Facultad
de
Ciencias

Monitorización de Datos Abiertos de Sensores Ambientales
(Environmental Sensors Open Data Monitoring)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jorge Pereda González

Director: Pablo Sanchez Barreiro

Julio – 2021

Resumen

Este Trabajo de Fin de Grado ha consistido en el desarrollo de una aplicación móvil la cual hace uso de información de sensores ambientales distribuidos por la ciudad de Santander. Esa información sobre sensores ambientales se obtiene del portal de datos abiertos del ayuntamiento de Santander. Concretamente, los sensores ambientales nos proporcionan información sobre la temperatura, la luminosidad y el ruido de distintos puntos de la ciudad.

El objetivo de esta aplicación es el de monitorizar esos datos y hacerlos más accesibles al usuario con varios fines como, por ejemplo, elegir una zona para la realización de ejercicio físico en la que no haga un calor excesivo o el nivel de ruido no sea muy molesto.

Para lograr ese objetivo, la aplicación permitirá al usuario elegir una serie de sensores de interés, crear alarmas que avisen en caso de que se superen ciertos valores establecidos o consultar estadísticas de los datos del sensor en un periodo de tiempo.

La aplicación se desarrollará para Android, para ello se seguirá una metodología ágil similar a Scrum.

Abstract

This Final Degree Project has consisted of the development of an mobile app which use information from environmental sensors distributed around the city of Santander. That information from environmental sensors is obtained from the open data web of the government of the city of Santander. Specifically, the environmental sensors provide us with information about the temperatura, brightness and noise in diferent spots of the city.

The goal of this app is monitor that data and make it more accessible to the user with purposes like choose the place where to do physical exercise with no excessive heat or the noise level don’t be very annoying.

To achieve that goal, the app will allow the user to choose some interesting sensors, create alarms that warn the user when some set values are exceded or check statistics of sensor data in a period of time.

The app will be developed for Android, so it will follow an agile methodology like Scrum.

1. Introducción, metodología y desarrollo.

En esta sección comentaremos de qué trata el proyecto, junto con los objetivos que persigue, para después hablar de qué metodología se ha llevado a cabo durante la realización del proyecto y de las principales herramientas de trabajo utilizadas.

1.1 Introducción.

En los últimos años se ha ido consolidando, por parte de diferentes organizaciones y administraciones, la sana práctica de publicar datos en abierto, conocidos como *open data*. Esto permite que terceras personas u organizaciones puedan analizar y procesar estos datos con diferentes fines, posibilitando así el desarrollo aplicaciones que exploten estos datos con un propósito concreto.

Dentro de estas organizaciones se encuentra el Ayuntamiento de Santander, el cual ha creado un portal de datos abiertos (http://datos.santander.es/) sobre diferentes aspectos de la ciudad, como, por ejemplo, la posición actual de su flota de autobuses o el estado de riego de sus jardines, entre muchas otras cuestiones.

Entre esos datos se encuentran los proporcionados por una serie de sensores ambientales que monitorizan variables como el nivel de luminosidad, ruido y temperatura de diferentes puntos de la ciudad. El objetivo de este Trabajo Fin de Grado es crear una aplicación que explote dichos datos con varios fines.

En primer lugar, se trataría de conocer el estado actual de las diferentes zonas de la ciudad de manera que podamos escoger aquella que consideremos más adecuada para la realización de un tipo de actividad concreta. Por ejemplo, si queremos dar un paseo tranquilo, podríamos analizar el estado de ruido de diferentes zonas de la ciudad y escoger aquella que en ese momento se encuentre en mayor silencio. De manera similar, se podrían analizar los valores de luminosidad para decidir qué rutas se consideran más seguras para realizar un trayecto de vuelta a casa, o qué zonas, por su temperatura, podrían ser más adecuadas para la práctica del deporte.

En segundo lugar, se monitorizarían ciertas zonas de la ciudad para analizar posteriormente con detalle su evolución. Por ejemplo, podríamos revisar los valores de ruido de diferentes zonas a lo largo de varias semanas antes de decidirnos por la zona donde preferiríamos comprar o alquilar una vivienda.

En tercer lugar, la aplicación podría también permitir establecer una serie de alertas que notifiquen cuándo en determinadas zonas de la ciudad se han generado valores dentro de un rango que se considere peligroso para sus habitantes. Por ejemplo, cuándo en una determinada zona de la ciudad se han excedido los valores máximos de ruido establecidos por las correspondientes normativas.

Con estas funcionalidades los ciudadanos podrían empezar a explotar en cierta forma los datos de temperatura, luminosidad y ruido que el Ayuntamiento de Santander pone de manera gratuita a disposición de sus ciudadanos. Una vez que los ciudadanos empiecen a explotar estos datos, estos mismos ciudadanos podrían empezar a demandar nuevos modos de uso de los mismos, los cuales podrían ir incorporándose a nuestra aplicación para hacerla crecer.

1.2 Metodología

Para el desarrollo del proyecto se optó por una metodología ágil fuertemente basada en *Scrum*[1]. Obviamente, al ser el proyecto desarrollado por una única persona y no un equipo de desarrolladores, ciertos aspectos de *Scrum* tuvieron que ser adaptados. Por ejemplo, no tenía sentido celebrar *Daily Scrum Meetings* con una única persona.

Se comenzó el desarrollo del proyecto con la creación de *Product Backlog* que contenía apenas media docena de historias de usuario. Este *Product Backlog* se introdujo en *ScrumDesk*, que fue la herramienta elegida inicialmente para la gestión del proyecto *Scrum*.

Inicialmente, se optó por *Sprints* de una duración semanal. Al comienzo de la semana, se mantenía una reunión con mi tutor del Trabajo Fin de Grado en la que se revisaba el trabajo hecho la semana anterior, que se correspondería con una *Product Review*, se definía el trabajo para la próxima semana, lo que correspondería con un *Spring Planning Meeting*.

Posteriormente, debido a la situación creada por la pandemia, se realizaron cambios en la metodología. Se siguieron realizando *Sprints* semanales con reuniones establecidas con mi tutor del Trabajo Fin de Grado, siempre de forma telemática. La herramienta *Scrumdesk* fue relegada a un segundo plano y continuamos realizando la gestión de las tareas de forma menos canónica respecto a *Scrum* y más informalmente mediante correo.

1.3 Herramientas de desarrollo

La aplicación ha sido desarrollada para Android y tiene como versión objetivo Android 10. Sin embargo, se ha definido una versión mínima necesaria de Android 7 para un uso correcto de la mayoría de funcionalidades principales.

El entorno de desarrollo usado ha sido Android Studio, en su versión 4.1.3. Durante el desarrollo de la app ha sido necesaria la utilización de librerías y APIs externas para el correcto funcionamiento de todas las funcionalidades. Una de ellas es la API de Google Maps la cual ha sido usada para mostrar mapas en la aplicación y poder indicar al usuario la posición real de cada sensor que está visualizando. También se ha utilizado *SQLite*[2], para implementar una pequeña base de datos local donde guardar información de las medidas tomadas para la generación de estadísticas de los sensores.

En cuanto a librerías, se ha utilizado una librería matemática llamada *commons-math* para agilizar el cálculo de estadísticas y también una librería llamada *TinyDB* la cual hace uso de la API *SharedPreferences* para guardar datos que son necesarios durante el flujo de la app pero que no son suficientemente complejos para requerir del uso de una base de datos.

La gestión de versiones se ha realizado mediante *Git*[1] [3], gestionado a través del propio *Android Studio* y manteniendo un repositorio centralizado. Este repositorio dispone de una rama principal *master* y una rama secundaria *develop* en la cual se realiza todo el trabajo antes de añadirlo a la rama *master*. De esta forma, en *master* sólo hay versiones estables de la aplicación, mientras que en *develop* se va almacenando el trabajo temporal necesario para ir implementando cada historia de usuario. Dado que el proyecto se desarrollaba por una única

[1] Repositorio GitLab: https://gitlab.com/jorgepereda1989/SensoresSantander

persona y sólo se trabajaba en una historia de usuario a la vez, por lo general, no se crearon ramas adicionales a estas dos.
Con el fin de controlar la calidad de la aplicación, aunque ya funcionara, se ha realizado un control de calidad llevado a cabo con la aplicación *Sonarqube*[4].

2. Especificación de requisitos.

Esta sección describe el proceso de Ingeniería de Requisitos llevado a cabo para el desarrollo de este proyecto. Inicialmente se describe el contexto en el que se desarrolla el proyecto para después hablar de la captura de requisitos que nos conduce al modelo de objetivos que se pretenden alcanzar y a las historias de usuario principales que se originan. Para terminar, se especifica qué requisitos no funcionales requiere satisfacer la aplicación.

2.1 Contexto

Tal como se ha comentado en la introducción, el Ayuntamiento de Santander pone a disposición de los ciudadanos diversos conjuntos de datos, algunos de ellos muy útiles e interesantes como *Puntos de venta/recarga TUS*, *Cámaras de tráfico*, *Sensores de Parking de Superficie*, *Estaciones de bicicletas*, *Señales de paradas de Taxi*, entre otras muchas.

Estos conjuntos de datos son accesibles de múltiples maneras, siendo las más comunes mediante ficheros de tipo HTML, JSON, XML o CSV. Algunos de ellos disponen de más formatos que otros, dependiendo del tipo de datos que ofrece.

Cómo y para qué usar estos conjuntos de datos queda a disposición del usuario. Algunos ejemplos de aplicaciones construidas sobre estos datos serían:

- Aplicación *TUS Santander* en AppStore. Hace uso de un dataset de estimaciones generado por el TUS para mostrar en su app todas las paradas de TUS con sus estimaciones de llegada.
- Aplicación *Vive Santander. Turismo, ocio y transporte* en GooglePlay. Usando varios datasets de recursos culturales de la ciudad crea una app con todo tipo de eventos, lugares típicos, monumentos o restaurantes, para cualquier usuario que quiera conocer más a fondo la ciudad.

En nuestro caso hacemos uso del dataset *Sensores Ambientales*, el cual nos ofrece datos de sensores establecidos en distintos puntos de la ciudad. Estos sensores disponen entre sus datos de batería del sensor restante (porcentaje), temperatura (grados centígrados), luminosidad (lúmenes), ruido (decibelios), tipo de sensor, latitud, longitud, un identificador y la fecha de última modificación. Estos datos son actualizados cada 5 minutos en los sensores actualmente en funcionamiento, ya que existen algunos sensores que no se actualizan pero que podemos descartar filtrando gracias a su fecha de última modificación. Por ejemplo, en la zona al sur del Hospital Valdecilla se encuentran varios sensores los cuales se actualizaron por última vez en Julio de 2018.

El acceso a esos datos se puede hacer de muchas maneras como RDF, HTML, JSON, N3, XML, TURTLE, CSV, ATOM o JSONLD, siendo JSON el formato que se usa por defecto en la respuesta de la API y el que se ha usado en este caso para acceder a los datos. Hay dos opciones para indicar el formato de respuesta que se desea:

1. Mediante el parámetro “Accept” en las cabeceras de la petición. Por ejemplo: Accept:application/rdf+xml.
2. Indicando el formato mediante su extensión en la petición. Por ejemplo: http://datos.gob.es/apidata/catalog/dataset.xml.

Los sensores a los que vamos a acceder están distribuidos por gran parte de la ciudad de Santander, pero con varios focos diferenciados con mayor densidad de sensores, los cuales corresponden a las calles entre Puertochico y Cañadio, plaza Pombo, la zona de estaciones de autobuses y trenes, alrededores de IES Santa Clara y unos pocos situados alrededor de la Facultad de Ciencias de la UC.

2.2 Captura de requisitos

Debido a la naturaleza del proyecto, ya que es un proyecto creado a iniciativa personal, los requisitos han sido consensuados con el tutor de este proyecto principalmente mediante *brainstorming* y continuas revisiones durante el avance del proyecto.

La idea inicial fue reconduciéndose con cada reunión y revisión hasta dar forma a la aplicación final. Por ejemplo, al llegar al desarrollo de la parte de mapa, se realizó un cambio general de la interfaz gráfica, pasando de la idea de mostrar en la página principal la posición de únicamente el sensor seleccionado, a relegar el mapa a otra ventana con todos los sensores de forma que no sobrecargase tanto al usuario en su vista principal. Otro caso de cómo se han incorporado cambios sobre la marcha es el de las estadísticas. Tras añadir esa función descubrimos la necesidad de crear una especie de servicio de limpieza que eliminase estadísticas antiguas cada cierto tiempo para no sobrecargar el almacenamiento del móvil.

2.3 Objetivos

Como se ha indicado en la introducción, existen tres grandes fines u objetivos que esta app debe cumplir:

- Conocer el estado actual de distintas zonas de la ciudad con el fin de escoger una zona para la realización de una actividad concreta. Para la realización de este objetivo, se busca que el usuario explore distintos sensores disponibles en zonas que le resulten interesantes, para que, comparando los datos que obtiene, pueda decidir qué zona le resulta más favorable para su actividad. Este objetivo sería de fácil cumplimiento simplemente accediendo al mapa completo de sensores, y una vez allí, entrar en la vista detallada de los sensores que estén en las zonas que desea conocer para visualizar toda su información. Si el usuario requiere tomar una decisión inmediata, la forma que acabamos de describir sería la más correcta, pero si el usuario prefiere tomarse más tiempo en su decisión, lo que puede hacer es agregar cada sensor investigado a favoritos para una consulta más cómoda y sencilla.
- Monitorizar zonas de la ciudad para analizar su evolución en el tiempo. La finalidad que se busca en este objetivo es la de poder disponer de unos datos que reflejen el estado de una zona durante un tiempo prolongado. Por ejemplo, el usuario necesita conocer la información media de una zona durante toda una semana para cerciorarse que los datos que ve un día concreto no son algo puntual, ya que la decisión que busca es algo prolongado, como la adquisición de una vivienda. Para que el usuario pueda ver satisfecho este objetivo, los sensores disponen de estadísticas que se irán recopilando una vez agregue el sensor deseado a sus favoritos. De esta forma puede ir consultando como van variando las estadísticas de ese sensor durante el tiempo que necesite.
- Establecer una serie de alertas que notifiquen al usuario de ciertos eventos o fenómenos especiales. Este objetivo busca que el usuario pueda desentenderse de la aplicación y

sea la propia aplicación, previa configuración, quien le notifique que está pasando, ya que al usuario le interesa estar al corriente de si una zona concreta sobrepasa unos valores peligrosos, como puede ser una temperatura excesiva. Para ello, la aplicación dispone de un sistema de alertas. El cual permite al usuario establecer alarmas en cada sensor que le notifiquen si el valor que ha establecido para el aviso se ha excedido o ha bajado de lo establecido, manteniendo notificado al usuario sin necesidad de consultar continuamente la información de los sensores.

2.4 Historias de usuario

Los requisitos funcionales del proyecto se fueron especificando como historias de usuario, de las cuales las principales implementadas son las siguientes:

- Mostrar sensores en mapa.
- Mostrar información detallada de un sensor.
- Crear grupos de sensores.
- Añadir sensores a grupos.
- Agrupar/filtrar sensores en mapa.
- Editar sensores.
- Crear alarmas de sensor.
- Mostrar alarmas.
- Mostrar dirección de sensor.
- Generar estadísticas de sensor.
- Mostrar estadísticas de sensor.

Junto a estas historias de usuario existen otras como eliminar grupo o eliminar alarma que no mostraremos aquí para no sobrecargar con detalles innecesarios.

A continuación, mostramos a modo de ejemplo el contenido de dos de estas historias de usuario.

Mostrar sensores en mapa.

Yo, como usuario de la app, quiero poder visualizar los sensores obtenidos del servidor en un mapa para poderlos situar espacialmente mejor, pudiendo así poder seleccionar aquellos que me interesen más fácilmente.

Prueba de aceptación:

Prueba 00: Éxito

1. El usuario selecciona la opción “Ver mapa”.
2. El sistema muestra la ventana de mapa.
3. El sistema descarga la información de sensores.
4. El sistema procesa los sensores disponibles y los incorpora al mapa como marcadores.
5. Se verifica que el usuario puede visualizar los sensores en el mapa y su ubicación es correcta.

Prueba 01: Sin conexión a internet.

1. El usuario selecciona la opción “Ver mapa”.
2. El sistema muestra la ventana de mapa.
3. El sistema no puede descargar la información de sensores porque no hay conexión a internet.

4. Se verifica que el sistema muestra un mensaje por pantalla al usuario indicando que no hay conexión y pidiendo que lo vuelva a intentar.

Prueba 02: Sin conexión al servidor.
1. El usuario selecciona la opción “Ver mapa”.
2. El sistema muestra la ventana de mapa.
3. El sistema no puede descargar la información de sensores porque el servidor de datos no responde.
4. Se verifica que el sistema muestra un mensaje por pantalla al usuario indicando que no se ha podido conectar al servidor y pidiendo que lo vuelva a intentar más tarde.

Mostrar alarmas.

Yo, como usuario de la app, quiero poder visualizar las alarmas establecidas para todos los sensores de manera que pueda administrarlas y visualizar las activaciones que se han producido para cada alarma.

Prueba de aceptación:

Prueba 00: Éxito.
1. El usuario selecciona la opción “Ver alarmas”.
2. El sistema muestra la ventana de alarmas.
3. El sistema carga la lista de alarmas establecidas.
4. Se verifica que el sistema que muestra la lista de alarmas establecidas.

Prueba 01: No hay alarmas.
1. El usuario selecciona la opción “Ver alarmas”.
2. El sistema muestra la ventana de alarmas.
3. El sistema intenta cargar la lista de alarmas establecidas, pero está vacía.
4. Se verifica que el sistema muestra un texto al usuario indicando que no hay alarmas.

Prueba 02: Error en la petición de la lista de alarmas.
1. El usuario selecciona la opción “Ver alarmas”.
2. El sistema muestra la ventana de alarmas.
3. El sistema intenta cargar la lista de alarmas establecidas, pero no existe.
4. Se verifica que el sistema muestra un texto de error al usuario pidiendo que vuelva a intentarlo e indicando que borre los datos guardados de la aplicación si el sistema persiste.

2.5 Requisitos no funcionales

Para analizar los requisitos no funcionales que debía cumplir esta aplicación analizamos el estándar ISO 25010, el cual categoriza los principales requisitos no funcionales que suelen tener que cumplir las aplicaciones software. Tras un análisis detallado de este estándar llegamos a la conclusión de que la aplicación no requiere satisfacer de forma especial ninguno de los requisitos de la ISO 25010 más allá de lo básico.

Si nos fijamos en algunas de las características de la ISO 25.010 que más comúnmente puedan requerir especial atención vemos el porqué de no requerir satisfacer esos requisitos de forma especial.

Por ejemplo, en cuanto a rendimiento, la aplicación no tenía que satisfacer ningún requisito no funcional más allá de los requeridos a cualquier tipo de aplicación software. Lo que se quiere decir con esto es que la aplicación tiene que ser más o menos rápida y hacer un uso razonable de las características del dispositivo móvil como la RAM o el almacenamiento, pero no existiendo nada especialmente remarcable en cuanto a rendimiento que sea imprescindible que esta aplicación deba satisfacer.

De forma similar, en cuanto a seguridad, dado que la aplicación no guarda ningún tipo de dato personal ni de carácter confidencial relevante, no se veía necesario establecer ningún requisito funcional al respecto de la seguridad.

3. Arquitectura y diseño

Esta sección describe como está estructurada la aplicación a nivel de arquitectura y diseño. En primer lugar, el patrón arquitectónico usado y como se organizan sus diferentes partes. A continuación, veremos la estructura de la base de datos y su implementación. Para terminar, mostraremos el diseño a nivel gráfico de las interfaces de la aplicación y el diseño de la lógica interna de las funcionalidades más complejas.

3.1 Arquitectura

La aplicación se trata de un cliente móvil que se conecta a un servicio de datos. Por tanto, se puede ver como una capa de presentación o front-end que consume datos de un servidor. Para el desarrollo de estas capas de presentación existen diversos patrones arquitectónicos como pueden ser el patrón Model View Controller(MVP)[5], o el patrón Modelo Vista Presentador(MVP)[6]. En nuestro caso escogimos el patrón *modelo vista presentador* por facilitar este el diseño y desarrollo de pruebas. La Figura 1 muestra la arquitectura de la aplicación.

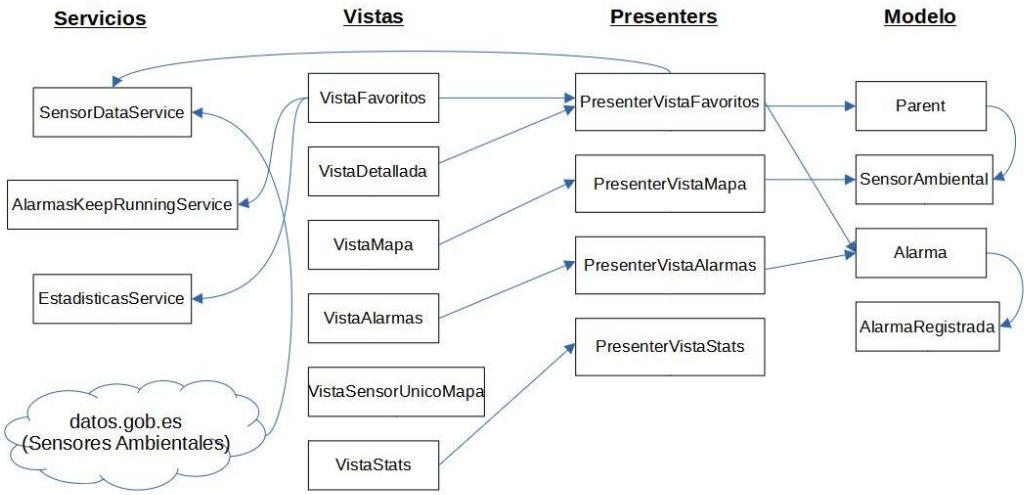

Figura 1. Arquitectura de la aplicación.

Para describir la arquitectura de la aplicación comenzaremos con las vistas, las cuales corresponden a cada interfaz que se le mostrará al usuario en la aplicación.

VistaFavoritos, es la vista principal de la aplicación donde se mostrarán los grupos de sensores creados con los sensores que se hayan asignado a cada grupo. *VistaDetallada* muestra la información completa de un sensor concreto y donde se puede agregar el sensor a un grupo. Ambas hacen uso del *PresenterVistaFavoritos*. Este *PresenterVistaFavoritos* tendría entre sus operaciones los métodos que se activan con cada evento generado desde las dos vistas que dependen de él. En sus operaciones, debido a que es el presenter principal y tiene más carga, hace uso tanto de objetos de tipo *Alarma* como de tipo *Parent*, que representa un grupo o elemento padre en la estructura jerárquica de sensores, los cuales a su vez tienen listas de *SensorAmbiental.*

VistaMapa corresponde a la vista que muestra el mapa completo de todos los sensores disponibles, diferenciándolos por su tipo. Desde esta interfaz se puede seleccionar un sensor para ver su vista detallada. Esta vista dispone de un presenter propio que sería

PresenterVistaMapa, el cual interactúa en sus operaciones con los sensores que serían objetos de tipo *SensorAmbiental*.

VistaAlarmas se encarga de mostrar una lista de las alarmas establecidas por el usuario y de las que han sido registradas en cada una de ellas, delegando cualquier operación realizada por el usuario a su *PresenterVistaAlarmas*.

VistaSensorUnicoMapa únicamente muestra la ubicación del sensor elegido en el mapa de Santander indicando su dirección, y como no dispone de opciones para el usuario no requiere de la ayuda de una clase presenter.

VistaStats mostrará, para el sensor seleccionado, una serie de estadísticas generales de los tres tipos de datos que recoge un sensor. El tipo de dato del que se quiere ver las estadísticas podrá cambiarse por el usuario gracias a las operaciones disponibles en *PresenterVistaStats*.

Por ultimo tenemos los servicios, los cuales no encajarían en la estructura del patrón Modelo Vista Presentador, pero sirven para añadir funcionalidades importantes a la aplicación.

SensorDataService es el servicio encargado de la recuperación de los datos del portal del Ayuntamiento de Santander, gestionando tanto su descarga como de la organización de todos sus datos en objetos de tipo *SensorAmbiental* con todos sus atributos.

AlarmasKeepRunningService se encarga de comprobar si las condiciones para generar una alarma, por ejemplo, por exceso de ruido, se cumplen, registrando en ese caso la fecha y el valor con el que ha saltado la alarma y notificando al usuario. Esta comprobación se realizará cada 5 minutos ya que es el tiempo de actualización de los sensores en el dataset.

EstadisticasService recoge muestras de los valores de los sensores para los cuales queremos generar información estadística. Estos valores se almacenan en una base de datos SQLite.

Tanto el servicio *AlarmasKeepRunningService* como *EstadisticasService* se inician junto con la aplicación, por lo que se ejecutan desde *VistaFavoritos* ya que es la vista inicial. El servicio *SensorDataService* en cambio se llama desde el *PresenterVistaFavoritos* debido a que puede ser llamado desde varios lugares y a que no es necesario en el arranque de la aplicación.

También debemos recordar que, aunque no forme parte de la parte desarrollada por nosotros, el servicio de datos de la web datos.gob.es, del que hemos hablado ampliamente en la sección 2.1, también forma parte de la arquitectura de la aplicación.

3.2 Diseño de las interfaces

Para el diseño de la interfaz de usuario se ha seguido un diseño simple pero intuitivo y sencillo de usar, con iconos que representen su funcionalidad de forma clara, o menús que indiquen que hacen con su propio nombre corto. A continuación, mostramos la interfaz de la aplicación y explicamos brevemente sus partes importantes.

Figura 2. VistaFavoritos.

La figura 2 corresponde a la interfaz de *VistaFavoritos*, que es la vista principal de la app y con la que nos recibe al arrancar. En la imagen podemos ver como hay 3 grupos añadidos, y uno de ellos, el grupo 2 podemos verlo desplegado para ver su contenido con 2 sensores. El primer sensor, *Sensor Ruido 1*, a su vez se ha desplegado para ver todas las opciones disponibles. La barra superior dispone de un menú el cual, en orden de izquierda a derecha, serviría para refrescar la pantalla (flecha circular), añadir un nuevo grupo (símbolo más), o desplazarse a una de las paginas opción que muestra el menú desplegable.

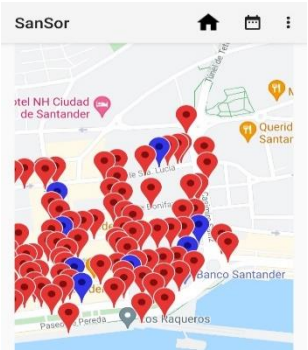

Figura 3. VistaMapa

La figura 3 muestra la interfaz de *VistaMapa* en la que podemos ver una zona de Santander donde se encuentran gran cantidad de sensores. Los colores indicando el tipo de sensor del que se trata: azul para los de tipo Ruido, y rojo para los de tipo Ambiental (temperatura y luz). Esta vista dispone de varias opciones en su menú superior, las cuales corresponden, de izquierda a derecha, al botón de retorno a la vista principal, un calendario para filtrar sensores por fecha de actualización y un pequeño menú de filtrado por tipo de sensores.

Figura 4. VistaDetallada

En la figura 4 podemos ver la vista detallada de un único sensor, mostrando su identificador, sus datos de temperatura y luminosidad y la fecha de última modificación de los datos del sensor. En esta vista es donde podremos añadir el sensor a un grupo de la página principal, usando para ello las opciones que aparecen abajo en la pantalla.

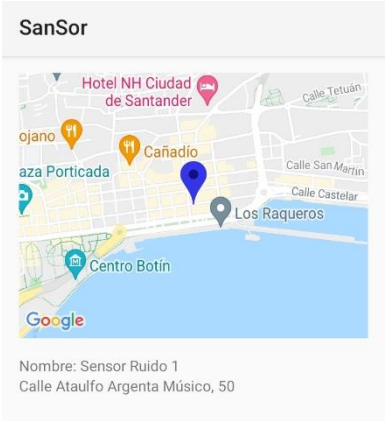

Figura 5. VistaSensorUnicoMapa

La figura 5 se corresponde con la vista de Sensor Único. En esta vista se muestra un único sensor colocado en el mapa para visualizar así mejor su posición e indicando la dirección de esa posición para mayor exactitud.

Figura 6. VistaAlarmas

En la figura 6 vemos como sería la información de una alarma establecida para un sensor concreto. En este caso se ha indicado que queremos que la app nos avise si la temperatura sobrepasa los 20 grados. Además, en este caso vemos como ya tiene registrada una incidencia relativa a un momento concreto en que ese sensor sobrepasó el valor establecido.

Figura 7. VistaStats

Por último, en la figura 7, podemos ver lo que nos muestra la vista de estadísticas. En esta vista, para un sensor concreto, se muestra media, mediana, valor máximo, valor mínimo y desviación estándar. Estas estadísticas podemos visualizarlas para diferentes magnitudes eligiendo el tipo que deseemos en el menú desplegable de los tres puntos, pero teniendo en cuenta que según el tipo de sensor existirán opciones que no muestren estadísticas ya que no recoge esos datos.

3.3 Diseño lógica interna

Los servicios implementados en la aplicación son la parte más compleja y difícil de entender de la aplicación, por lo que vamos a explicar el funcionamiento y su lógica interna para conocer como realizan sus operaciones durante su ejecución.

3.3.1 Servicio de alertas AlarmasKeepRunningService

Como ya hemos explicado anteriormente, este servicio es el que se encarga de comprobar si las alarmas fijadas se activan, notificando al usuario si se da el caso. Para explicar su funcionamiento nos ayudaremos de un pequeño diagrama de estados que podemos ver en la siguiente figura 8.

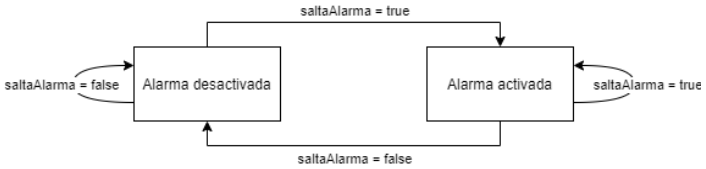

Figura 8. Diagrama estados de alarmas

El servicio comprueba cada 5 minutos si un determinado sensor supera el valor indicado. Por ejemplo, una determinada temperatura. Si el sensor excede el valor indicado, el servicio establece verdadero como valor de una variable denominada *saltaAlarma*. La comprobación se realiza cada 5 minutos que es el tiempo de actualización de los sensores, por lo que no tiene sentido realizar estas comprobaciones con una frecuencia menor.

El servicio se inicia para una alarma concreta en el estado desactivado y se comporta de acuerdo con las siguientes reglas.

- Si la alarma esta desactivada y *saltaAlarma* es falso, el estado no cambia, ya que no ha pasado nada y no queremos notificar al usuario.
- Si la alarma esta desactivada y *saltaAlarma* cambia a verdadero, la alarma pasa a estar activada, y se envía al usuario una notificación emergente indicando qué alarma ha sido activada y con qué datos. También queda registrada en ese momento la alarma activada en la lista de alarmas registradas.

- Si la alarma está activada y *saltaAlarma* es verdadero, no cambia nada. Esto se debe a que no queremos que una vez activada una alarma se vuelva a notificar al usuario que un sensor ha superado un determinado valor. Es decir, si, por ejemplo, un sensor excede la temperatura de 25 grados, no queremos estar mandando una notificación al usuario cada 5 minutos hasta que la temperatura de ese sensor baje.
- Si la alarma esta activada y *saltaAlarma* es falso, volvemos al estado inicial. Este cambio se producirá cuando el valor establecido para el aviso vuelve a Estar por debajo del valor establecido para las alarmas. Es decir, si la alarma se activó al sobrepasar 20 grados de temperatura, no volverá a desactivarse hasta que baje de esos 20 grados establecidos en la alarma.

3.3.2 Servicio de estadísticas EstadisticasService

El funcionamiento de este servicio de estadísticas dispone de varias opciones que requieren de una pequeña explicación. Esas opciones son las siguientes:

- *Frecuencia de muestreo*. Es el intervalo de tiempo que pasará entre cada recogida de muestras de datos. Por defecto será de 5 minutos, pero podemos establecerlo en 1 hora o 1 día.
- *Tiempo de cálculo*. Se usa para conocer qué intervalo de medidas se van a usar para calcular las estadísticas. Es decir, si el tiempo de cálculo es 1 hora, que sería el valor por defecto, las estadísticas serán calculadas solo con las medidas de la última hora. Este campo podemos modificarlo para tener estadísticas de todo 1 día o de 1 semana.
- *Tiempo de vida de estadísticas*. Este tiempo hace referencia a cuánto tiempo se mantendrán en la base de datos las medidas recogidas para las estadísticas. La opción por defecto sería de 1 día, pero podremos establecerlo en 1 semana o 1 mes de tiempo de vida.

Vistas las opciones, su funcionamiento es sencillo. El servicio se arranca con los valores por defecto de las opciones y toma la primera medida. El servicio continúa recogiendo medidas cada intervalo de tiempo establecido en la frecuencia de muestreo. Estas medidas son añadidas a la base de datos, lugar de donde se vuelven a recoger para hacer los cálculos de todas las estadísticas de acuerdo con el valor establecido en la opción tiempo de cálculo. Por último, se realiza la operación de limpieza que comprobará qué medidas deben ser eliminadas de la base de datos en el caso de que sobrepasen el tiempo de vida establecido.

3.4 Base de datos: accesos y esquema

La aplicación hace uso de una pequeña base de datos en SQLite para guardar las estadísticas. Esa base de datos consta de una única tabla, la cual se muestra en la figura 9.

Medidas
- id(PK) - id_sensor - fecha - fecha_cortada - temp - ruido - luz

Figura 9. Tabla medidas

Como hemos comentado, para la implementación de esta pequeña base de datos se ha usado SQLite. Esta implementación requiere de una estructura y unas clases de apoyo que podemos ver en la figura 10 y que explicamos a continuación.

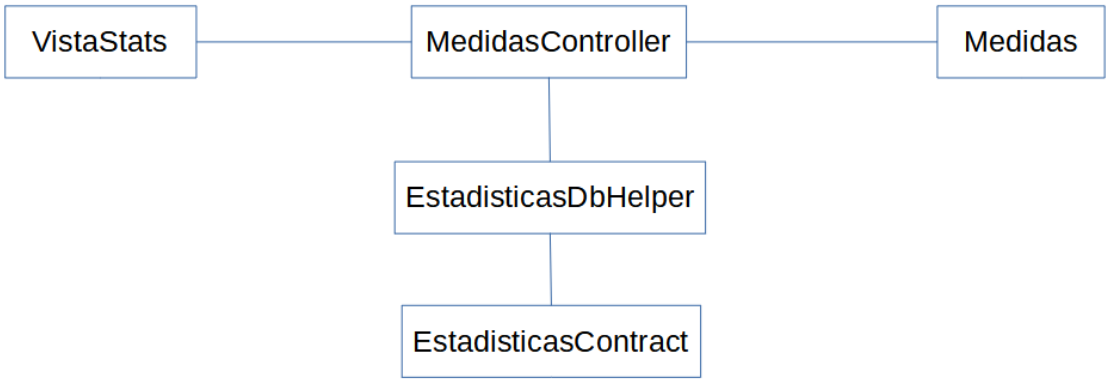

Figura 10. Estructura BBDD SQLite

- *VistaStats*: la clase vista que se encarga de la gestión de la interfaz y por lo tanto de mostrar los datos de las estadísticas y de cualquier operación ajena a la base de datos como cálculos realizados a partir de los datos obtenidos de la tabla medidas.
- *MedidasController*: esta clase contiene los métodos CRUD como un nuevo *insert* de una medida, *delete* de una medida y distintos *selects* necesarios. Es la clase encargada de interactuar entre la parte SQL y la parte Java.
- *EstadisticasDbHelper*: esta clase es la que permite crear el esquema de la base de datos en el terminal del cliente cuando se instala la app y eliminar dicha app cuando la desinstala.
- *EstadisticasContract*: contiene la estructura que debe seguir cada entrada que se realice en la tabla de la base de datos indicando sus columnas.
- *Medidas*: esta clase será la representación en Java de la tabla SQL medidas, por lo que un objeto de esta clase estará haciendo referencia a una fila de la tabla medidas.

El resto de información que requiere ser guardada, al ser pequeñas listas de objetos, se almacenan mediante una clase de ayuda llamada *TinyDB* la cual hace uso de *SharedPreferences* para su funcionamiento. Estas listas son:

- Parents. El conjunto de grupos con sus sensores hijos.
- ListaAlarmas. La lista de alarmas establecidas por el usuario.
- NombreGrupos. Una lista con únicamente los String de los nombres de los grupos creados.

4. Implementación

En esta sección mostraremos algunos ejemplos de cómo se ha llevado a cabo la codificación de este proyecto.

4.1 Añadir sensor a favoritos

Para añadir un sensor a favoritos la interacción necesaria comienza en la VistaDetallada donde se encuentra el botón “Añadir a favoritos” cuya codificación es la siguiente.

```
botonAddFav.setOnClickListener(new View.OnClickListener() {
	@Override
	public void onClick(View v) {
		mPresenter.onClickAddFavorito(sensor, grupoSeleccionado);
	}
});
```

Figura 11. Código añadir a favoritos 1

Al hacer clic en el botón, se activa el listener, el cual tiene un método onClick que realiza una llamada al método onClickAddFavorito de la clase presenter. Es en ese método de la clase presenter donde se realiza toda la lógica para añadir el sensor a la lista de favoritos a partir de los argumentos que ha recibido indicando que sensor es y a que grupo se añadirá.

A continuación, vemos el método onClickAddFavorito de la clase presenter al que acabamos de hacer referencia.

```
public void onClickAddFavorito(final SensorAmbiental sensor, final String grupo){
	TinyDB tinydb = new TinyDB(mView.getAppContext());
	parents = tinydb.getListParent("parents");
	AlertDialog.Builder builder = new AlertDialog.Builder(mView.getActivityContext());
	builder.setTitle("Introduce el nombre del nuevo sensor:");
	final EditText inputSensor = new EditText(mView.getActivityContext());
	inputSensor.setInputType(InputType.TYPE_CLASS_TEXT);
	builder.setView(inputSensor);
	builder.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
		@Override
		public void onClick(DialogInterface dialog, int which) {
			String mText = inputSensor.getText().toString();
			sensor.setTitulo(mText);
			for (Parent p : parents){
				if(p.getNombre().equals(grupo)){
					p.addChild(sensor);
				}
				TinyDB tinydb = new TinyDB(mView.getAppContext());
				tinydb.putListParent("parents", parents);
				Intent volverALista = new Intent(mView.getActivityContext(),
VistaFavoritos.class);
				mView.getActivityContext().startActivity(volverALista);
			}
		}
	});
	builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
		@Override
		public void onClick(DialogInterface dialog, int which) {
			dialog.cancel();
		}
	});
	builder.show();
}
```

Figura 12. Código añadir a favoritos 2

Ya en el presenter, el método onClickAddFavorito primero coge la lista actualizada de Parents con el método de TinyDB (lineas 2 y 3). A partir de ahí, se crea un AlertDialog (línea 4) que le muestra al usuario un pequeño formulario para indicar el nombre que le quiere poner al nuevo sensor que se va a añadir. Ese nombre se le aplica al objeto sensor, (líneas 12 y 13) se añade como sensor hijo al parent correspondiente (línea 16) y se actualiza la lista parents en TinyDB (líneas 18 y 19). Una vez hecho todo se indica que vuelva a la vista principal mediante un intent (líneas 20 y 21). Para terminar, se establece la opción de cancelar el AlertDialog para no añadir el sensor (líneas 25 a 30) y se muestra el AlertDialog (línea 31).

4.2 Acceso a la base de datos.

Durante la creación y consulta de estadísticas se realizan continuos accesos a la base de datos. A continuación, vamos a mostrar la codificación de un acceso que correspondería a una instrucción SELECT para obtener las temperaturas guardadas de un sensor concreto.

```
1  public ArrayList<String> obtenerTemperaturasSensor(String sensorId) {
2        ArrayList<String> temperaturas = new ArrayList<>();
3        SQLiteDatabase baseDeDatos = ayudanteBD.getReadableDatabase();
4        String[] columnasAConsultar = {campotemp};
5        Cursor cursor = baseDeDatos.query(
6                        nombre_Tabla,
7                        columnasAConsultar,
8                        campoidsensor + " = ?",
9                        new String[]{sensorId},
10                       null,
11                       null,
12                       null
13       );
14       if (cursor == null) {
15               return temperaturas;
16       }
17       if (!cursor.moveToFirst()) return temperaturas;
18       do {
19               String temp = cursor.getString(0);
20               temperaturas.add(temp);
21       } while (cursor.moveToNext());
22       cursor.close();
23       return temperaturas;
24 }
```

Figura 13. Código acceso a la base de datos

Primero, el método establece las variables necesarias, como son el array de resultado de temperaturas (línea 2), la base de datos donde se va a realizar la consulta (línea 3) que se obtiene desde ayudanteBD que corresponde a la clase EstadisticasDbHelper de la cual hablamos en la sección 3.4, y un string indicando que columnas se van a consultar (línea 4). Establecidas las variables se crea un cursor con la query que se va a consultar a la base de datos (líneas 5 a 13). Esta query sería la traducción directa de la instrucción SELECT en SQL en la cual los tres últimos campos corresponden a groupBy, having y orderBy que no son necesarios y por ello se les asigna null. Si el cursor es null (línea 14) o está vacío (línea 17) retornara la lista de temperaturas vacía. En caso contrario, recorrerá cada elemento resultante de la consulta y los irá añadiendo a la lista de temperaturas (líneas 18 a 21). Una vez recorridos todos los elementos se procede a cerrar el cursor y a devolver la lista de temperaturas obtenidas en la consulta.

4.3 Servicio estadísticas

La codificación de los servicios realizados es muy amplia y requiere de mucha explicación detallada, por lo que vamos a mostrar solamente una parte del servicio de estadísticas, el cual corresponde a como se inicia el servicio.

```
1 public void startTimer() {
2       final int[] intervaloMuestreo = new int[1];
3       timer = new Timer();
4       TimerTask timerTask = new TimerTask() {
5               public void run() {
6                       TinyDB tinydb = new TinyDB(getBaseContext());
7                       parents = tinydb.getListParent("parents");
8                       for (Parent p : parents) {
9                               for (SensorAmbiental sensor : p.getChildren()) {
10                                      intervaloMuestreo[0] = sensor.getIntervaloStatsMuestreo();
11                                      updateSensor(sensor);
12                                      recogeStats(sensor);
13                              }
14                      }
15              }
16      };
17      if(intervaloMuestreo[0] ==0){
18              timer.schedule(timerTask, 0, 300000); //5 min
19      }
20      if(intervaloMuestreo[0] ==1){
21              timer.schedule(timerTask, 0, 3600000); //1 hora
22      }
23      if(intervaloMuestreo[0] ==2){
24              timer.schedule(timerTask, 0, 86400000); //1 dia
25      }
26      Intent inResult = new Intent(ACTION);
27      inResult.putExtra("resultCodeStats", Activity.RESULT_OK);
28      inResult.putExtra("parentsResult", parents);
29      LocalBroadcastManager.getInstance(this).sendBroadcast(inResult);
30}
```

Figura 14. Código servicio estadísticas

Esta parte del servicio de estadísticas corresponde al método startTimer, el cual se encarga de arrancar una tarea periódica, ya que queremos que nuestro servicio se ejecute permanentemente cada cierto intervalo de tiempo. Para ello creamos una timerTask (línea 4) la cual va a realizar varias operaciones para cada uno de los sensores guardados por el usuario. Estas operaciones son las de recoger que intervalo de muestreo tiene establecido el sensor (línea 10), actualizar el sensor para disponer de datos actuales en cada ejecución (línea 11) y recoger las estadísticas del sensor (línea 12), que será la parte que añada una nueva medida en la base de datos con los datos obtenidos del sensor en ese momento. Como hemos comentado, queremos que la ejecución del servicio sea periódica, por lo que hay que programar ese intervalo de ejecución, el cual dependerá del intervalo de muestreo establecido (líneas 17 a 25). Por último, indicamos en un intent que será enviado mediante un Broadcast (línea 29) primero el resultado ok de la operación (línea 27) y la lista de sensores usada ya que se podría haber realizado algún cambio (línea 28).

5. Pruebas, control de calidad y despliegue

En esta sección hablaremos de las pruebas realizadas, del control de calidad que se ha efectuado junto con sus resultados y del despliegue que se ha llevado a cabo.

5.1 Pruebas

Con respecto a las pruebas, no se han implementado apenas pruebas automatizadas para verificar el correcto funcionamiento de la aplicación, ya que el código mantenía diversas dependencias que resultaban difíciles de aislar. Por ejemplo, para probar correctamente los métodos de la clase *PresenterVistaFavoritos* necesitaba aislar la referencia a la clase *TinyDB,* para lo que, por desgracia, carecía de los conocimientos técnicos al ser una librería ajena al desarrollo propio de la aplicación. Por tanto, se crearon pruebas unitarias exclusivamente para la clase *Parent*. La figura 15 muestra el código de varias de estas pruebas unitarias.

```
1       public void testAddChild() {
2               assertTrue(mParent.addChild(createSensor("Sensor1")));
3       }
4
5       public void testRemoveChild() {
6               SensorAmbiental sensor = createSensor("testSensor");
7               mParent.addChild(sensor);
8
9               assertTrue(mParent.removeChild(sensor));
10              SensorAmbiental fakeSensor = createSensor("fakeSensor");
11              assertFalse(mParent.removeChild(fakeSensor));
12      }
13
14      public SensorAmbiental createSensor(String text){
15              SensorAmbiental sensor = new SensorAmbiental();
16              sensor.setIdentificador("00008");
17              sensor.setTitulo(text);
18              return sensor;
19      }
```

Figura 15. Pruebas unitarias Parent

El resto de la aplicación se probó de manera manual, ejecutando a mano las diversas funcionalidades implementadas y comprobando que su resultado era correcto. Especialmente complejo de verificar fueron los servicios, ya que había que verificar que éstos realizasen una serie de actividades periódicas y se activasen ante unas determinadas condiciones. Por ejemplo, para probar el funcionamiento de las alarmas, se fijaron varios avisos en intervalos de temperatura fácilmente alcanzables durante el día y se comprobó durante varios días que se producían notificaciones y que las alarmas producidas quedaban registradas.

Finalmente, mencionar que las pruebas de aceptación se realizaron semanalmente utilizando a mi tutor del Trabajo Fin de Grado como *Product Owner* o representante de los usuarios finales. Cada semana, se mostraba el trabajo realizado en la semana anterior al tutor, quien realizaba ciertas sugerencias de cambio para mejorar el funcionamiento de la aplicación. Por ejemplo, al añadir nuevas opciones a los sensores, la interfaz principal quedo sobrecargada, por lo tanto, se comentó como colocar todas las opciones de forma cómoda para el usuario, también se introdujo la posibilidad de que los sensores se mostrasen en un formato tipo *CardView* y que fuese desplegable para que las opciones no ocupen tanto espacio mientras el usuario no las necesite.

5.2 Control de calidad

Como comentamos en la sección 1.3 herramientas de desarrollo, el control de calidad realizado a la aplicación es realizado mediante sonarqube.

En un primer análisis los resultados obtenidos contenían bastantes problemas como podemos ver en la siguiente figura.

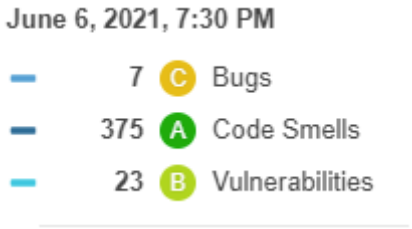

Figura 16. Sonar inicial

Debido a la cantidad de bugs y vulnerabilidades existentes, se realiza un plan de acción para corregirlos, dando especial prioridad a los bugs existentes, en segundo lugar, a las vulnerabilidades y por último se eliminarán tantos code smells como fuera posible.

Durante el escaneo inicial también se descubrió un fallo de seguridad marcado como leve, el cual avisaba del riesgo de la existencia de un acceso al almacenamiento externo a la aplicación. Este fallo de seguridad se revisó detenidamente y se marcó como seguro, ya que como explicamos en los requisitos no funcionales en la sección 2.5, la aplicación no dispone de datos personales ni de tipo delicado que puedan ser susceptibles a obtenerse por parte de terceros.

El resultado final después de la corrección de bugs, vulnerabilidades y eliminación de todos los code smells posibles es el que podemos ver en la siguiente figura.

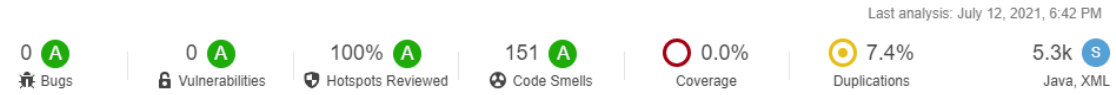

Figura 17. Sonar final

5.3 Despliegue

El despliegue de la aplicación se ha realizado a través de *Google Play*. Para ello se ha generado un *Android App Bundle* firmado desde Android Studio y se ha publicado en dicho repositorio. La configuración establecida en Google Play se ha configurado para que la aplicación sea únicamente para España, ya que solo tiene sentido su uso en Santander, en idioma español, con una clasificación de contenido PEGI 3. También se ha establecido que es una aplicación sin anuncios y se ha colocado en la categoría de aplicación de herramientas, con las etiquetas *Estilo de vida, Herramientas, Tiempo.*

6. Conclusiones y trabajos futuros

Al comenzar el proyecto, partíamos de una idea interesante, pero que podría parecer simple y sencilla. Poco a poco, esa idea fue cogiendo forma y aumentando en características y complejidad. Ese aumento de características no ha impedido ni modificado el objetivo principal de la aplicación que es el de mantener informado al usuario de los datos ambientales de su entorno en la ciudad de Santander, si no que ha aumentado las maneras en las que el usuario puede ver cumplido ese objetivo final.

Entre esas características secundarias, las más interesantes han sido los servicios de gestión de alarmas y de estadísticas, los cuales han sido probablemente la parte más compleja de la aplicación, debido sobre todo a su necesidad de uso continuo sin interrupción, pero han resultado satisfactorios y muy útiles.

La realización de este proyecto durante la pandemia y compaginado con un trabajo a tiempo completo no ha resultado fácil en ocasiones y puede haber ocasionado algún retraso en el tiempo total del proyecto, pero no ha impedido que la aplicación y el proyecto hayan sido completados.

Esta aplicación tiene bastante margen de mejora de cara al futuro, tanto en su diseño para estar a la altura de aplicaciones mucho más usadas, como en la mejora de sus características ya implementadas. Por ejemplo, sería interesante incorporar algún tipo de librería gráfica avanzada que permita generar gráficas avanzadas que muestren la evolución de un sensor a través del tiempo y exportar estas gráficas como imágenes que puedan ser fácilmente compartidas entre diversas aplicaciones. También es interesante pensar, después haber usado en este proyecto datos públicos, en nuevos proyectos que aprovechen la existencia de tantos datos abiertos públicos.

Referencias

[1] Jeff Sutherland, Rini van Solingen, Eelco Rustenberg. The Power of Scrum. End of Line Clearance Book. 2011.
[2] Mike Owens. The Definitive Guide to SQLite. Apress. 2006.
[3] Jon Loeliger, Matthew McCullough. Version Control with Git: Powerful tools and techniques for collaborative software development. O'Reilly Media. 2012.
[4] G. Ann Campbell, Patroklos P. Papapetrou. SonarQube in Action. Manning Publications. 2013.
[5] Peter Späth. Beginning Java MVC 1.0: Model View Controller Development to Build Web, Cloud, and Microservices Applications. Apress. 2020.
[6] Martin Fowler. https://acortar.link/Ltofs. 2006.