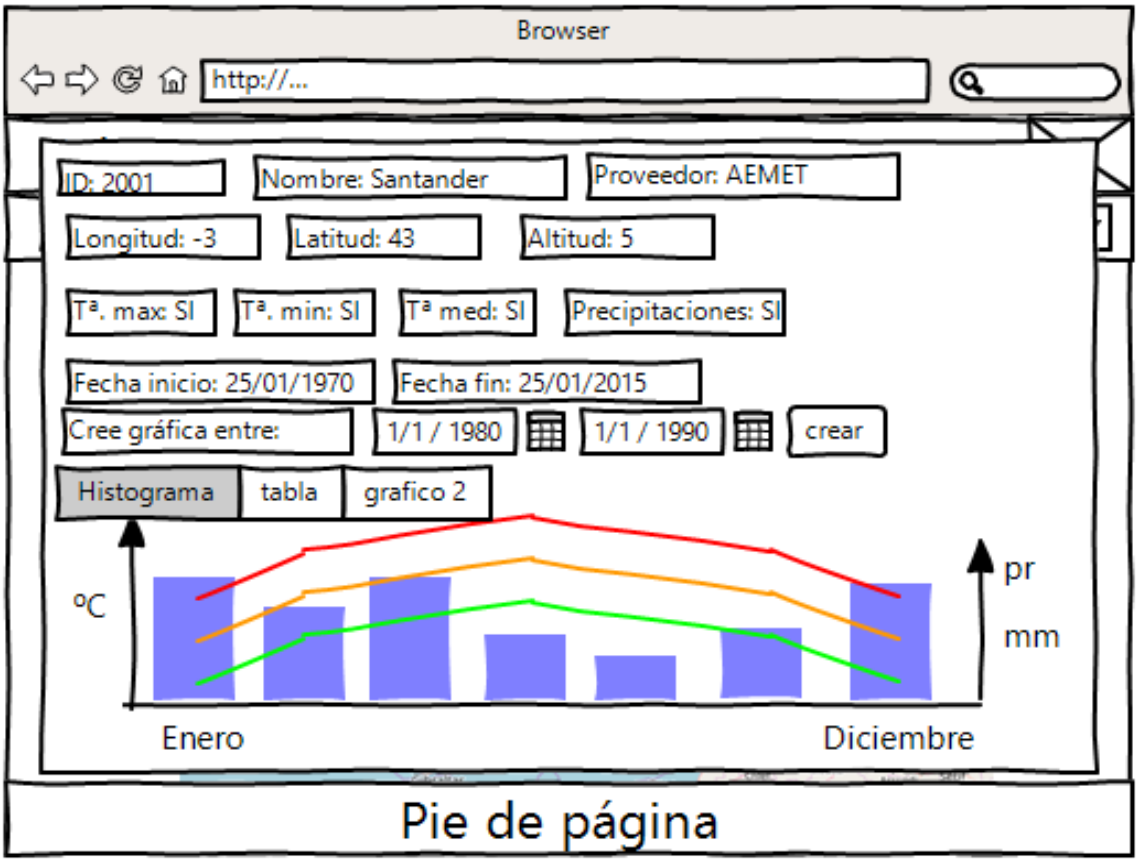

Figura 4: Pantalla con datos de estación seleccionada

4. Diseño del sistema

En este apartado, se describe el diseño que se seguirá para implementar la aplicación del proyecto.

4.1. Diseño arquitectónico

El patrón arquitectónico empleado en la aplicación es el patrón modelo-vista-controlador, que nos permite estructurar en tres capas los componentes de la misma. Esto nos permite facilitar el mantenimiento del código, su reutilización, así como facilitar su desarrollo. Al ser una aplicación de una sola página, este patrón es implementado sobre el código JavaScript, siendo el servidor encargado únicamente de responder a las llamadas del cliente.

Este patrón se compone de tres elementos:

1) Modelo: Capa de datos que contiene la información almacenada que utiliza el sistema, y contiene las herramientas para acceder a los mismos.
2) Controlador: Capa de negocio que se encarga del flujo de datos entre el modelo y la vista.
3) Vista: Capa de presentación encargada de mostrar la información al usuario. Permite la interacción con el usuario a través de una interfaz.

4.1.1. Diseño detallado

A continuación, se procede a especificar el diseño detallado de la aplicación, cuyo diagrama se muestra en la Figura 5

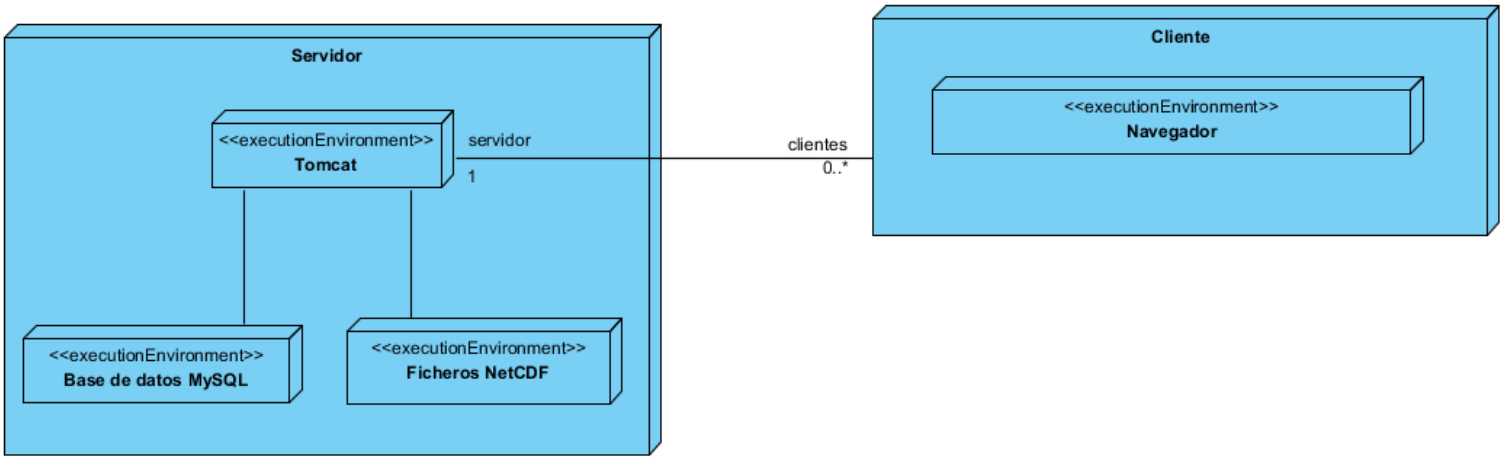

Figura 5: Diagrama de despliegue

En este diagrama, se puede observar que está compuesto de dos módulos:

- El primero, el servidor, contendrá el servidor Tomcat, que nos permitirá la comunicación con el cliente. Además, alojará a la base de datos MySQL y a los ficheros NetCDF, que contienen los datos a entregar al cliente.

- El segundo, el cliente, que será donde el usuario ejecute la aplicación desde su navegador web.

En la Figura 6, se muestra el diagrama de la base de datos MySQL de la aplicación, que consistirá en una única tabla, que almacena los datos característicos de cada estación de la red secundaria. El motivo de existencia de esta base de datos es almacenar valores de uso recurrente y evitar así acceder a leer estos datos a los ficheros NetCDF, cuyo coste de lectura es mayor.

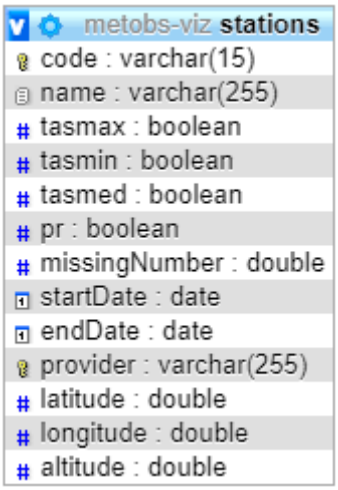

Figura 6: Diagrama de Base de Datos MySQL

Como se puede ver, la entidad estación, consta de una clave primaria de dos atributos, código y proveedor, para garantizar que no permita el almacenado de estaciones con el mismo código del mismo proveedor. Esto permitirá en un futuro almacenar estaciones de distintos proveedores, que pudieran tener el mismo código que las estaciones que trata actualmente el sistema, las de la red secundaria de AEMET.

5. Desarrollo del sistema

En este apartado, se detalla cómo se ha implementado el sistema

5.1. Modelo

En primer lugar, se describe el proceso que se ha llevado a cabo para procesar los datos iniciales recibidos y su adaptación a las necesidades del problema planteado.

Estos datos se encontraban almacenados en un conjunto de ficheros de texto, donde estaban recogidos los datos diarios de temperaturas o precipitaciones relativos a cada estación. Estos valores no se encontraban en las unidades del sistema internacional, por lo que fue necesario adaptarlos al estándar. Además, se contaba con un fichero adicional en formato xls en el que se encontraban los atributos de todas las estaciones de la red secundaria, como son las coordenadas, el nombre y el identificador de la estación.

En las figuras 7, 8 y 9, se muestran capturas de los tres tipos de ficheros de datos de los que obtendremos la información de precipitaciones, temperaturas y características de las estaciones respectivamente.

Datos-Pd(1950-1954).txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
INDICATIVO;AÑO;MES;DIA;NOMBRE;ALTITUD;C_X;C_Y;NOM_PROV;LONGITUD;LATITUD;P77
0002;1950;1;1;L'AMETLLA DE MAR;22;820261;4532998;TARRAGONA;0048031;405305;0
0002;1950;1;2;L'AMETLLA DE MAR;22;820261;4532998;TARRAGONA;0048031;405305;25
0002;1950;1;3;L'AMETLLA DE MAR;22;820261;4532998;TARRAGONA;0048031;405305;0
0002;1950;1;4;L'AMETLLA DE MAR;22;820261;4532998;TARRAGONA;0048031;405305;0
0002;1950;1;5;L'AMETLLA DE MAR;22;820261;4532998;TARRAGONA;0048031;405305;0
```

Figura 7: Fichero de precipitaciones

Datos-Td(1950-1959).txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
INDICATIVO;AÑO;MES;DIA;NOMBRE;ALTITUD;C_X;C_Y;NOM_PROV;LONGITUD;LATITUD;TMAX;TMIN;TMED
0013;1950;1;1;CAMBRILS;19;840590;4554931;TARRAGONA;0103131;410425;;80;
0013;1950;1;2;CAMBRILS;19;840590;4554931;TARRAGONA;0103131;410425;;60;
0013;1950;1;3;CAMBRILS;19;840590;4554931;TARRAGONA;0103131;410425;;60;
```

Figura 8: Fichero de temperaturas

	A	B	C	D	E
1	IND	ESTACION	LON	LAT	ALT
2	0001	EL PERELLO	0.71139	40.875	142
3	0002	L'AMETLLA DE MAR	0.80083	40.8847	22
4	0002I	VANDELLOS (CENTRAL NUCLEAR)	0.87167	40.9586	34
5	0008	MONTBRIO DEL CAMP	1.0036	41.1194	120
6	0009	ALFORJA (HOSPITAL)	0.97583	41.2086	370
7	0013	CAMBRILS	1.0536	41.0736	19
8	0016	REUS	1.1167	41.15	100

Figura 9: xls de características de estaciones

La Figura 10 ilustra el proceso que se sigue para llevar esta información al formato estándar. Se reciben un conjunto de datos de observaciones almacenados en distintos ficheros de texto, y un fichero en formato xls, en el cual se encuentra almacenada la información relativa a las coordenadas y nombre las estaciones de la estación secundaria. De este conjunto de datos, queremos generar cuatro ficheros NetCDF, uno por variable recogida, que contengan las observaciones cosechadas y la información referente a las estaciones.

Se quieren almacenar estos datos en formato NetCDF, porque este tipo de formato nos permite tratar series temporales con facilidad, así como un acceso rápido a estos datos.

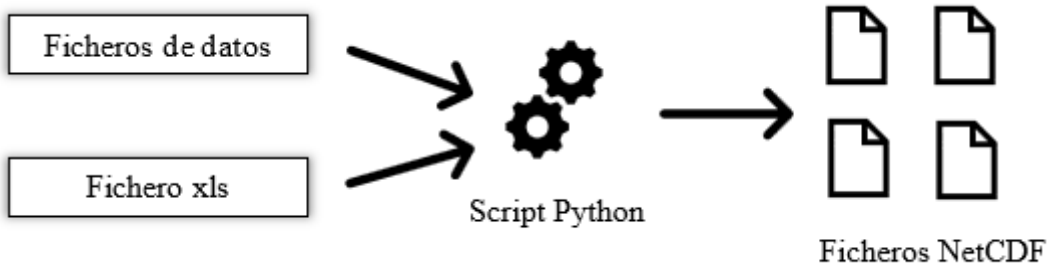

Figura 10: Flujo de datos en el modelo

En este proceso se ha hecho uso del lenguaje de programación Python y la librería Pandas, ya que nos permiten tratar datos y series temporales con facilidad.

A continuación, se detallan los pasos que se han seguido para tratar estos datos.

Para comenzar, fue necesario un filtrado de los ficheros de texto para obtener únicamente los valores que resultaban relevantes para el problema planteado. Es por esto, por lo que seleccionamos únicamente los valores de temperatura y precipitación con su fecha de recogida. Descartaremos de estos ficheros las coordenadas, que no se encuentran en el estándar internacional, el nombre de la estación, que obtendremos del fichero xls, y el nombre de la provincia, que no resulta relevante para el problema planteado.

Más adelante, fue necesario convertir los datos de precipitación que estaban marcados como dato no recogido (valor igual a menos cuatro) en la fecha indicada y su valor inmediatamente posterior a Not a Number. Esto es debido, a que hay fechas en las que no se ha comprobado el valor recogido en alguna de las estaciones, por lo que el valor recogido en ese día no está disponible, y el dato del siguiente día es probable que sea erróneo al contener precipitación acumulada del día anterior. También se ha requerido modificar los valores que estén marcados como inapreciables (valor igual a menos 3) transformándolos a cero, ya que este sería aproximadamente su valor.

A continuación, se procedió a adecuar los nombres de las variables y las unidades al estándar internacional. Se requirió detectar y solucionar problemas con las fechas recogidas, como pudiera ser el caso de que existieran fechas repetidas o hubiera fechas ausentes entre la primera y la última fecha en la que se disponen de datos.

Una vez completado esto, se realizó la lectura del fichero xls, del cual se obtuvo el nombre y las coordenadas de cada una de las estaciones. Estos metadatos se añadieron al conjunto de datos anterior y todo ello se guardó en un fichero NetCDF.

Al finalizar esta fase se tuvieron estos datos tratados y almacenados en cuatro ficheros NetCDF, uno por cada variable. Cada fichero NetCDF contiene una serie temporal con las mediciones de cada estación y sus metadatos.

5.2. Implementación del controlador y del servicio de acceso a datos

A continuación, se explica cómo se realizó la implementación que nos permitirá hacer uso de los datos anteriormente tratados.

La implementación de la lectura de los ficheros NetCDF hizo por medio de las herramientas proporcionadas para ello por la UCAR (University Corporation for Atmospheric Research). Con esto, se consigue acceso a los datos diarios recogidos por las estaciones y a sus características básicas (nombre, código y posición geográfica).

Para satisfacer los requisitos eran necesarias otras características, que no están recogidas en los ficheros NetCDF, como son el día que empezó a recoger datos una estación determinada o el porcentaje de días sin datos. El cálculo de estas características fue implementado en Java y para su almacenamiento, fue necesario hacer uso de la herramienta JOOQ, que permite la ejecución de sentencias SQL para poder así almacenar

estos valores en una base de datos MySQL y poder hacer uso posteriormente de esta información en el sistema. La obtención de estas características y su almacenado en una base de datos, ha sido implementado sobre un test unitario implementado con JUnit. La lectura de los datos de la base de datos MySQL generada y su utilización se ha realizado en el servicio REST.

Para que la capa de presentación tuviera acceso a estos datos, se creó un servicio REST que permitirá que la capa de presentación pueda obtener estos datos.

Este servicio REST proporcionará un JSON con los datos que se le soliciten mediante las llamadas del cliente.

En la Figura 11, se muestra la clase del ObservationReader, que implementa el servicio encargado de la lectura de los datos almacenados en los ficheros NetCDF requeridos por el usuario.

Como se puede ver, esta clase contiene una serie de métodos que nos retornarán los datos solicitados en función de una serie de características. En nuestro caso, el cliente solo realizará peticiones de lectura al servidor, y no podrá modificar ni realizar escrituras sobre los datos ya existentes.

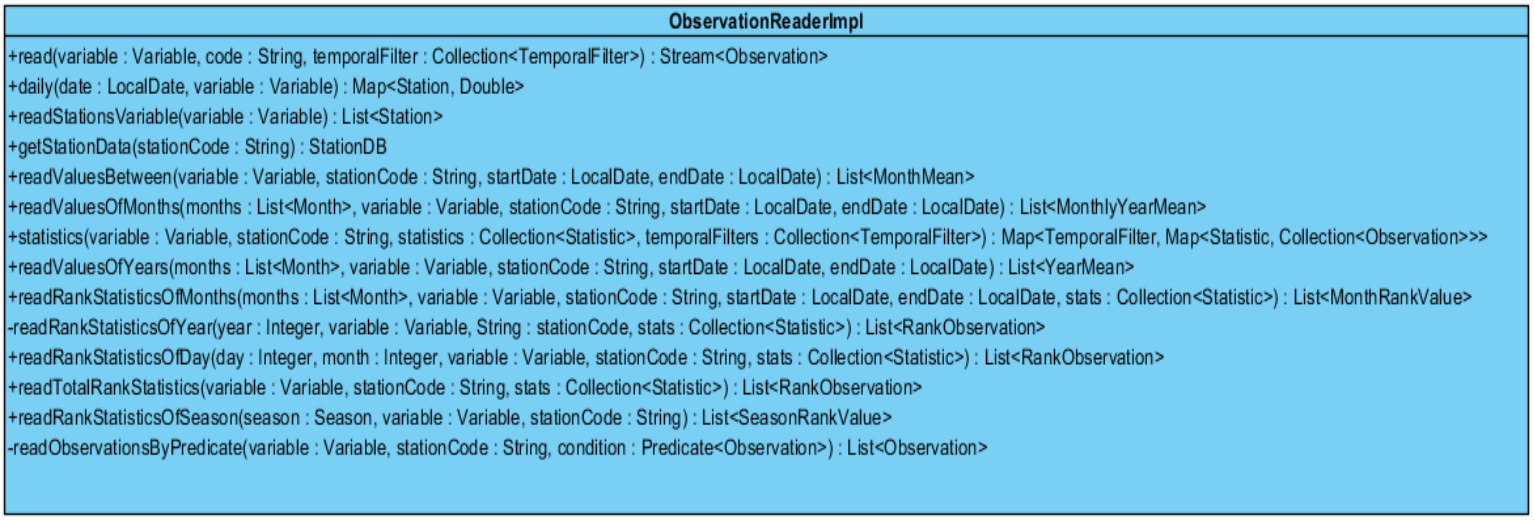

Figura 11: Clase servicio ObservationReader

Este servicio realiza la lectura de los ficheros NetCDF para obtener de los datos leídos agregaciones y valores estadísticos de estos.

Para evitar la carga en memoria de datos a escala diaria, el cálculo de los datos estadísticos se implementó de manera incremental. Esto es debido a que, de cada estación de las que se disponen, se almacenan algo más de veinticuatro mil observaciones, lo que hace inviable el cálculo de estos datos por la vía tradicional ya que su coste sería muy elevado.

Para agilizar las peticiones del cliente, se ha permitido el almacenamiento en cache de datos según las peticiones, para así minimizar el tiempo de respuesta de las solicitudes del cliente cuando estas sean muy similares o empleen el mismo conjunto de datos.

5.3. Implementación de la capa de presentación

En este apartado, se expone como se ha llevado a cabo la implementación y el diseño de la capa de presentación, que permitirá al usuario tener una interacción directa con la aplicación. Esta implementación se realizó con Angular [7] y las herramientas que proporciona.

Las interfaces de usuario están basadas en HTML y hacen uso de CSS, y todo lo relacionado con las funcionalidades del sistema hace uso de TypeScript. En la siguiente imagen, Figura 12, se muestra la organización de esta parte del proyecto.

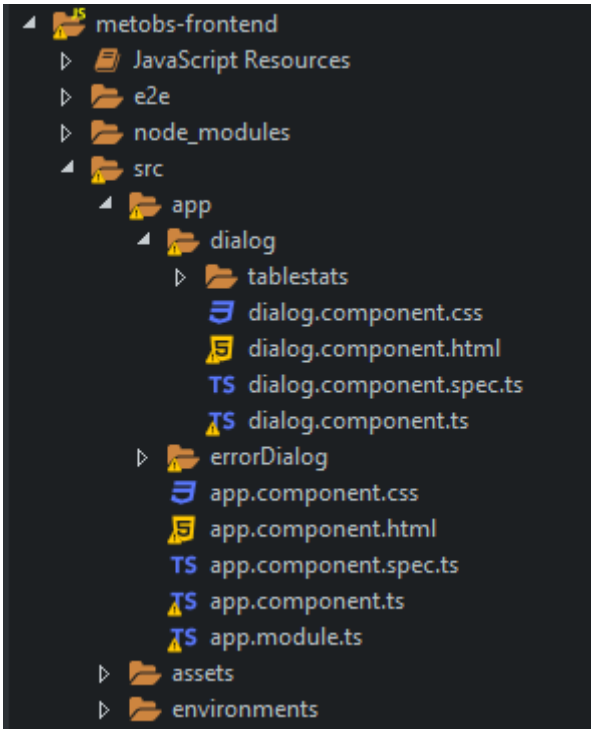

Figura 12: Estructura capa de presentación

La capa de presentación está compuesta de cuatro elementos:

1. Componente principal(app.component): En él se recogen la interfaz principal de la aplicación y sus funcionalidades (ficheros HTML, CSS y TypeScript). También se encuentra en este componente el fichero app.module, que es un fichero general donde se recogerán e importarán todos los paquetes, módulos y componentes empleados en el sistema.

2. Componente Dialog: En él se recoge la interfaz y funcionalidades de la vista de detalle de las estaciones. Este componente se desplegará sobre el componente principal en el caso de ser inicializado.

3. Componente ErrorDialog: Este componente es simplemente una ventana emergente que se desplegará sobre la aplicación para notificar al usuario de algún suceso que se haya producido durante la ejecución del programa. Este componente recibe como entradas un título y un mensaje que serán los textos a mostrar al usuario. En la Figura 13 se muestra el aspecto de este mensaje de error.

4. Componente tableStats: Componente auxiliar para mostrar las tablas de los datos máximos y mínimos en la aplicación.

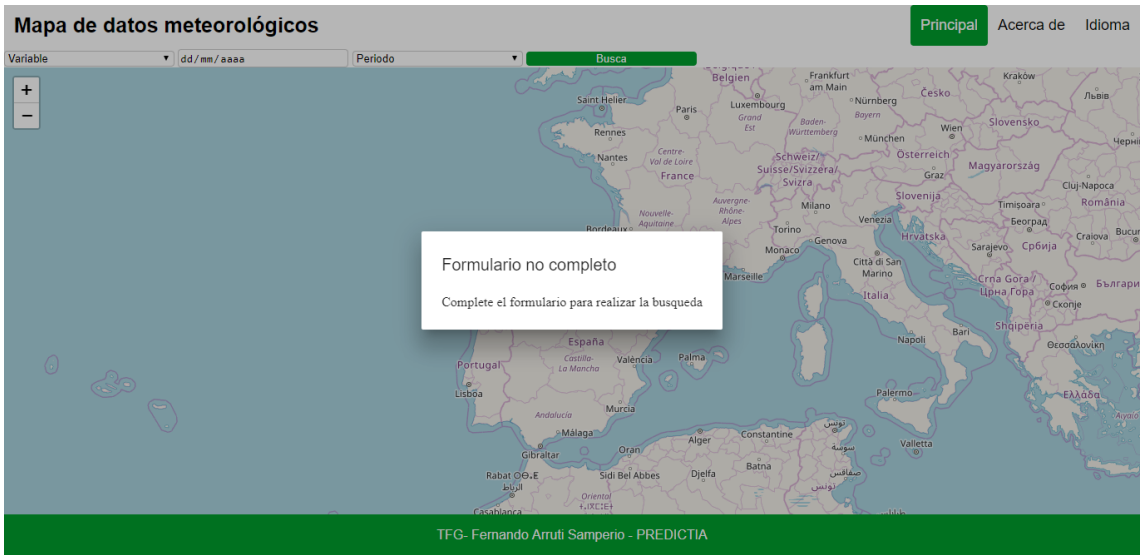

Figura 13: Mensaje de error

A continuación, se detalla la implementación de la capa de presentación.

En la Figura 14, se muestra la página principal del sistema y lo que se encontrará el usuario nada más iniciar la aplicación.

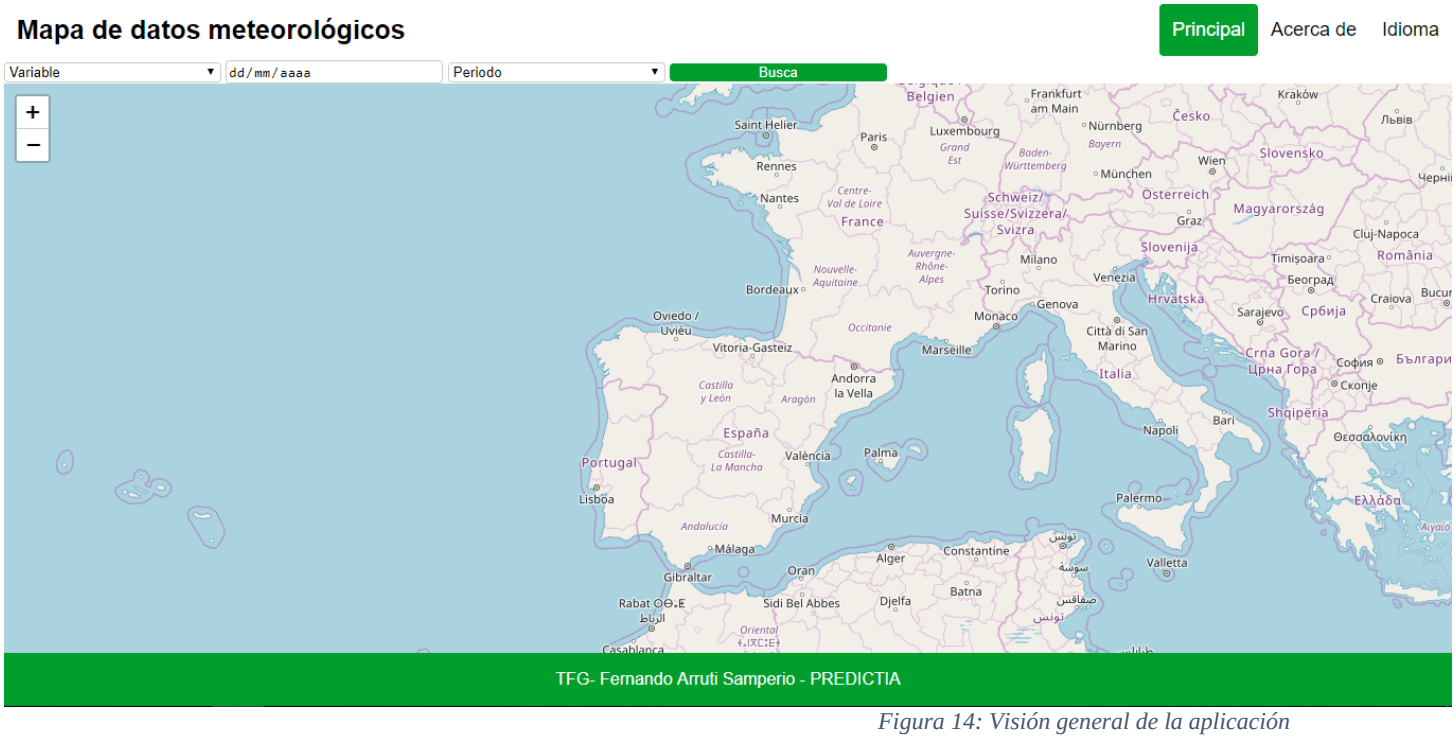

Figura 14: Visión general de la aplicación

Como se puede observar, en la cabecera, a la derecha, se encuentra la pestaña “Idioma”. Esta pestaña, generará un desplegable al pasar el cursor sobre ella, que nos permitirá cambiar el idioma a una de las dos opciones disponibles en la aplicación, español e inglés.

Esto ha sido posible haciendo uso de la herramienta ngx-translate [8], que permitirá traducir las palabras que se deseen de un diccionario que hayamos definido previamente.

A continuación, se detalla el funcionamiento de esta herramienta de traducción.

En la sección de código siguiente, se encuentran los métodos que serán activados cuando el usuario pulse sobre una de las dos opciones disponibles. Dentro de estos métodos, se realizará una llamada al servicio *translate*, que pondrá a por defecto el idioma seleccionado por el usuario, y realizará la traducción de los textos en los que así se haya indicado a este idioma.

TypeScript

```
onclickEsp() {
            this.translate.setDefaultLang('es');
            this.translate.use('es');
}
onclickEng() {
            this.translate.setDefaultLang('en');
            this.translate.use('en');
}
```

El siguiente fragmento de código es solo una muestra de cómo se realiza una traducción de un texto sobre el código HTML. Aquí, el servicio *translate* buscará en el diccionario definido la palabra con clave “MeteoMap”, y la traducirá al idioma que este activo.

HTML

```
{{'MeteoMap' | translate }}
```

La siguiente sección de código, realiza la misma función que el fragmento anterior, pero esta vez en el lenguaje TypeScript. Buscará la palabra “English” en el diccionario y la almacenará en la variable “month”, según el idioma que este activo en ese momento.

TypeScript

```
this.translate.get('English').subscribe((res: string) => {
            month = res;
});
```

Seguidamente, se detalla el proceso seguido para poder hacer uso del mapa *Leaflet* en la aplicación.

Para inicializar el mapa y mostrarlo por pantalla, solo ha sido necesario hacer uso de las siguientes líneas de código HTML y TypeScript.

HTML

```
<div id="map" style="height: 80%;"></div>
```

TypeScript

```
ngOnInit (): void {
       // inicializa mapa
      this.map = L.map('map', {
             center: [40.4167, -3.70325],
             minZoom: 2,
             zoom: 5
        });
      L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png')
      .addTo(this.map);
}
```

En el fichero HTML, lo único que haremos será crear una sección de un tamaño determinado, en la cual insertaremos el mapa desde el código TypeScript. En el fichero TS, el método `ngOnInit` se ejecutará una vez se arranque la aplicación. En este, inicializamos el mapa *Leaflet* en la sección creada anteriormente, y ajustamos el centro inicial en un punto determinado (en este caso, seleccionamos la puerta del Sol de Madrid al estar aproximadamente en el centro peninsular). También le indicamos que haga uso del mapa *OpenStreetMap* haciendo uso de la función `tileLayer` sobre la referencia al mapa.

Una vez se nos muestra el mapa, es necesario permitir al usuario introducir una fecha en la que quiera visualizar los datos recogidos por las estaciones de una variable en concreto, y mostrar en el mapa un marcador para cada estación con un valor recogido.

Para ello, se ha procedido con la creación de un formulario que llame a una función que nos obtenga estos datos y nos los muestre representados con marcadores sobre el mapa.

A continuación, se encuentra el código HTML con este formulario.

HTML

```
<form #dataForm="ngForm" (ngSubmit)="onSubmit(dataForm)">
<label for="variable">Variable</label>
<select name="variable" ngModel>
       <option value="TEMPERATURE_MAX">Maximum temperature</option>
       <option value="TEMPERATURE_MIN">Minimum temperature</option>
       <option value="TEMPERATURE_MEAN">Mean temperature</option>
       <option value="PRECIPITATION">Precipitation</option> </select>
<label for="date">Date</label>
<input  type="date"  name="date"  id="date"  min="1950-01-01"  max="2015-12-
31"ngModel>
<label for="range">Rango periodo</label>
<select name="period" ngModel>
       <option value="daily">Daily</option>
       <option disabled value="weekly">Weekly</option>
       <option disabled value="monthly">Monthly</option>
       <option disabled value="yearly">Yearly</option>
</select> <button type="submit">Busca</button>
</form>
```

En él, se da la posibilidad al usuario de seleccionar una de las cuatro variables recogidas en nuestro sistema y le indicamos que seleccione una fecha de entre las que se disponen de datos. También le damos la posibilidad de seleccionar el tipo de dato que desee, en el caso de querer mostrar datos semanales, mensuales o anuales, en vez de diarios, que es la

opción por defecto. En el desarrollo realizado, solo se ha implementado el caso de los datos diarios, dejando el resto de las opciones deshabilitadas para programación en un futuro.

Este formulario, una vez completado, mostrará los resultados cuando se pulse el botón de búsqueda, que llamará a la función `onSubmit` pasándole como parámetro los datos introducidos en este.

A continuación, se detalla la implementación de esta función `onSubmit`, que recogerá los datos introducidos por el usuario y mostrará los marcadores en el mapa *Leaflet* a partir de estos.

TypeScript

```
onSubmit(form: NgForm) {
       // escala de colores
       let scale;
       if (form.value.variable === 'PRECIPITATION') {
            scale = chroma.scale(['#FFFFFF', '#90EE90', '#009FFF', '#6666FF',
            '#1306FF', '#9123FF', '#6E00DB', '#5E00A6', '#410059' ]).domain([0,
            30, 60, 90, 120, 150, 180, 210, 239]);
       } else {
            scale = chroma.scale(['#2E2E73', '#282898', '#201FBB', '#1A1ADC',
            '#3654DE', '#548EDC', '#72CADE', '#6DD8DF', '#55CDE2', '#38BBDC',
            '#20B0DC', '#19BAA6', '#1CCE6A', '#1BDF22', '#82C319', '#DCA819',
            '#DD921A', '#DE7C1A', '#DF671A', '#DE501A', '#DD3819', '#DD2319',
            '#D21A1E', '#C31927', '#AD1A30', '#9A1A3B', '#871A44', '#871A44' ])
            .domain([-3, -1.5, 0, 1.5, 3, 4.5, 6, 7.5, 9, 10.5, 13, 15.5, 18,
            19.5, 21, 22.5, 24, 25.5, 27, 29.5, 31, 32.5, 34, 36.5, 38, 39.5]);
       }
       const myFeatureGroup=L.featureGroup().addTo(this.map);
       myFeatureGroup.clearLayers();
       let marker, name, valor, code;
       this.spinnerService.show();
       this.http.get<StationValue[]>('http://localhost:4200/rest/
       daily?variable='+form.value.variable + '&date=2010-10-11')
       .subscribe(response => {
            const self = this;
            for (let i = 0; i < response.length; i++) {
                 valor=response[i].value;
                 marker=L.circle([response[i].latitude,response[i].longitude],
                 { color: scale(response[i].value).hex(), fillColor: '#f03',
                 Opacity: 0.5, radius: 10 })
                .on('click', function () {
                        name = response[i].name;
                        code = response[i].code;
                        self.openDialog(self,name, code);
                }).addTo(myFeatureGroup);
            }
            this.spinnerService.hide();
        });
}
```

Esta función, dependiendo del tipo de variable que reciba, mostrará los marcadores de un color u otro dependiendo del valor recogido. Esta funcionalidad ha sido implementada haciendo uso de la librería chroma.js [9], que nos permite obtener un color de un rango de colores acorde a su posición en un array predefinido de valores.

Los marcadores que se vayan a mostrar serán añadidos a un featureGroup, que se añadirá directamente al mapa y nos permitirá tratarlos como a un conjunto. Para obtener los datos requeridos ha sido necesario lanzar una petición get contra el servicio creado en el controlador, que nos retornará un array con los datos de la estación y el valor recogido. Esto ha sido posible gracias a la herramienta HTTPClient [10] que nos proporciona Angular, que facilita las peticiones HTTP con los servicios backEnd.

Dentro de esta petición, recorremos la respuesta creando uno a uno los marcadores con sus características propias y los vamos añadiendo al featureGroup creado anteriormente. En nuestro caso, cada marcador será un círculo, pudiendo ser otro tipo de figura como un icono o cualquier forma predefinida.

A la hora de crear cada marcador, le indicamos que cuando a este se le haga clic, abra una ventana emergente haciendo una llamada al método `openDialog`, pasándole como parámetro el identificador y código de la estación del marcador pulsado, y una referencia al componente principal.

Como se puede ver, una línea antes de realizar la petición http, hacemos uso de la función `show()` sobre la variable spinnerService, que contiene al objeto Ng4LoadingSpinnerService [11], que nos permitirá mostrar un icono de carga cuando el sistema este cargando los marcadores en el mapa. Este icono se ocultará una vez se complete la carga de datos haciendo uso del método `hide().`

A continuación, se muestra el fragmento de código HTML que permite mostrar el icono de carga:

HTML

```
<ng4-loading-spinner [threshold]='0'> </ng4-loading-spinner>
```

El parámetro *threshold* lo que nos indica es el tiempo mínimo que se mostrará el *spinner*. Ha sido necesario añadir este atributo, ya que el valor por defecto era lo suficientemente grande para que el usuario visualizase el mapa sin que se aún se hubieran cargado los marcadores en el mapa.

En las figuras 15 y 16 se muestra la visualización de la aplicación durante la carga de los marcadores y la visualización de estos sobre el mapa.

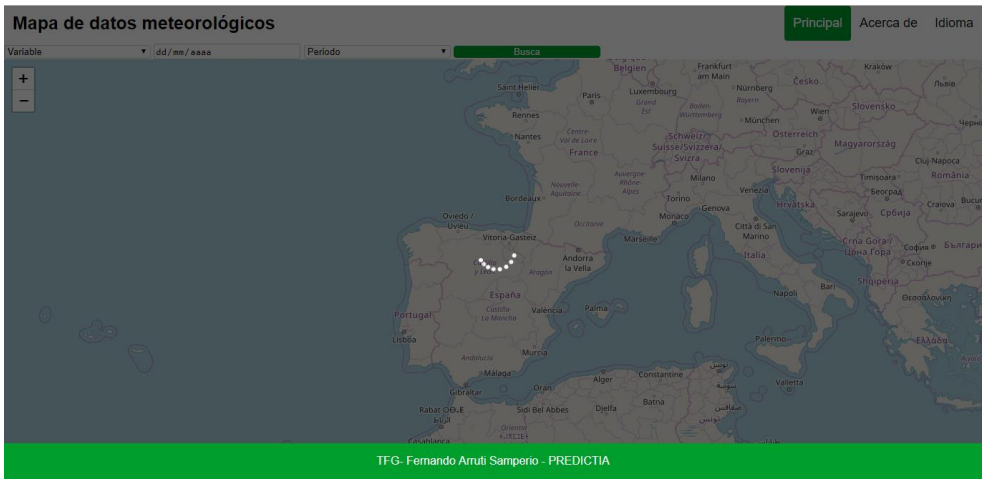

Figura 15: Estado durante la carga de los marcadores

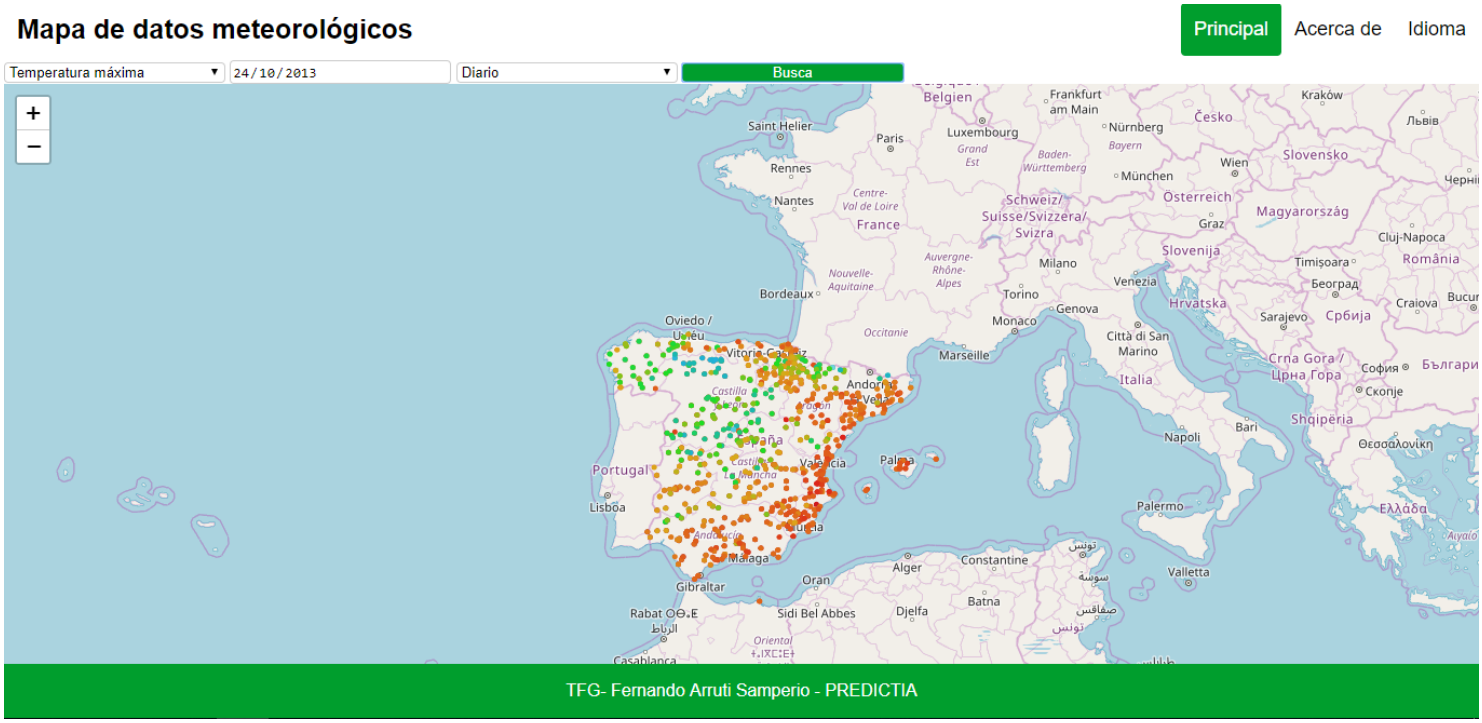

Figura 16: Visualización de los marcadores sobre el mapa

Ahora, se procede a especificar el funcionamiento del sistema cuando el usuario clique sobre uno de los marcadores mostrados.

En el siguiente recuadro, se detalla la implementación del método `openDialog`, que será activado una vez el usuario pulse sobre uno de los marcadores mostrados.

TypeScript

```
openDialog(self, name, code) {
      console.log('entra a abrir dialog: ' + self.dialog);
      this.name = name;
      this.code = code;
      const dialogRef = self.dialog.open(DialogComponent,{
      //abre ventana emergente con detalles estacion
            width: '100%',
            height: '90%',
```

```
                data: {
                        name: this.name,
                }
        });
        dialogRef.afterClosed().subscribe(result => {
                console.log('The dialog was closed');
        });
}
```

Esta función, abre un nuevo componente de la aplicación como una ventana emergente del tamaño que le asignamos al invocarla. A esta ventana le pasamos como parámetro el nombre, código y detalles de la estación, y se solapa encima del componente principal. Para realizar esta funcionalidad, fue necesario hacer uso del componente *MatDialog* [12], que nos permite abrir nuevas ventanas haciendo transferencias de datos entre unas y otras.

Una vez hecho esto, se implementó el Dialog que se recoge en el script siguiente, en el cual se muestran los datos de la estación y gráficas a partir de los datos recogidos por esta.

En el siguiente recuadro, se muestra tanto la inicialización del Dialog como su plantilla HTML.

TypeScript

```
export class DialogComponent implements OnInit {
        ngOnInit(): void {
                this.getData();
        }
        constructor(private http: HttpClient, public dialogRef: MatDialogRef
        <DialogComponent>, @Inject(MAT_DIALOG_DATA) public data: any, translate:
         TranslateService)    {
                this.translate = translate;
                translate.setDefaultLang('es');
                translate.use('es');
        }
       getData(): void {
                this.http.get<StationData>('http://localhost:4200/rest/
                getStationData?code=' + this.data.code).subscribe(
                response => {
                this.name = response.name;
                this.code = response.code;

          //obtencion de la respuesta con las caracteristicas de la estación
          //mismo proceso para todos los valores retornados
                });
        }
}
```

HTML

```
<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
   <h1 mat-dialog-title> {{ 'Details of the station' | translate }}</h1>
   <div mat-dialog-content>
       <p>
```

```
        {{ 'Selected station' | translate }}: {{name}}, {{ 'Code' | translate }}:
        {{code}}
        </p>
        <p>{{ 'Latitud' | translate }}: {{latitude}}, {{ 'Longitude' | translate
            }}: {{longitude}}, {{ 'Altitude' | translate }}: {{altitude}}
        </p>
        <p> {{ 'Provider' | translate }} : {{provider}}
        </p>
        <p>{{ 'Start date' | translate }}: {{startDate}}, {{ 'End date' |
        translate }}: {{endDate}}
        </p>
        <p>{{ 'Percentage of days without data' | translate }}: {{missingNumber}}
        </p>
        <p>
        {{'Values Collected' | translate }}: {{temperatureMax}},
        {{temperatureMin}}, {{temperatureMean}}, {{precipitation}}
        </p>
    </div>
    <div>
        <form id="formDates" #dataForm="ngForm" >
              <label for="date">{{ 'Start date' | translate }}: </label>
                    <input type="date" name="dateIni" id="dateIni" min="1950-01-
                    01" max="2015-12-31" (change)=" makeHistogram (dataForm)"
                    ngModel>
              <label for="date">{{ 'End date' | translate }}: </label>
                     <input type="date" name="dateEnd" min="1950-01-01"
                     max="2015-12-31"id="dateEnd" (change) =
                    "makeHistogram(dataForm)"ngModel>
        </form>
    </div>
    <mat-tab-group md-dynamic-height>
    <mat-tab label="Histograma">
       <div id="histogram"> <!-- Histogram will appear here --> </div>
    </mat-tab>
    <mat-tab label="Grafica anual">
    <!-- Month selector -->
    <div class="tab-content">
    <label>Select month</label>
        <ng-select [items]="arrayMonth"
                      bindLabel="name"
                      bindValue="engName"
                      [multiple]="true"
                      placeholder="Select months"
                      [(ngModel)]="selectedMonthNames"
                      (change)="makeMonthlyDataGrafic()">
        </ng-select>
    </div>
    <div id="yearGraphic" style="height: 400px;">
        <!-- anual graph will appear here -->
    </div>
    </mat-tab-group>
    </body>
</html>
```

Como se puede leer en el código TypeScript del componente del Dialog, este recibe inyectado en el constructor los valores que le pasamos en la función que se activa al clicar sobre un marcador. Esto es, el código de la estación marcada.

En la función `ngOnInit`, que se ejecuta automáticamente al entrar al componente, hacemos la llamada a la función `getDate`, que realiza una llamada al servicio que nos retorna las características de la estación con el código pasado como parámetro.

En la primera sección del código HTML, mostramos por pantalla estos valores devueltos por el servicio. En la segunda sección, tenemos un formulario en el que el usuario deberá introducir dos fechas, que nos servirán para mostrar gráficas o tablas entre esos dos puntos temporales. Cuando el usuario introduce una fecha, se hace una llamada a la función `makeHistogram`, cuya implementación se detallará más adelante. Esta función nos muestra por pantalla un histograma de los datos.

En las secciones `<mat-tab>` [13] tendremos las pestañas con las opciones disponibles en nuestra aplicación, como son las gráficas o tablas mostradas acorde a una serie de parámetros.

En la Figura 17 se muestra como es el aspecto del Dialog una vez clicamos sobre un marcador.

Figura 17: Vista de los detalles de la estación

El siguiente fragmento de código, es el encargado de generar el histograma a partir de las dos fechas introducidas en el formulario.

TypeScript

```
makeHistogram(form: NgForm): void {
   if (form.value.dateIni !== '' && form.value.dateEnd !== '') {
       this.initialDateGraph = form.value.dateIni;
       this.endlyDateGraph = form.value.dateEnd;
       const histogramGraph = document.getElementById('histogram');
       const yearGraphic = document.getElementById('yearGraphic');
       if(document.body.contains(histogramGraph)){
             this.getAxisHistogramGrafic(form.value.dateIni,form.value.dateEnd
             , this.data.code);
             }else
             if (document.body.contains(yearGraphic) &&
                 this.selectedMonthNames != null)
             {
```

```
                this.getAxisGraficMonthly(this.initialDateGraph,
                this.endlyDateGraph, this.code);
            }
        }
}

getAxisHistogramGrafic(iniDate, endDate, code) {
        const promise = new Promise((resolve, reject) => {
            this.http.get<MonthlyValue[]>('http://localhost:4200/rest/getValu
            esBetween?stationName='+code+'&startDate='+iniDate+'&endDate=' +
            endDate).toPromise().then(response =>
            {
                this.makeGraphic(response);
                resolve();
            });
        });
}

makeGraphic(response) {
        this.precipitations.length = 0;
        this.temperaturesMax.length = 0;
        this.temperaturesMin.length = 0;
        this.temperaturesMean.length = 0;
        for (let i = 0; i < response.length; i++) {
            if (response[i].variable === 'PRECIPITATION') {
                this.precipitations.push(response[i].value);
            }
            if (response[i].variable === 'TEMPERATURE_MAX'){
                this.temperaturesMax.push(response[i].value);
            }
            if (response[i].variable === 'TEMPERATURE_MIN'){
                this.temperaturesMin.push(response[i].value);
            }
            if (response[i].variable === 'TEMPERATURE_MEAN') {
                this.temperaturesMean.push(response[i].value);
            }
        }
        for (let i = 0; i < this.arrayMonthAndYear.length; i++) {
            this.months.push(this.arrayMonthAndYear[i].name);
        }
        const dataTmax = {
            x: this.months,
            y: this.temperaturesMax,
            name: 'temperatura maxima',
            type: 'scatter', yaxis: 'y2'
        };
        const dataTmin = {
            x: this.months,
            y: this.temperaturesMin,
            name: 'temperatura minima',
            type: 'scatter', yaxis: 'y2'
        };
        const dataTmean = {
            x: this.months,
            y: this.temperaturesMean,
            name: 'temperatura media',
            type: 'scatter',
            yaxis: 'y2'
        };
        const dataPr = {
            x: this.months,
            y: this.precipitations,
            name: 'precipitaciones',
            type: 'bar'
        };
        const data = [dataPr, dataTmean, dataTmin, dataTmax];
        const layout = {
            title: 'histograma estacion',
```

```
            width: 800,
            yaxis: {
               title: 'precipitations',
               titlefont: { color: '#1f77b4' }, tickfont: { color: '#1f77b4' }
            },
            yaxis2: {
               title: 'temperaturas', titlefont: { color: '#ff0000' },
               tickfont: { color: '#ff7f0e' },
               anchor: 'x',
               overlaying: 'y',
               side: 'right',
            }
      };
      Plotly.purge('histogram'); // refresh in case graphic exist
      Plotly.plot('histogram', data, layout);
}
```

La función `makeHistogram`, recibe los datos del formulario de fechas, y solo actúa en el caso de que el usuario haya introducido las dos fechas requeridas. La función dibujará una gráfica u otra en función de la pestaña sobre la que se encuentre a la hora de introducir la última de las dos fechas. Este fragmento de código solo contiene dos de los casos, siendo el mismo proceso para cada una de las pestañas existentes.

El caso en el que el usuario se encuentra sobre el histograma, la función llamará al método `getAxisHistogramGrafic`, pasándole como parámetro las fechas del formulario y el código de la estación. Esta función, realiza una petición al servicio para obtener los datos a dibujar en la gráfica. Ha sido necesario introducir dentro de la petición al servicio la llamada al método `makeGraphic,` debido a que, de otra manera, esta función realizaba la impresión del gráfico sin tener aún los datos disponibles.

Esta función `makeGraphic`, haciendo uso de la herramienta *Plotly*, nos mostrará en el lugar determinado para ello un gráfico en el que el eje X se configurará con los meses del año, y en el eje Y tendremos 2 ejes, uno para temperaturas y otro para precipitaciones. Los datos relativos a precipitaciones se mostrarán en forma de barra y los datos relativos a temperaturas se representarán con una línea continua entre principio y final de año, una línea por variable.

En las figuras 18 y 19 se muestran los dos tipos de gráfico creados con *Plotly* en nuestra aplicación, uno en el que el eje X represente los meses del año y otro en el que el eje X represente los años desde los que hay datos.

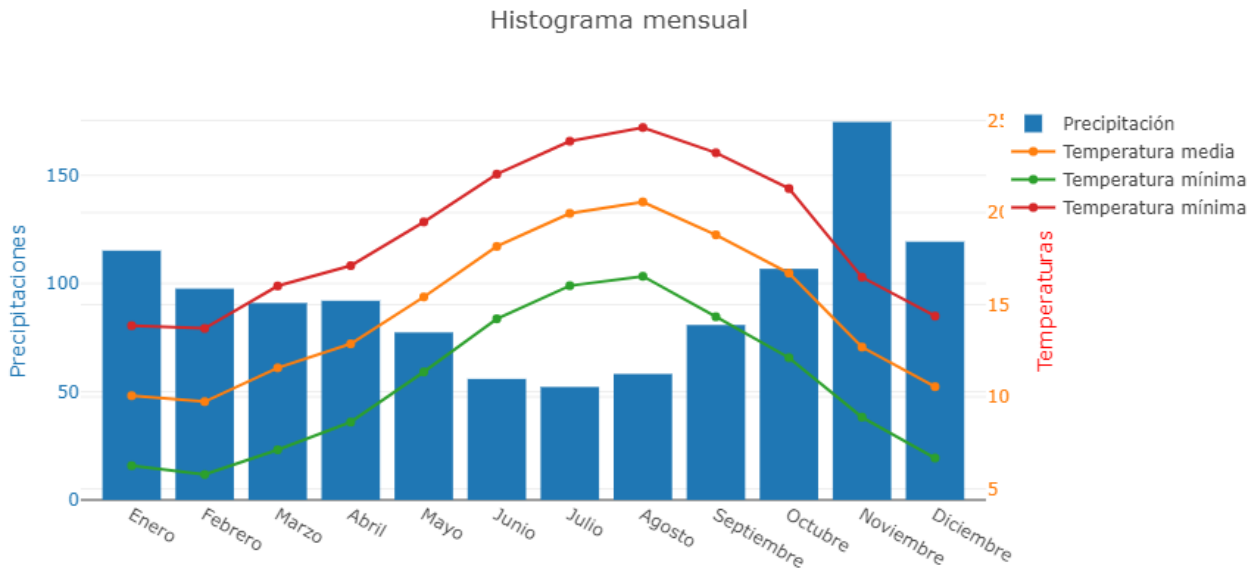

Figura 18: Histograma mensual

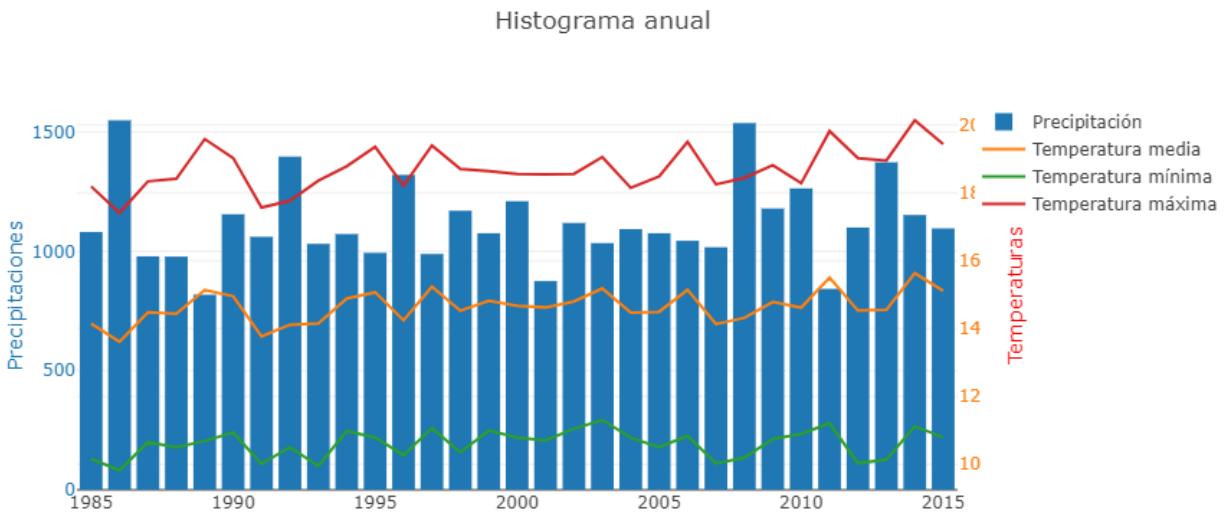

Figura 19: Histograma anual

Una vez detallada la creación de gráficas, la otra funcionalidad para visualizar datos en la aplicación, son las tablas, que se encuentran en la pestaña “Datos estadísticos”.

En la Figura 20, se muestra como es la visualización de estos datos en la aplicación.

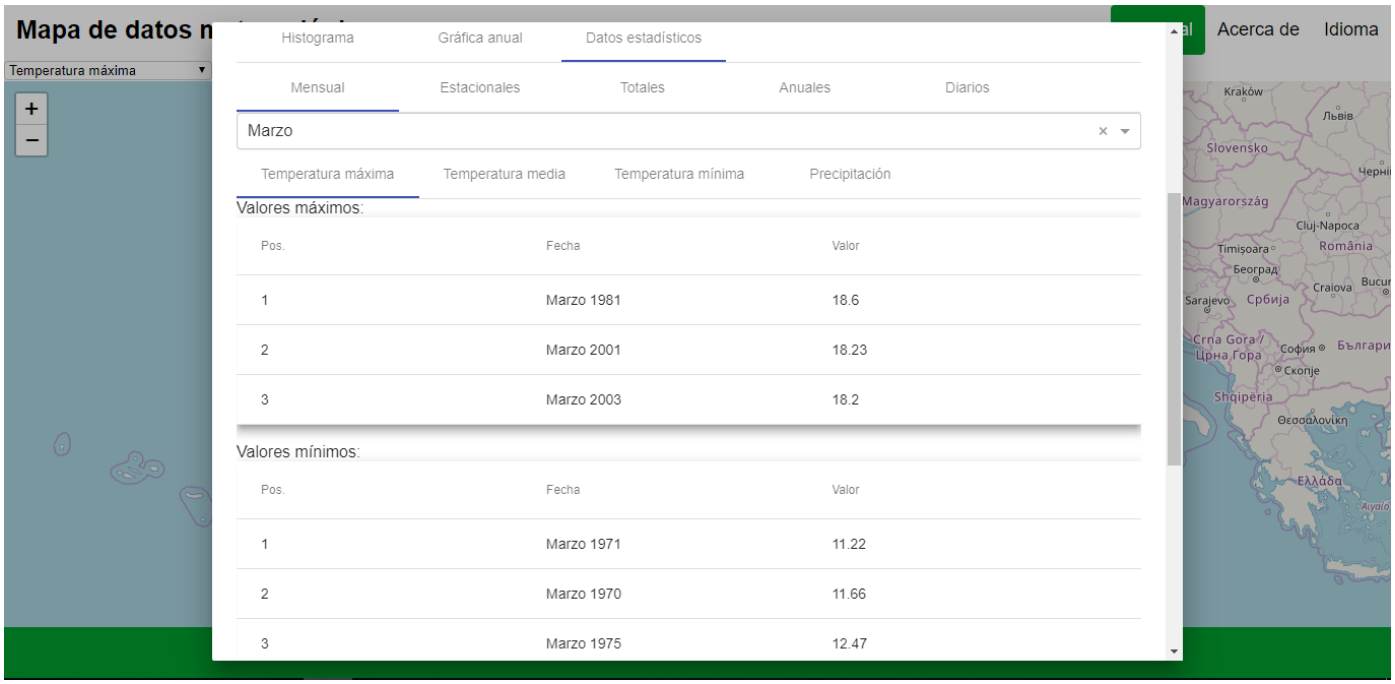

Figura 20: Visualización de valores estadísticos

La aplicación dispone de cinco opciones de visualización de valores máximos y mínimos recogidos: Datos mensuales, estacionales, totales, anuales y diarios. Cada una de estas opciones, estará subdivida a su vez en cuatro pestañas, una por variable recogida.

El usuario podrá visualizar en diferentes tablas los valores máximos y mínimos recogidos. En la Figura 20, por ejemplo, se visualizan los meses de marzo que mayor temperatura máxima media han tenido en una tabla y en otra los que menor temperatura máxima media se ha recogido. Esto es así para cualquiera de las cuatro opciones disponibles: Temperatura máxima, mínima, media y precipitaciones

A continuación, se detalla la implementación para mostrar las tablas por pantalla.

TypeScript (Componente Dialog)

```
getYearTable() {
   this.loadingSpinner = true;
   this.loadingTable = true;
   const promise = new Promise((resolve, reject) =>
       {
        this.http.get < RankValue[] >
           ('http://localhost:4200/rest/readTopStatisticsOfYear?year='
           + this.selectedYear + '&stationCode=' + this.code + '&n=3&variable='
           + this.yearTabActive) .toPromise()
           .then(response => {
                if (response.length === 0) {
                      this.openDialog(this, 'NoData', 'NoDataContent');
                      this.loadingSpinner = false;
                } else {
                      this.loadingTable = false;
                      this.loadingSpinner = false;
                      this.yearTableMaximumValues = new MatTableDataSource(
                               response.filter(res => res.type === 'TOP3'));
                      this.yearTableMinimumValues = new MatTableDataSource(
                      response.filter(res => res.type === 'BOTTOM3'));
                 }
```

```
    });
}
```

HTML (Componente Dialog)

```
<div class="spinner-container">
<mat-spinner [style.display]="loadingSpinner ? 'block' : 'none'" style="margin:0
auto;" diameter = "300">
</mat-spinner>
</div>

<app-tablestats [maximumTableValues]="yearTableMaximumValues"
[minimumTableValues]="yearTableMaximumValues" [style.display]="loadingTable ===
false ? 'block' : 'none'">
</app-tablestats>
```

TypeScript (Componente tablestats)

```
export class TablestatsComponent implements OnInit {
       @Input() maximumTableValues: MatTableDataSource < RankValue > ;
       @Input() minimumTableValues: MatTableDataSource < RankValue > ;
        displayedColumns = ['position', 'date', 'value'];
        constructor() {}
        ngOnInit() {}
}
```

HTML (Componente tablestats)

```
<div>{{ 'Maximum values' | translate }}: </div>
<mat-table class="rank-table-month mat-elevation-z8"
          [dataSource]="maximumTableValues">
   <ng-container matColumnDef="position">
      <mat-header-cell *matHeaderCellDef>Pos.</mat-header-cell>
      <mat-cell *matCellDef="let rankValue">{{rankValue.position}}</mat-cell>
   </ng-container>
   <ng-container matColumnDef="date">
  <mat-header-cell *matHeaderCellDef>{{ 'Date' | translate }}</mat-header-cell>
      <mat-cell *matCellDef="let rankValue">{{rankValue.date}}</mat-cell>
   </ng-container> <ng-container matColumnDef="value">
 <mat-header-cell *matHeaderCellDef>{{ 'Value' | translate }}</mat-header-cell>
 <mat-cell *matCellDef="let rankValue">{{rankValue.value}}</mat-cell>
 </ng-container>
 <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
 <mat-row *matRowDef="let row; columns: displayedColumns"></mat-row>
</mat-table>
// mismo caso tabla de valores minimos
```

Para obtener los valores máximos y mínimos, se realiza una llamada al servicio, y los valores obtenidos de esta llamada se almacenan en las tablas del tipo MatTableDataSource [14] que contiene el componente Dialog. Estas tablas se mostrarán una vez se haya completado la carga, cuando la variable *loading* table pase a ser false.

Para mostrar estas tablas, se hará uso del componente creado para ello, el componente "Tablestats". Este componente recibe como entradas los valores obtenidos en el servicio, e indica que se muestren tres columnas: posición, fecha y valor.

6. Pruebas

En este apartado, se detallan las pruebas que se han llevado a cabo para determinar que el funcionamiento del código generado es el correcto.

6.1. Unitarias

Las pruebas unitarias nos permiten comprobar que el funcionamiento de cada módulo por separado es el correcto.

Para comprobar que el funcionamiento del código es el adecuado, se han aplicado pruebas mediante la técnica de caja negra a cada uno de los módulos de los que se compone el sistema. Esta técnica consiste en verificar la funcionalidad del sistema sin tener en cuenta la estructura interna del código, únicamente comprobando si para unas determinadas entradas, se producen unas determinadas salidas, tal y como se puede observar en la Figura 21.

Figura 21: Técnica de caja negra

Estas pruebas fueron realizadas de forma manual en cada módulo, identificando las particiones de equivalencia de cada módulo y comprobando si para los posibles escenarios de entrada, se producen las salidas esperadas.

6.2. Integración

Las pruebas de integración son aquellas que permiten comprobar que la interacción entre los distintos componentes de los que se compone sistema es la correcta.

A continuación, se detallan brevemente las pruebas de integración realizadas de forma manual:

- Funcionamiento correcto: El sistema muestra al usuario los componentes esperados en su interacción con el sistema en cada una de las opciones disponibles en este.

- Búsqueda sin resultados: Se ha comprobado en cada una de las búsquedas disponibles en el sistema, que se notifica al usuario cuando no se disponen de datos sobre la búsqueda realizada.

- Petición al servidor cuando este está fuera de servicio: Se ha comprobado que el sistema notifica que el servidor no se encuentra disponible cuando se le lanza una petición desde la aplicación.

6.3. Aceptación

Las pruebas de aceptacion son aquellas en las que el cliente comprueba que el sistema se ajusta a lo requerido y funciona correctamente.

En cada iteración del proyecto, se ha ido comprobando con el cliente que los requisitos iniciales planteados al inicio de la iteración estaban plasmados en el producto. Se ha comprobado fundamentalmente que el funcionamiento era el esperado, los tiempos de respuesta se ajustaban a los mínimos exigidos y la configuración de la interfaz era la deseada e impedía al usuario cometer errores cuando se le solicita introducir cualquier tipo de valor.

7. Análisis de calidad

Para garantizar que el código generado se ajustaba a unos mínimos estándares de calidad, se ha hecho uso de la herramienta *SonarQube* para corregir defectos y mejorar el sistema.

En la Figura 22, se muestran los resultados obtenidos en el primer análisis de la parte del proyecto implementada en *Java*.

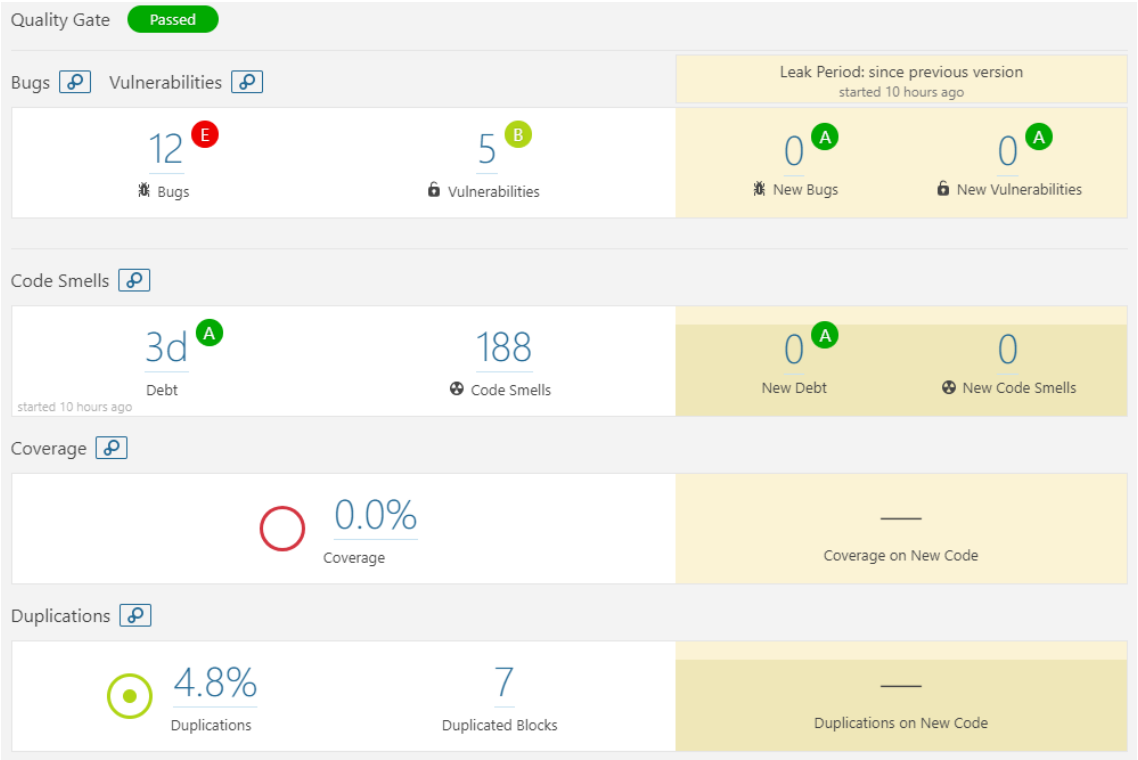

Figura 22: Primer análisis de código Java con Sonar

Como se puede observar, la métrica de calidad aparece como superada, por lo que, para mejorar aún más el código se procedió a solucionar los Bugs, vulnerabilidades y los "Code Smells" más destacados, así como intentar reducir código duplicado.

El resultado de aplicar estas correcciones y refactorizaciones propuestas por Sonar se puede observar en la Figura 23. Se ha conseguido eliminar todos los Bugs y vulnerabilidades, así como reducir en un día la deuda técnica y un número significativo de "Code Smells".

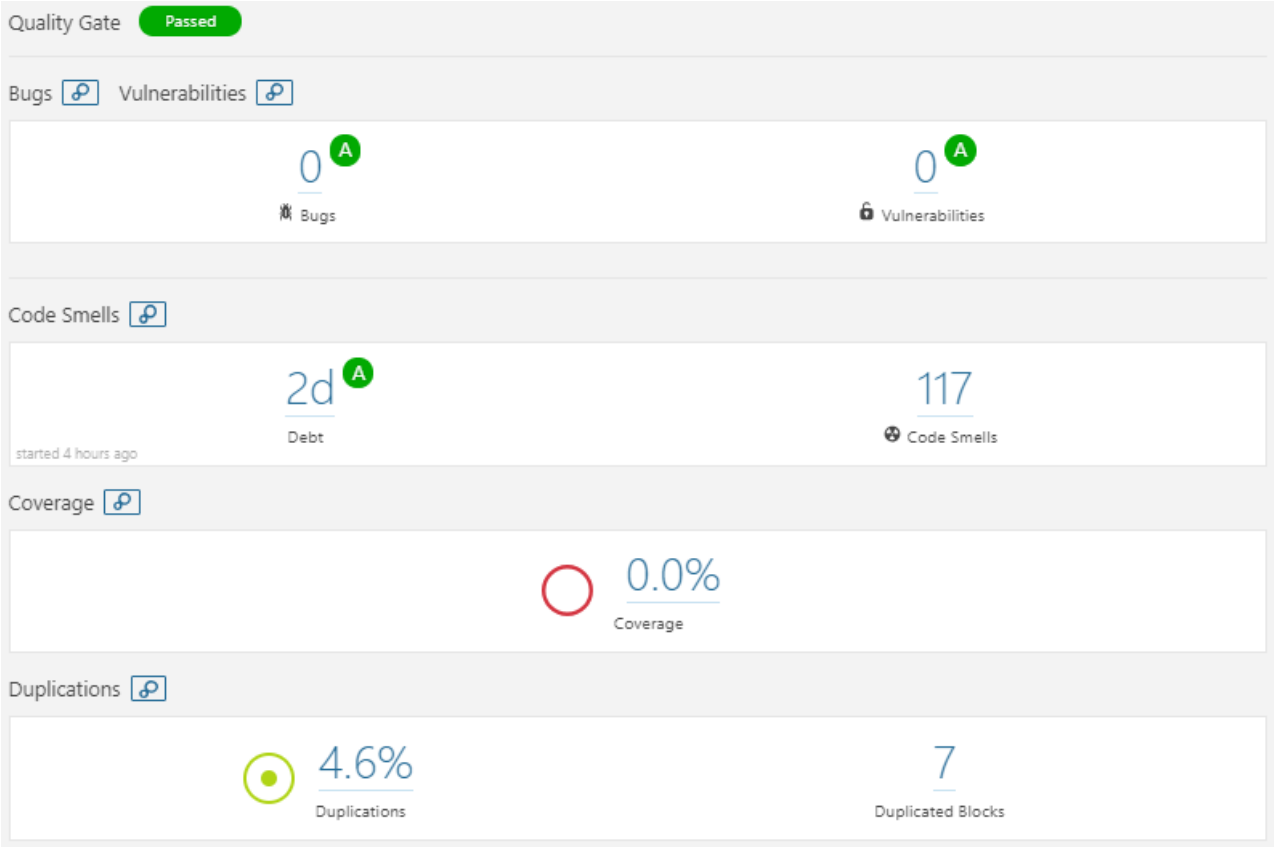

Figura 23: Último análisis de código Java con Sonar

En la Figura 24, se muestran los resultados obtenidos al aplicar con *Sonar* un análisis de código sobre la parte del proyecto desarrollada en *Angular*.

Como se puede observar, el resultado de este análisis fue satisfactorio, al no detectar ni Bugs ni vulnerabilidades, teniendo únicamente un leve número de "Code Smells". La deuda técnica fue de solamente dos horas.

El número de código duplicado detectado por la herramienta fue elevado, debido al fichero de configuración webpack.config.js, como se puede observar en la Figura 25. El otro fichero con código duplicado ha sido el fichero dialog.component.ts, teniendo casi el nueve por ciento de código duplicado.

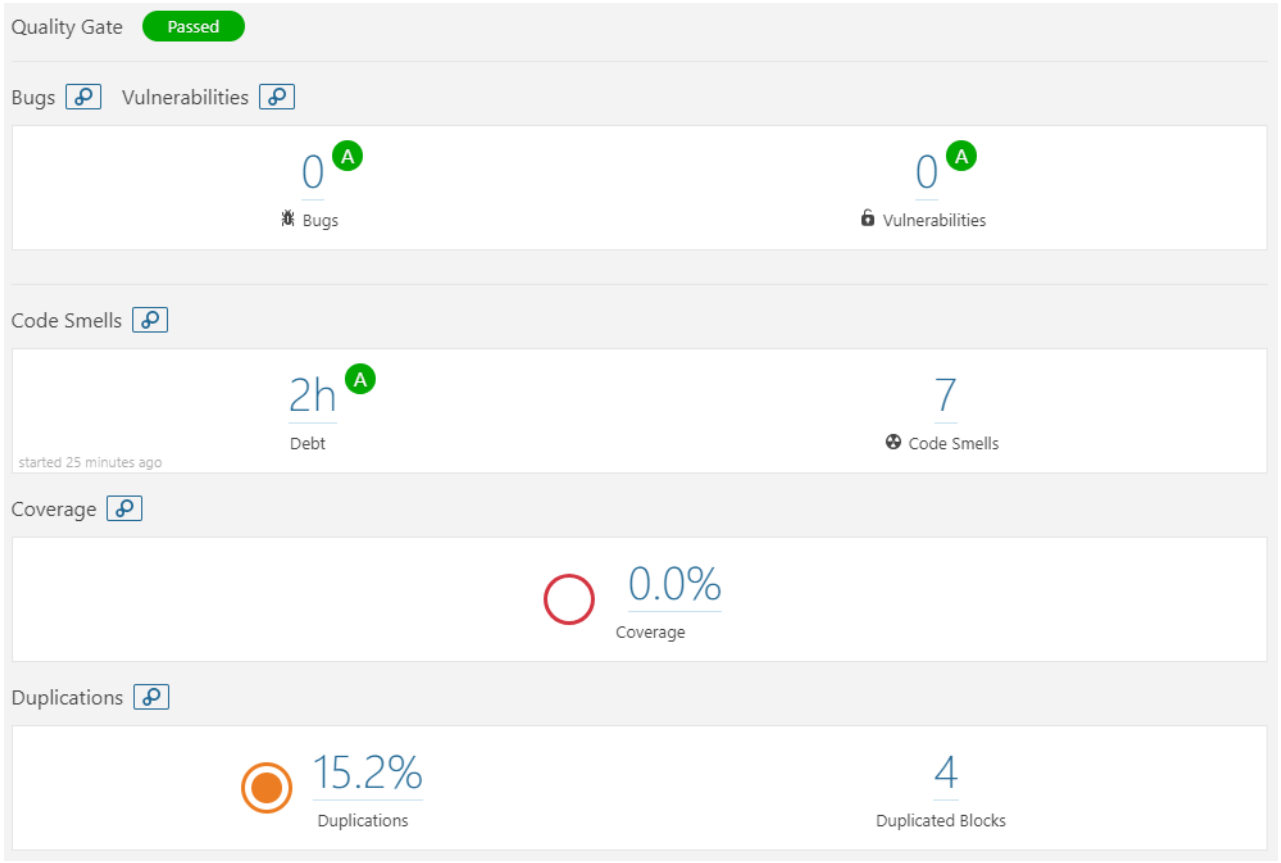

Figura 24: Primer análisis de la parte Angular

Figura 25: Número de líneas de código duplicadas por fichero detectadas

Para mejorar el código, se procedió a eliminar los “Code Smells” existentes. El resultado de aplicar estos cambios se puede observar en la Figura 26, donde se indica que la deuda técnica ha sido reducida a cero y han sido eliminados los “Code Smells” existentes.

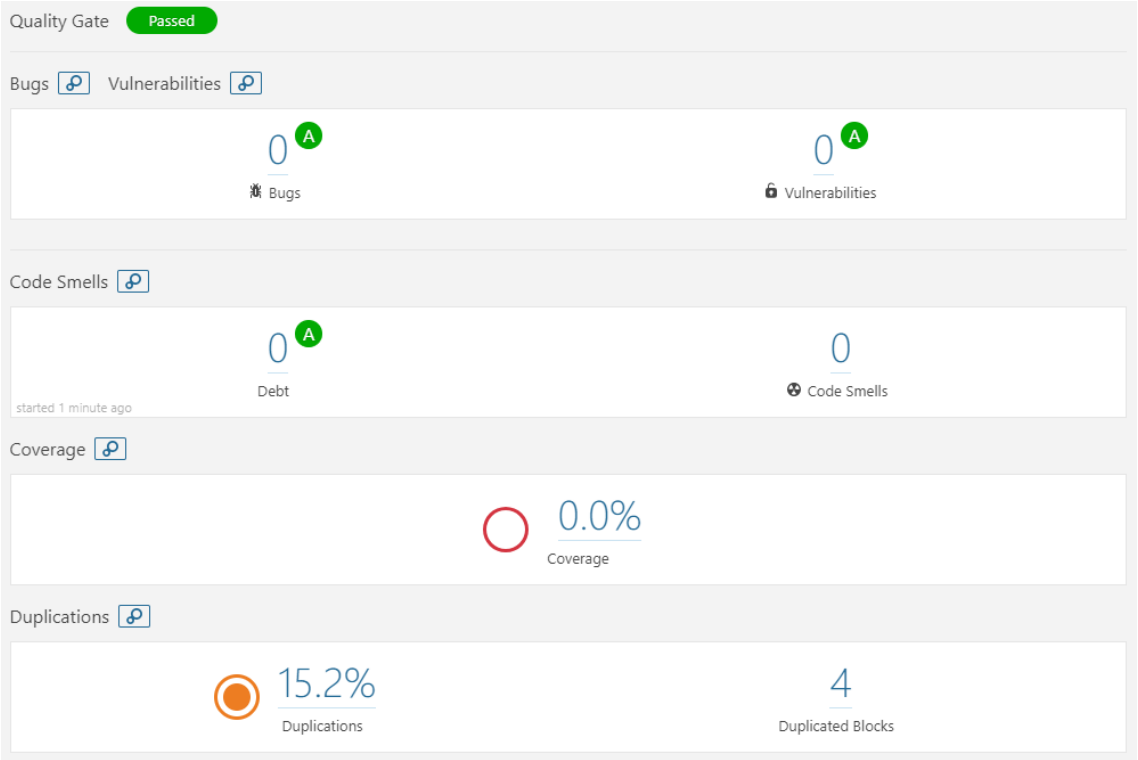

Figura 26: Último análisis de la parte de Angular con Sonar

8. Conclusiones y trabajos futuros

En este apartado, se detallan las conclusiones obtenidas tras la elaboración del proyecto, así como las posibles ampliaciones que se pueden realizar al mismo.

8.1. Conclusiones

El presente proyecto tiene como finalidad la elaboración de un sistema Web de consulta y visualización de los datos existentes en el registro histórico de observaciones recogidas por la red secundaria de estaciones de AEMET.

El proyecto ha cumplido con los requisitos planteados al inicio del desarrollo de este, encontrándose al acabar el desarrollo una versión completamente funcional en modo local.

El desarrollo del proyecto de manera iterativa ha sido idóneo para poder comprobar el funcionamiento del sistema durante el proceso, así como realizar el aprendizaje de las nuevas herramientas empleadas de una manera escalonada.

En cuanto a los conocimientos adquiridos, este proyecto me ha servido para aprender nuevas herramientas y lenguajes, destacando especialmente la herramienta Angular, que permite el desarrollo de aplicaciones web repletas de funcionalidad de una manera sencilla y lejos de la complejidad de otros lenguajes como pueden ser JavaScript o PHP.

El desarrollo de todo el ciclo de vida del sistema, desde su concepción a su completa implementación, documentación y prueba me ha permitido poner en práctica todos los conocimientos adquiridos durante la carrera

Además, la estancia en la empresa PREDICTIA durante los meses de prácticas allí, me han sido de mucha ayuda, tanto para reforzar conocimientos, así como adquirir mayor destreza y competencia en el ámbito de la profesión informática.

8.2. Trabajos futuros

Cumplidos los requisitos establecidos inicialmente, el sistema desarrollado puede ser extendido con nuevas funcionalidades.

A continuación, se listan y se detallan brevemente estas:

1 Tratamiento de datos a una escala diferente a la escala diaria, ya pueda ser mensual, estacional o anual.

2 Presentación de diferentes valores estadísticos en el cliente, como pueden ser la mediana o percentiles.

3 Incorporación de datos procedentes de otras estaciones.

4 Mejora de detalles estéticos en la capa de presentación, como pueden ser la forma de los iconos o la vista de detalle de las estaciones.

Referencias

[1] AEMET. Registros climáticos: http://www.aemet.es/es/idi/clima/registros_climaticos . Última vez visto: 19/06/2018

[2] Aurum Solutions. Modelos de desarrollo iterativos: http://aurumsol.com/espanol/articulos/art1/art1-4.html , Última vez visto: 19/06/2018

[3] Plotly. Modern Visualization for the Data Era: https://plot.ly/ , Última vez visto: 19/06/2018

[4] Pivotal Software. Spring: the source for modern java: https://spring.io/ , Última vez visto: 19/06/2018

[5] Unidata. An Introduction to NetCDF: https://www.unidata.ucar.edu/software/netcdf/docs/netcdf_introduction.html, Última vez visto: 19/06/2018

[6] Vladimir Agafonkin. https://leafletjs.com/examples/quick-start/ , Última vez visto: 19/06/2018

[7] Google. One framework. Mobile & desktop.: https://angular.io/ , Última vez visto: 19/06/2018

[8] BabelEdit. The internationalization (i18n) library for Angular: https://github.com/ngx-translate/core , Última vez visto: 19/06/2018

[9] gka. Chroma.js: https://gka.github.io/chroma.js/, Última vez visto: 19/06/2018

[10] Google. HttpClient: https://angular.io/guide/http , Última vez visto: 19/06/2018

[11] Amit Mahida. ng4-loading-spinner: https://www.npmjs.com/package/ng4-loading-spinner , Última vez visto: 19/06/2018

[12] Google. Dialog: https://material.angular.io/components/dialog/api , Última vez visto: 19/06/2018

[13] Google. MatTab: https://material.angular.io/components/tabs/overview , Última vez visto: 19/06/2018

[14] Google. MatTable: https://material.angular.io/components/table/overview , Última vez visto: 19/06/2018