Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación móvil Android para el acceso a las previsiones meteorológicas proporcionadas por la API de AEMET

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Alejandro Marco Palomares

Tutor: David De Andrés Martínez

[2018/2019]

Resumen

Hoy en día, la mayoría de las personas disponen de un smartphone y aproximadamente el 80% de estos dispositivos tienen el sistema operativo Android. Por esta razón, en este trabajo de fin de grado se desarrollará una aplicación meteorológica para dispositivos móviles, con el fin de llegar a un mayor número de usuarios.

Con esta aplicación, se pretende agilizar el proceso de consulta del tiempo atmosférico mediante una interfaz intuitiva y fácil de usar. Permitirá a los usuarios obtener información acerca del clima de un municipio o ciudad española en el día o la semana actual.

Para su implementación, se va a utilizar el Entorno de Desarrollo Integrado (IDE) Android Studio y las previsiones meteorológicas se obtendrán a través de la API que nos proporciona AEMET (Agencia Estatal de Meteorología). El movimiento Open Data ofrece gran cantidad de información, pero acceder a ella es difícil, aunque sea pública. Por ello, se plantea facilitar este proceso a través de una aplicación móvil.

Palabras clave: Android, aplicación móvil, meteorológica, Open Data, AEMET

Resum

Hui en dia, la majoria de les persones disposen d'un smartphone i aproximadament el 80% d'estos dispositius tenen el sistema operatiu Android. Per aquesta raó, en este treball de fi de grau es desenvoluparà una aplicació meteorològica per a dispositius mòbils, a fi d'arribar a un nombre més gran d'usuaris.

Amb esta aplicació, es pretén agilitzar el procés de consulta del temps atmosfèric per mitjà d'una interfície intuïtiva i fàcil d'usar. Permetrà als usuaris obtindre informació sobre el clima d'un municipi o ciutat espanyola en el dia o la setmana actual.

Per a la seua implementació, es va a utilitzar l'Entorn de Desenrotllament Integrat (IDE) Android Studio i les previsions meteorològiques s'obtindran a través de l'API que ens proporciona AEMET (Agencia Estatal de Meteorología). El moviment Open Data oferix gran quantitat d'informació, però accedir a ella és difícil, encara que siga pública. Per això, es planteja facilitar este procés a través d'una aplicació mòbil.

Paraules clau: Android, aplicació mòbil, meteorològica, Open Data, AEMET

Abstract

Nowadays, most people have a smartphone and aproximately 80% of these devices have the Android operating system. For this reason, in this end-of-degree project I will develop a meteorological application for mobile devices, in order to reach a greater number of users.

With this application, we intend to streamline the weather consultation process through an intuitive and easy-to-use interface. Allow users to obtain information about the climate of a Spanish city or municipality on the current day or week.

For its implementation, we are going to use the Integrated Development Environment (IDE) Android Studio and weather forecasts will be obtained through the API provided by AEMET (Agencia Estatal de Meteorología). The Open Data movement offers a lot of information, but accessing it is difficult, even if it is public. Therefore, it is proposed to facilitate this process through a mobile application.

Keywords: Android, mobile app, meteorogical, Open Data, AEMET

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de fragmentos de código

1. Introducción

1.1. Motivación

Hoy en día, la proliferación de dispositivos digitales como smartphones, tabletas y ordenadores han hecho que el mundo esté cada vez más conectado entre sí, y la información sea más accesible para las personas.

Existe gran cantidad de información abierta y disponible en la web, algo que persigue el movimiento conocido como "Open Data". Este término ha cobrado gran importancia y se ha extendido rápidamente a escala internacional [1]. El "Open Data" o "Datos Abiertos" es una filosofía en la que gran cantidad de organismos e instituciones públicas o privadas exponen sus datos en distintos formatos (JSON, XML, XLS…). Dichos datos no exigen ningún tipo de permiso para su uso.

Se trata de una serie de datos que se han abierto a los usuarios para permitir el tratamiento de dicha información y abrir un abanico de posibilidades a las personas o entidades que sepan beneficiarse.

Sin embargo, aunque sean abiertos, no todos pueden acceder a ella ya que a veces se requieren ciertos conocimientos en programación, como es el caso de AEMET. La cual proporciona, a través de una API[1], información meteorológica en formato JSON[2]. Por este motivo, se ha decidido crear una aplicación que facilite el acceso a dicha información y las personas puedan disfrutar de ella.

Actualmente, existe una gran diversidad de aplicaciones meteorológicas de pago, o que ofrecen un servicio mínimo y cobran una pequeña cantidad de dinero para poder disfrutar de todas las funcionalidades. Es necesario que existan aplicaciones que se puedan utilizar de manera gratuita.

Por otro lado, el móvil ha ganado la batalla a los ordenadores. Un estudio de la GSMA, asociación que organiza el Mobile World Congress (MWC) celebrado en Barcelona, afirma qué en 2017, el número de usuarios de telefonía móvil alcanzó los 5.000 millones [2]. Hoy en día, es difícil mirar a tu alrededor y no encontrar a una persona que no disponga de un smartphone. Se ha convertido en un aparato indispensable en el día a día.

[1] API: interfaz que provee acceso a un conjunto de características de una aplicación software.
[2] JSON: formato ligero de texto utilizado comúnmente para transmitir datos.

El hecho de que la popularidad de los dispositivos móviles sea mayor que la de los PC ha influido notablemente a la hora de la elección del proyecto, de escoger las herramientas para desarrollar la aplicación y el tipo de plataforma, en este caso, Android.

1.2. Objetivos

El objetivo principal de este trabajo es el desarrollo de una aplicación móvil Android que permita la consulta de las previsiones meteorológicas proporcionadas por la API de AEMET [3]**.** Del mismo modo, también se pretende resaltar la importancia de los datos abiertos y los beneficios que nos brindan.

Para cumplir dicha meta se va a dar especial importancia al diseño, haciéndolo atractivo y novedoso. Respecto al proceso de creación, se ha decidido seguir estilos y patrones de diseño de Google Material Design para conseguir un aspecto más profesional. También la aplicación debe ser fácil de utilizar para cualquier tipo de persona.

Ha de ser a la vez consistente, para evitar que el usuario se sienta molesto durante su manipulación. La información mostrada debe ser coherente con la proporcionada por la API, de manera que ningún dato sea malinterpretado y todo lo mostrado sea lo más exacto posible.

El usuario podrá obtener información acerca del clima en distintas regiones de España, ya sean pequeños municipios o grandes ciudades. Las predicciones podrán visualizarse en formato horario o diario. Además, la aplicación tendrá otras funcionalidades como la consulta de favoritos, mapas proporcionados por AEMET y una sección de ayuda.

Se ofrecerá la posibilidad de traducir la aplicación al inglés para que el lenguaje no sea un inconveniente para aquellos usuarios que no dominen el español.

Otro objetivo secundario es la gestión de las notificaciones, una opción que debe tener el usuario para estar al tanto del clima actual. Consiste en una notificación diaria, a modo de mensaje que informa de aspectos como la temperatura o el estado atmosférico.

Finalmente, ante posibles modificaciones de la aplicación se debe seguir un conjunto de buenas prácticas, como la utilización de patrones arquitectónicos y una adecuada documentación del código implementado, creando una estructura de proyecto sólida y que facilite futuras inserciones.

1.3. Estructura de la memoria

En este apartado vamos a explicar brevemente el contenido de las partes en las que se divide este trabajo.

En primer lugar, veremos los orígenes de Android como sistema operativo y nos centraremos en su arquitectura. En el apartado 3, trataremos el estado del arte de las aplicaciones meteorológicas del mercado y realizaremos un estudio de las características de nuestra aplicación frente a ellas. Seguidamente, en el apartado 4 nos adentraremos en la definición de los requisitos del sistema mediante técnicas estudiadas en asignaturas como Ingeniería del Software o Análisis y Especificación de requisitos. En el apartado 6, hablaremos del diseño de la aplicación. A continuación, hablaremos de las tecnologías que hemos utilizado durante el desarrollo de este proyecto. Una vez dicho esto, pasaremos al aparatado 8, donde veremos en detalle los aspectos más relevantes de su implementación. En el siguiente punto, apartado 9, comentaremos los diferentes tipos de pruebas realizadas a la aplicación. En el punto 10, pasaremos a la conclusión y haremos una reflexión de si realmente hemos cumplido con los objetivos descritos en el apartado 1.2. Además, estableceremos una relación entre el proyecto y los estudios cursados en el grado de Ingeniería Informática. Finalizaremos con una sección donde se habla de la continuación de este proyecto en un futuro. Las últimas páginas contienen las referencias y una lista con los acrónimos empleados en este trabajo.

2. Android

2.1. ¿Qué es Android?

Figura 1. Logotipo de Android: Andy

Actualmente, es el sistema operativo de código abierto más popular en dispositivos móviles. Está presente en miles de millones de dispositivos, que abarcan desde teléfonos y tabletas hasta coches y relojes.

Es un sistema operativo basado en LINUX de unos 12 años de antigüedad que evoluciona constantemente sus versiones, llegando hoy en día a la versión Android 9.0 Pie. La figura 1 representa al logotipo de Android, también conocido como Andy.

A continuación, vamos a tratar los orígenes e historia de Android [4].

2.2. Historia

La primera versión beta de Android nace el 5 de noviembre de 2007 bajo la Open Handset Alliance (OHA), una alianza comercial liderada por Google y formada por otras 34 empresas desarrolladoras y fabricantes de teléfonos móviles.

El 23 de septiembre de 2008 se crea su versión estable, Android 1.0. El teléfono móvil donde pudo verse funcionando por primera vez fue el HTC Dream.

Sin embargo, hay que remontarse años atrás en el tiempo para llegar hasta sus inicios. En 2003, se fundó Android Inc. de la mano de Rich Miner, Andy Rubin, Nick Sears y Chris White con la idea de desarrollar software para dispositivos móviles en torno a las preferencias del usuario y competir contra Windows Mobile y Simbian, los sistemas operativos más populares de ese momento.

En 2005, Google compró Android Inc. y comenzó el desarrollo de una nueva plataforma móvil, basada en un sistema similar al de BlackBerry con teclado QWERTY.

En sus inicios fue prácticamente desconocido, pero ha ido evolucionando hasta convertirse en el SO para dispositivos móviles más común en todo el mundo.

2.3. Arquitectura

Android está basado en una arquitectura por capas, construido sobre un kernel de Linux [5].

En la figura 2 podemos observar las diferentes capas que la conforman y que seguidamente vamos a comentar.

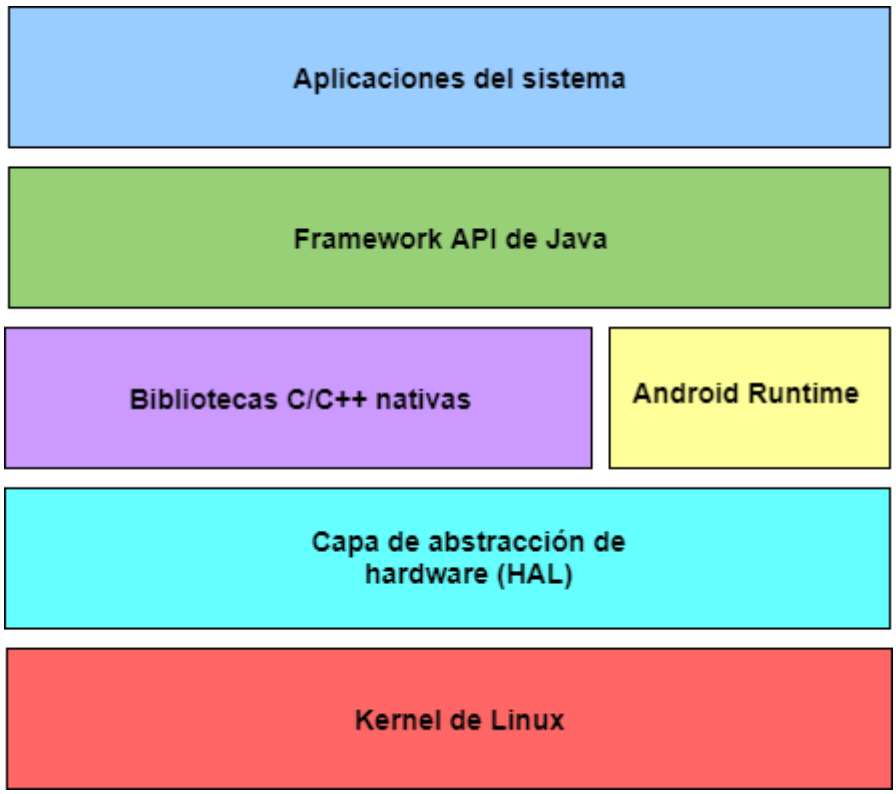

Figura 2. Arquitectura de Android

2.3.1. Kernel de Linux

El kernel Linux es la base de Android. Se encarga de manejar los recursos hardware como la creación de subprocesos o la administración de memoria de bajo nivel.

2.3.2. Capa de abstracción de hardware (HAL)

Esta capa proporciona un conjunto de interfaces que muestran las capacidades hardware del dispositivo a la capa de framework de la API Java.

Se trata de varios módulos de biblioteca, los cuales implementan una interfaz para un componente específico como la cámara, ubicación o conexión bluetooth. De modo que cuando se solicita acceder al hardware del dispositivo móvil, el sistema carga dicho módulo para el componente hardware.

2.3.3. Android Runtime (ART)

Es el entorno de ejecución de aplicaciones usado en el sistema operativo Android.

ART reemplaza a la antigua Dalvik, una máquina virtual que permite ejecutar aplicaciones Java, sacrificando la portabilidad característica de Java para construir aplicaciones con mejores prestaciones en cuanto al rendimiento y al ahorro de batería.

En Android, el código Java es compilado a bytecode gracias a la máquina virtual de Java y posteriormente, es traducido a código entendible para ART o Dalvik.

En la versión Android 4.4 es opcional el uso de este entorno de ejecución, pero a partir de su versión 5.0 Lollipop cada aplicación lanza sus procesos mediante instancias del tiempo de ejecución de Android (ART).

Además, introduce una serie de avances como la compilación anticipada (AOT) y la mejora en el desarrollo y la depuración de las aplicaciones. Algo que beneficia a los desarrolladores, mejorando el rendimiento del programa.

2.3.4. Bibliotecas C/C++ nativas

Se trata de un conjunto de bibliotecas escritas en lenguaje C y C++. Existen componentes de Android como el HAL y el ART que necesitan de estas bibliotecas nativas para funcionar.

Dos ejemplos de este tipo de bibliotecas son OpenGL para el renderizado de gráficos 3D y SQLite en la gestión de bases de datos.

2.3.5. Framework de la Java API

Esta es la capa donde se sitúan las bibliotecas necesarias para construir aplicaciones. En ella están los paquetes android.*, los cuales proporcionan una potente herramienta de desarrollo. Representa el acceso al API de Android.

Algunos de los servicios que proporciona son:

- Un sistema de vista enriquecido que permite la compilación o depuración de aplicaciones, entre otras muchas funciones.
- Administradores de recursos, notificaciones y actividades que proveen acceso a objetos de diseño como gráficos, botones o layouts, la gestión de alertas y el ciclo de vida de las aplicaciones.
- Proveedores de contenido que permiten el acceso a otras aplicaciones como Contactos o Calculadora.

2.3.6. Aplicaciones

La capa de aplicaciones es accesible a los usuarios y la interacción con ella resulta muy simple e intuitiva. La conforman todo el abanico de aplicaciones para la navegación de internet, contactos, sistemas de mensajería y juegos, entre otros elementos.

3. El mercado de las aplicaciones meteorológicas

3.1. Comparativa con otras aplicaciones

Hoy en día, existen muchas aplicaciones meteorológicas con las que estar al tanto del clima actual. Podemos encontrarnos con aplicaciones web, de escritorio o para dispositivos móviles. En este caso nos hemos centrado en las últimas, situándonos en el contexto de la nuestra.

3.1.1. El tiempo de AEMET

Figura 3. Logo de AEMET

La aplicación Android oficial de AEMET [3] proporciona información meteorológica para miles de localidades de España. La figura 3 representa su logo.

Los datos a partir de los cuales obtiene las predicciones climáticas son accesibles a través de su API. Están disponibles para todo el mundo sin restricciones de derechos de autor.

En cuanto al diseño se puede decir que resulta poco novedoso y no muy intuitivo para aquellas personas que no estén familiarizadas con aplicaciones meteorológicas. La interfaz deja mucho que desear con respecto al resto de aplicaciones del mercado, que presentan la información de manera más llamativa.

Tampoco dispone de un buen nivel de traducción que permita a usuarios extranjeros, que residan en España, adaptarlo a su idioma para su mejor comprensión.

Como puede apreciarse en la figura 4, tras seleccionar el inglés como idioma en el dispositivo móvil, únicamente la fecha actual se traduce, el resto permanece igual.

Por ello, otro objetivo de este trabajo de fin de grado es mejorar estos aspectos de la aplicación.

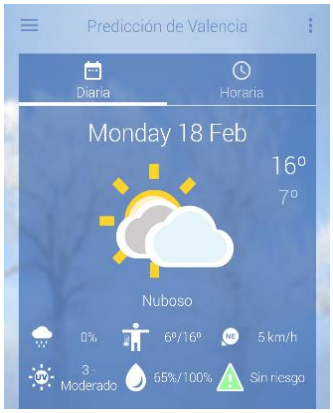

Figura 4. Ejemplo de traducción pobre en AEMET.

3.1.2. Eltiempo.es

Eltiempo.es [6] una aplicación que mejora varios aspectos con respecto a AEMET, como el diseño y la manera de mostrar la información, ya que presenta una vista más profesional y atractiva para los usuarios. La figura 5 representa su logotipo.

Figura 5. Logo de Eltiempo.es

Permite visualizar tanto predicciones climáticas, como mapas, avisos oficiales, noticias y vídeos. Además, puedes consultar previsiones marítimas y en estaciones de esquí, interesante para aquellos aficionados el deporte.

En cambio, una de las características funcionales que más llama la atención es que, a través de una opción en su menú lateral, te permite crear una postal con una foto hecha por ti y compartirla con tus amigos.

En general, una aplicación bastante completa en todos los sentidos, la única pega son los anuncios que aparecen constantemente.

3.1.3. AccuWeather

AccuWeather [7] es una de las aplicaciones meteorológicas más conocidas y mejor valoradas. Además, lleva mucho tiempo en el mercado y no solamente español, sino a nivel internacional.

Figura 6. Logo de AccuWeather

Mediante un diseño sencillo muestra, en forma de tarjetas, la información meteorológica. Ofrece detalles acerca de las condiciones climatológicas como el índice de rayos ultravioleta, el viento y la humedad, entre otras muchas.

Un inconveniente sería la publicidad que se muestra en todo momento, a no ser que se realice el pago por un servicio sin anuncios. Lo cual supone una pequeña inversión de dinero, que no todo el mundo está dispuesto a realizar. Una muestra de ello son las figuras 7 y 8.

Figura 7. Mensajes publicitarios en AccuWeather

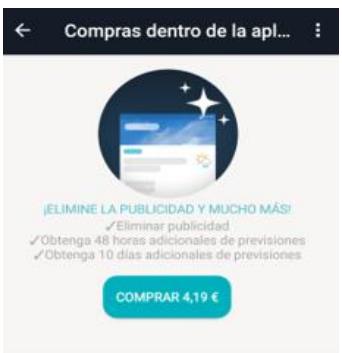

Figura 8. Opción para eliminar la publicidad en AccuWeather

3.1.4. The Weather Channel

Figura 9. Logo de The Weather Channel

The Weather Channel [8] es una cadena estadounidense de pronósticos climatológicos creada en 1982 y caracterizada por el logo de la figura 9.

En cuanto a su aplicación, posee una estructura muy sencilla: un menú desplegable para los ajustes, un menú principal en la parte superior que te permite realizar la búsqueda de municipios y otro en la parte inferior para predicciones, radar e índice de polución.

Utiliza un formato de pestañas para mostrar los datos, similar a las aplicaciones ya vistas. El tema de esta aplicación es bastante visual, se basa en el juego de sombras y en el contraste del color blanco con tonos más azulados, lo que produce mayor claridad en la información.

Existe poco que reprochar de esta aplicación y muchos detalles a los que prestar especial atención para incluirlos en la nuestra.

3.2. Estudio de características

En la tabla 1 se muestra una tabla comparativa con las características más importantes obtenidas del estudio del mercado de aplicaciones meteorológicas y si estas las cumplen o no.

Tabla 1. Comparativa de características

Características	AEMET	Eltiempo.es	AccuWeather	The Weather Channel
Diseño novedoso y atractivo	NO	SÍ	SÍ	SÍ
Predicciones diarias/horarias	SÍ	SÍ	SÍ	SÍ
Uso de la geolocalización para obtener la predicción en tu ubicación actual	SÍ	SÍ	SÍ	SÍ
Traducción	NO	SÍ	SÍ	SÍ
Gestión de favoritos	SI	SÍ	NO	NO
Notificaciones	NO	SÍ	SÍ	SÍ
Consulta de mapas y radares	SÍ	SÍ	SÍ	SÍ
Presencia de anuncios publicitarios	NO	SÍ	SÍ	SÍ
Retroalimentación	NO	SÍ	NO	NO
Opción de ayuda para el uso de la aplicación	SÍ	NO	NO	SÍ

Con respecto a la información mostrada, se puede ver que todas ellas comparten ciertas funcionalidades como: predicciones horarias y diarias, uso de geolocalización para obtener la previsión de la ubicación actual y la consulta de mapas y radares.

Las predicciones son imprescindibles para cualquier tipo de aplicación meteorológica y la geolocalización facilita el acceso rápido a la predicción de tu ubicación. Las notificaciones, mapas y radares son necesarias para darle mayor magnitud a nuestra aplicación.

La presencia de anuncios publicitarios es conveniente si quisiéramos obtener ganancias, pero en nuestro caso, no es un objetivo, ya que la intención es que las personas puedan disfrutar de la información de manera gratuita. Entonces, sería un inconveniente para nuestros usuarios.

La traducción es un aspecto destacable para tratar de llegar al mayor número de usuarios posible, por lo que debe de estar presente.

La opción de guardar en favoritos evita que se pierda tiempo en la búsqueda, de modo que si un usuario tiene la costumbre de ver la previsión de un sitio concreto habitualmente puede acceder directamente a través del menú favoritos.

Un aspecto importante de una buena aplicación es que disponga de una buena retroalimentación por parte del sistema, de modo que el usuario sepa en todo momento lo que está sucediendo, mediante el uso de mensajes informativos o ventanas de confirmación de acciones.

Además, es útil disponer de una sección de ayuda para aquellas personas que no entiendan el funcionamiento de la aplicación, bien mostrando imágenes que representen los pasos a seguir o con un apartado de ayuda al cliente en línea.

Mediante el uso del sistema MoSCoW [9] (Must have, Should have, Could have, and Won't have) priorizamos las funcionalidades a implementar en la aplicación, contemplamos aquellas que son interesantes para añadir y desechamos las que no sean necesarias. En la tabla 2 podemos ver el resultado.

Tabla 2. Sistema MoSCoW

Método MoSCoW				
Características	Debe tener	Debería tener	Podría tener	No tendría
Diseño novedoso y atractivo	X			

Predicciones diarias/horarias	X			
Geolocalización	X			
Capacidad de traducción	X			
Gestión de favoritos	X			
Notificaciones diarias	X			
Consulta de avisos climatológicos			X	
Consulta de mapas y radares		X		
Presencia de anuncios publicitarios				X
Sección de ayuda			X	
Retroalimentación (feedback)	X			

Para el desarrollo de nuestra aplicación móvil se han tenido en cuenta las características más importantes mencionadas en la tabla anterior, es decir, aquellas clasificadas en la columna “Debe tener” y también las que pertenecen a la columna “Debería tener”, para evitar que la aplicación sea muy básica.

Es imprescindible que el usuario se encuentre cómodo y no tenga dificultades durante su uso. Es probable que, si nos alejamos del estereotipo de aplicación meteorológica, no consigamos los objetivos esperados en este trabajo.

4. Especificación

En el desarrollo software una tarea muy común es la especificación de requisitos, es decir, la identificación de los requerimientos del sistema [10]. Se trata de una tarea de gran importancia que permite definir con exactitud las funcionalidades de la aplicación y ayuda a no malinterpretarlas durante su implementación.

En primer lugar, debemos distinguir entre requisitos funcionales y no funcionales.

4.1. Requisitos funcionales

Son aquellas características que requiere nuestra aplicación. Se pueden entender como el "qué" debe hacer.

- RF 1. El sistema podrá realizar una búsqueda por localidades para obtener las predicciones.
- RF 2. Se permitirá la opción de guardar o eliminar una localidad como favorita.
- RF 3. Se podrán visualizar todas las localidades favoritas previamente guardadas y podremos acceder a su predicción directamente.
- RF 4. La aplicación ofrecerá la posibilidad de obtener las predicciones de la ubicación actual del usuario.
- RF 5. El sistema tendrá una opción para visualizar mapas de rayos ultravioletas, analíticos, predictivos e imágenes de radar.
- RF 6. El usuario, si lo desea, podrá permitir el uso de notificaciones diarias con información meteorológica actual de una localidad favorita.

4.2. Requisitos no funcionales

No hacen referencia a las funciones específicas del sistema, sino a las propiedades que este debe satisfacer [11]. Suponen restricciones en su implementación.

Para su definición vamos a agruparlos en varias categorías.

4.2.1. Usabilidad

Característica relacionada con el modo de interacción entre el usuario y la aplicación.

- RNF 1. La aplicación debe estar diseñada de manera que se adapte a la resolución o al tamaño de pantalla de cualquier dispositivo móvil.

- RNF 2. Los usuarios habituados a las aplicaciones meteorológicas deben entender al instante el funcionamiento de la aplicación. El tiempo para una persona no habituada no debe ser superior a 5 minutos.
- RNF 3. La aplicación debe seguir los patrones establecidos por Google Material Design.
- RNF 4. El sistema proporcionará feedback[3] constante al usuario durante el uso de la aplicación, bien informando de un error o confirmando una acción realizada.
- RNF 5. Para aquellas tareas que no finalicen inmediatamente se deberá mostrar una barra de progreso que informe de su estado actual.
- RNF 6. La aplicación debe ser apta para usuarios de habla inglesa o español, y permitir una fácil adaptación a cualquier otro idioma.

4.2.2. Eficiencia

Capacidad para realizar adecuadamente una función.

- RNF 7. La aplicación no debe tardar más de 3 segundos en iniciarse.
- RNF 8. No demorar más de 6 segundos en la carga de información con las predicciones meteorológicas tras haber seleccionado una localidad, para que el usuario no desespere.

4.2.3. Consistencia

Característica que define a algo sólido y estable.

- RNF 9. La aplicación no debe bloquearse en ningún momento mientras se esté utilizando.

4.2.4. Integridad

Capacidad para mostrar algo sin ser distorsionado.

- RNF 10. La información mostrada debe ser auténtica y coherente con los datos que la Agencia Española de Meteorología nos proporciona a través de su API.

4.2.5. Mantenibilidad

Facilidad para ser modificado.

- RNF 11. El código de la aplicación debe estar bien estructurado, utilizando patrones arquitectónicos, a modo que si hay que añadir o corregir alguna funcionalidad tardemos el menor tiempo posible en solucionarlo.

[3] Feedback: del inglés, retroalimentación, que proporciona respuestas.

4.2.6. Compatibilidad

Cualidad de un elemento que le permite poder trabajar con otro correctamente.

- RNF 12. La aplicación debe funcionar con la versión 22 de la API Android y superiores.

4.3. Diagrama de casos de uso

Tras haber identificado los requisitos vamos a ayudarnos de los diagramas de casos de uso para especificar el comportamiento deseado de la aplicación.

Un diagrama de casos de uso describe un conjunto de secuencias de acciones para alcanzar cierto objetivo [12]. Estas acciones son llevadas a cabo por un personaje, conocido como actor.

En las figuras 10 y 11, se han representado los diagramas de casos de uso mediante notación UML y a través de la herramienta BoUML.

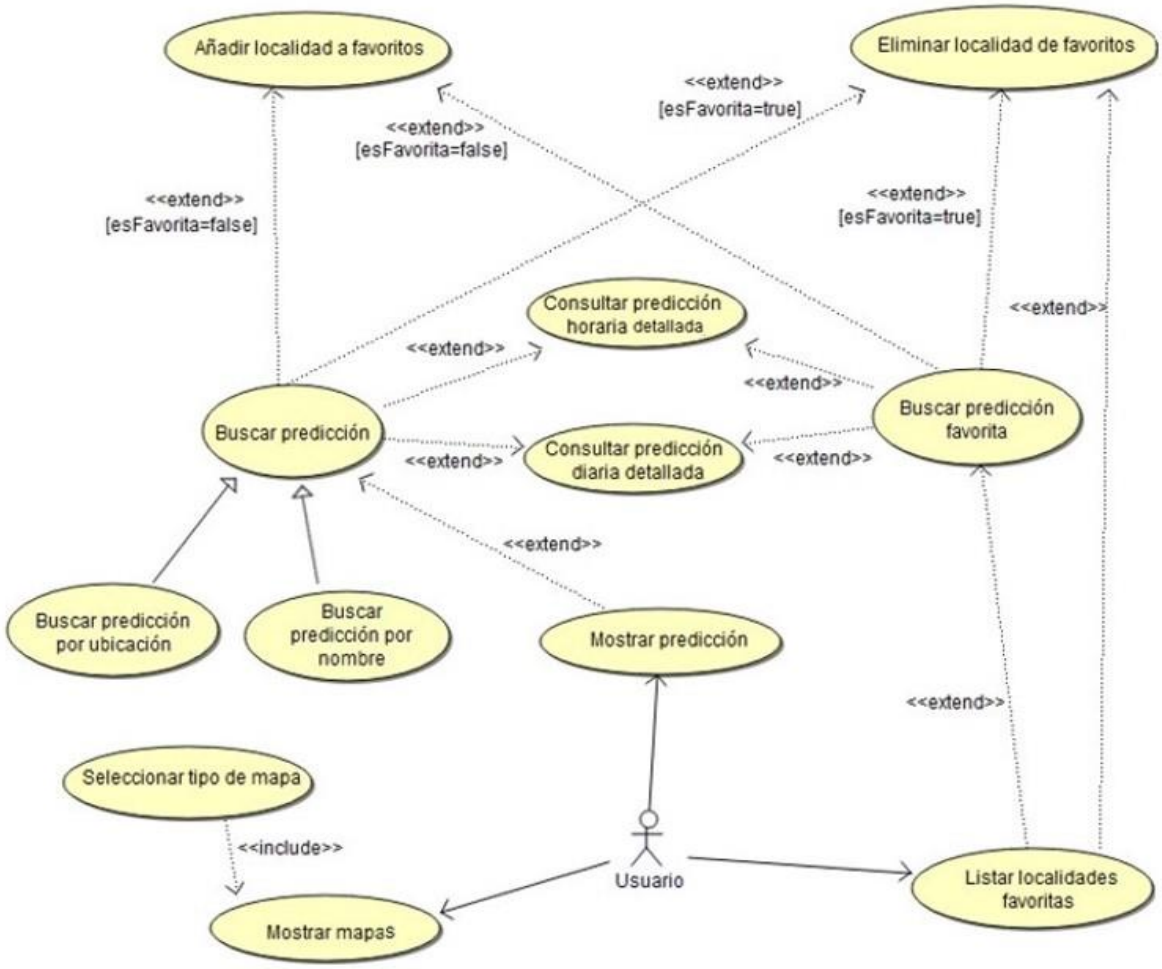

Figura 10. Consulta de información meteorológica

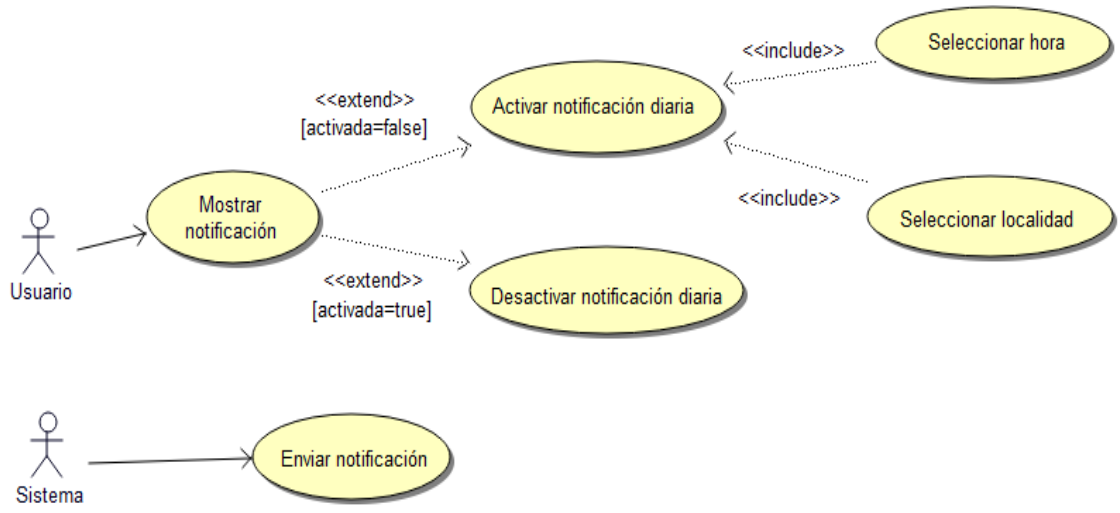

Figura 11. Gestión de notificaciones

En estos diagramas aparecen dos actores. Por un lado, si nos fijamos en la figura 10, el actor usuario puede consultar la predicción meteorológica de una localidad, bien buscándola por nombre o bien geolocalizando su posición actual. Además, puede acceder al detalle de cada una de estas predicciones, añadir, eliminar o listar las localidades favoritas y visualizar mapas climatológicos.

Por otro lado, en la figura 11, que representa la gestión de las notificaciones, el usuario puede visualizar la notificación activa si existe alguna, y configurarla (activarla o desactivarla). El actor sistema es quien envía la envía después de que este la haya configurado.

4.4. Especificación de los casos de uso

En este apartado vamos a describir más en detalle los casos de uso. A partir de este momento, hablaremos de CU cuando nos queramos referir a ellos.

Hemos decidido omitir la especificación de algunos CU para no extender demasiado este apartado. A continuación, veremos los más significativos.

Tabla 3. CU Mostrar predicción

CU	Mostrar predicción	**1**
Descripción	Muestra una nueva ventana que permite obtener las predicciones de una localidad.	
Actor	Usuario	

Pre-condición	• Estar situado sobre el menú horizontal desplegable.
Flujo básico	1. El usuario selecciona la opción “Inicio”.
Flujo alternativo	Si selecciona cualquier otra opción se abrirá otra ventana con otra funcionalidad.
Post-condición	• Se accede a la funcionalidad de las predicciones.

Tabla 4. CU Consultar predicción por búsqueda

CU	Buscar predicción por nombre	**2**
Descripción	Obtiene información meteorológica de una localidad española.	
Actor	Usuario	
Pre-condición	• Tener acceso a Internet. • Tener acceso a la API de AEMET.	
Flujo básico	1. El usuario introduce una localidad en el buscador. 2. Pulsa botón “Buscar” o selecciona una localidad sugerida.	
Flujo alternativo	Si no hay conexión a Internet, se muestra un mensaje de advertencia. Si la localidad introducida no existe, se muestra un mensaje de advertencia.	
Post-condición	• Se muestra la predicción meteorológica de dicha localidad.	

Tabla 5. CU Consultar predicción por ubicación actual

CU	Buscar predicción por ubicación actual	**3**
Descripción	Geolocaliza la posición del usuario y obtiene la predicción meteorológica.	
Actor	Usuario	
Pre-condición	• Tener acceso a Internet. • Tener acceso a la API de AEMET • Conceder el permiso de ubicación a la aplicación.	

Flujo básico	1. El usuario selecciona el botón de geolocalización. 2. El usuario acepta el permiso de ubicación solicitado por la aplicación.
Flujo alternativo	Si no hay conexión a Internet, se muestra un mensaje de advertencia. Si la ubicación actual de usuario no existe en la base de datos de la aplicación, se muestra un mensaje de advertencia. Si no se concede el permiso no se podrá cargar la predicción.
Post-condición	• Se muestra la predicción meteorológica de dicha localidad.

Tabla 6. CU Añadir localidad a favoritos

CU	Añadir localidad a favoritos	**4**
Descripción	Guarda la localidad actual en la lista de favoritos.	
Actor	Usuario	
Pre-condición	• Disponer en ese momento de la predicción de una localidad en pantalla. • La localidad mostrada no es favorita.	
Flujo básico	1. Busca la predicción de una localidad. 2. Pulsa el botón de añadir a favoritos.	
Flujo alternativo	Si el usuario pulsa el botón de añadir a favoritos sin haber seleccionado ninguna localidad. Se muestra un mensaje advirtiendo de que antes debe seleccionar una. Si la localidad ya es favorita, entonces se elimina de favoritos.	
Post-condición	• El icono de favoritos se activa y se añade a la lista de favoritos.	

Tabla 7. CU Eliminar localidad de favoritos

CU	Eliminar localidad de favoritos	**5**
Descripción	Elimina la localidad actual de la lista de favoritos.	
Actor	Usuario	
Pre-condición	• Disponer en ese momento de la predicción de una localidad en pantalla. • La localidad mostrada es favorita.	

Flujo básico	1. El usuario se sitúa sobre el menú de inicio. 2. Busca una localidad. 3. Pulsa el botón de eliminar de favoritos.
Flujo alternativo	Si el usuario pulsa el botón de eliminar de favoritos sin haber seleccionado ninguna localidad, se muestra un mensaje advirtiendo de que antes debe seleccionar una localidad favorita. Si la localidad no es favorita, entonces se añade a favoritos.
Post-condición	• El icono de favoritos se desactiva y se elimina de la lista de favoritos.

Tabla 8. CU Consultar predicción horaria detallada

CU	Consultar predicción horaria detallada	**6**
Descripción	Obtener información climatológica más precisa de una hora del día actual o de mañana.	
Actor	Usuario	
Pre-condición	• La predicción de una localidad ha sido cargada previamente.	
Flujo básico	1. Accede a la pestaña “Horaria” 2. Selecciona cualquier elemento (hora) de la lista vertical.	
Flujo alternativo	No aplica	
Post-condición	• Se abre una nueva ventana con información climática de la hora escogida.	

Tabla 9. CU Consultar predicción diaria detallada

CU	Consultar predicción diaria detallada	**7**
Descripción	Obtener información climatológica más en detalle de un día de la semana.	
Actor	Usuario	
Pre-condición	• La predicción de una localidad ha sido cargada previamente.	
Flujo básico	1. Accede a la pestaña “Diaria”. 2. Selecciona cualquier elemento (día) de la lista horizontal.	

Flujo alternativo	No aplica
Post-condición	• Se abre una nueva ventana con información climática del día escogido.

Tabla 10. CU Listar localidades favoritas

CU	Listar localidades favoritas	**8**
Descripción	Muestra una nueva ventana con un listado de todas las localidades guardadas como favoritas.	
Actor	Usuario	
Pre-condición	• Estar situado sobre el menú horizontal desplegable	
Flujo básico	1. El usuario selecciona la opción “Favoritos”.	
Flujo alternativo	Si selecciona cualquier otra opción se abrirá otra ventana con otra funcionalidad.	
Post-condición	• Se accede a la funcionalidad de los favoritos.	

Tabla 11. CU Buscar predicción favorita

CU	Buscar predicción favorita	**9**
Descripción	Obtiene la predicción meteorológica de la localidad escogida.	
Actor	Usuario	
Pre-condición	• Tener como mínimo una localidad favorita guardada.	
Flujo básico	1. El usuario pulsa una localidad de la lista.	
Flujo alternativo	No aplica.	
Post-condición	• Se muestra la predicción meteorológica de dicha localidad.	

Tabla 12. CU Mostrar mapas

CU	Mostrar mapas	**10**
Descripción	Muestra una nueva ventana que permite visualizar imágenes de radar y mapas climatológicos.	

Actor	Usuario
Pre-condición	• Estar situado sobre el menú horizontal desplegable.
Flujo básico	1. Selecciona la opción “Mapas” del menú desplegable.
Flujo alternativo	Si selecciona cualquier otra opción se abrirá otra ventana con otra funcionalidad.
Post-condición	• Se accede a la funcionalidad de los mapas.

Tabla 13. CU Seleccionar tipo de mapa

CU	Seleccionar tipo de mapa	**11**
Descripción	Visualiza mapas e imágenes de radar con una pequeña descripción.	
Actor	Usuario	
Pre-condición	• Tener acceso a Internet • Tener acceso a la API de AEMET	
Flujo básico	1. Elige un tipo de mapa o imagen de radar entre las opciones.	
Flujo alternativo	Si no hay acceso a Internet no se completa la carga. Si la API no da servicio no se completa la carga.	
Post-condición	• Aparecen los mapas o imágenes de radar y su descripción.	

Tabla 14. CU Mostrar notificación

CU	Mostrar notificación	**12**
Descripción	Muestra una nueva ventana que permite gestionar una notificación.	
Actor	Usuario	
Pre-condición	• Estar situado sobre el menú horizontal desplegable.	
Flujo básico	1. Selecciona la opción “Notificaciones” del menú desplegable.	
Flujo alternativo	Si selecciona cualquier otra opción se abrirá otra ventana con otra funcionalidad.	
Post-condición	• Se accede a la funcionalidad de las notificaciones.	

Tabla 15. CU Activar notificación diaria

CU	Activar notificación diaria	**13**
Descripción	Permite activar una notificación diaria.	
Actor	Usuario	
Pre-condición	• Tener desactivada la notificación. • Haber seleccionado una hora. • Haber seleccionado una localidad.	
Flujo básico	1. El usuario selecciona una hora. 2. El usuario selecciona una localidad. 3. Activa el botón de notificación.	
Flujo alternativo	Si el usuario tiene la notificación activada, la desactiva.	
Post-condición	• La alarma se muestra en el dispositivo móvil a la hora seleccionada.	

Tabla 16. CU Desactivar notificación diaria

CU	Desactivar notificación diaria	**14**
Descripción	Permite desactivar la notificación diaria.	
Actor	Usuario	
Pre-condición	• Tener activada la notificación.	
Flujo básico	1. Pulsa el botón de notificación.	
Flujo alternativo	Si el usuario tiene la notificación desactivada, la activa.	
Post-condición	• La alarma se desactiva.	

Tabla 17. CU Enviar notificación

CU	Enviar notificación	**15**
Descripción	Envía una notificación informando del clima de una localidad.	
Actor	Sistema	

Pre-condición	• Tener acceso a Internet y a la API. • El usuario debe haber activado la notificación.
Flujo básico	No aplica.
Flujo alternativo	Si no hay conexión a internet ni a la API la notificación no se crea.
Post-condición	• La notificación aparece en el dispositivo móvil a la hora indicada con información climática de la localidad escogida.

4.5. Mockups

A la hora de diseñar la interfaz de una aplicación es común el prototipado o el uso de mockups[4], para representar gráficamente las ventanas que la componen.

Durante el proceso de ingeniería del software, es una técnica muy utilizada que ofrece a los futuros usuarios o al resto de stakeholders[5], una visión estimada del producto [13]. En nuestro caso, nos ayuda a evitar pérdidas de tiempo innecesarias en la fase de implementación, planificando previamente el diseño y la comunicación entre las distintas ventanas de la aplicación. Sin pasar por alto alguna funcionalidad acordada.

En la representación de cada mockup, hemos señalado los elementos de la ventana que disparan los casos de uso.

[4] Mockup: Representación parcial de la interfaz del sistema, en forma de boceto.
[5] Stakeholder: Persona interesada en el producto en desarrollo.

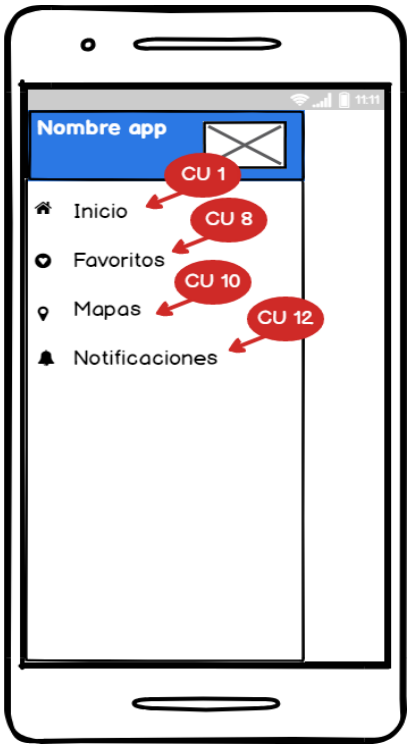

Figura 12. Menú desplegable horizontal

Menú horizontal

Este menú desplegable, que podemos ver en la figura 12, hace la función de ventana controladora desde donde podemos acceder al resto de funcionalidades de la aplicación.

Figura 13. Ventana de inicio con predicción actual

Ventana de inicio (Pestaña hoy)

Desde la ventana de inicio podemos acceder a las predicciones climáticas de una localidad, seleccionando una de estas tres pestañas: “Hoy”, “Horaria” o “Diaria”.

Si nos fijamos en la figura 13, en la barra de acciones superior, hay un buscador que nos permite introducir la localidad deseada.

El recuadro de tono grisáceo contiene un campo de texto, donde se cargará el lugar buscado, y un botón a su derecha que permitirá añadirlo o eliminarlo de la lista de favoritos.

Más abajo se mostrarán la información a través de diversos campos de texto y una imagen que mostrará el estado del cielo.

Figura 14. Ventana de inicio con predicción horaria

Ventana de inicio (Pestaña horaria)

La pestaña de predicción horaria contiene una lista de elementos seleccionables formados por dos campos textuales con la hora y la temperatura y una imagen en el centro. Además de una barra que permite deslizar verticalmente, como podemos ver en la figura 14.

Figura 15. Ventana de información horaria detallada

Información horaria detallada

Esta ventana, representada con la figura 15, es el resultado de haber seleccionado uno de los elementos de la lista anterior. Muestra de manera más minuciosa la información climatológica en cierta hora del día.

Figura 16. Ventana inicial con predicción diaria

Ventana de inicio (Pestaña diaria)

Como podemos observar en la figura 16, esta pestaña muestra la predicción diaria de la localidad seleccionada a través de una lista horizontal de elementos seleccionables, con un campo de texto que representa al día de la semana y un icono representando el estado.

Más abajo se muestran unas gráficas que proveen más información.

Figura 17. Ventana de información diaria detallada

Información diaria detallada

Tras haber seleccionado un elemento de la lista anterior nos aparece esta ventana, similar a la figura 17, con una imagen y un texto descriptivo en la parte superior y una serie de datos climatológicos del día en concreto.

Figura 18. Ventana de favoritos

Ventana de favoritos

En la figura 18 se muestran, tras haber seleccionado la opción Favoritos del menú, todas las localidades favoritas guardadas previamente. Cada uno de estos elementos está formado por un texto con el nombre de dicha localidad y un botón que permite eliminarlo de la lista.

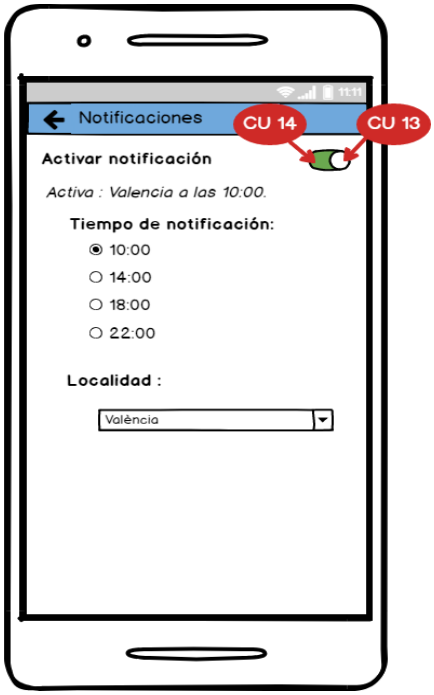

Figura 19. Ventana de notificaciones

Ventana de notificaciones

Desde aquí podemos configurar las notificaciones de la aplicación. Se compone de un botón para activar o desactivar las notificaciones y dos grupos de controles de selección única, para elegir la hora y la localidad a notificar. En la figura 19 podemos contemplar estos detalles.

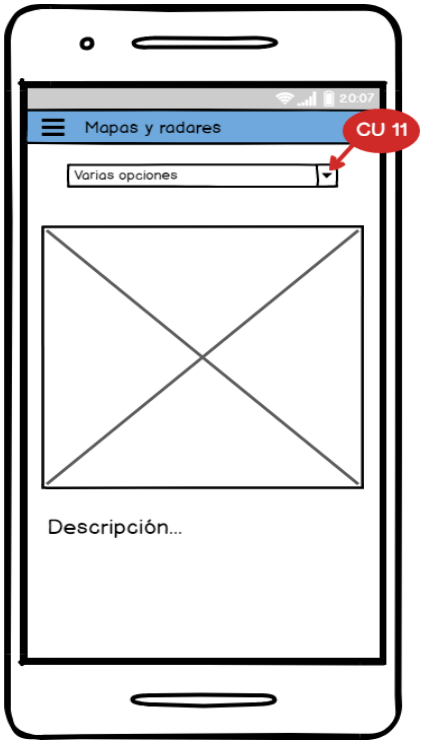

Figura 20. Ventana de mapas

Ventana de mapas

Aquí se muestran por medio de imágenes: mapas significativos, de análisis y capturas de radares previstos por AEMET.

El menú desplegable de la parte superior, que vemos en la figura 20, muestra distintos tipos de mapas para elegir y en la parte inferior se muestra una pequeña descripción de la imagen.

4.6. Modelo de dominio

El modelo de dominio nos permite apreciar de manera directa las relaciones entre las distintas entidades de la aplicación. En la figura 21 se representan dichas entidades.

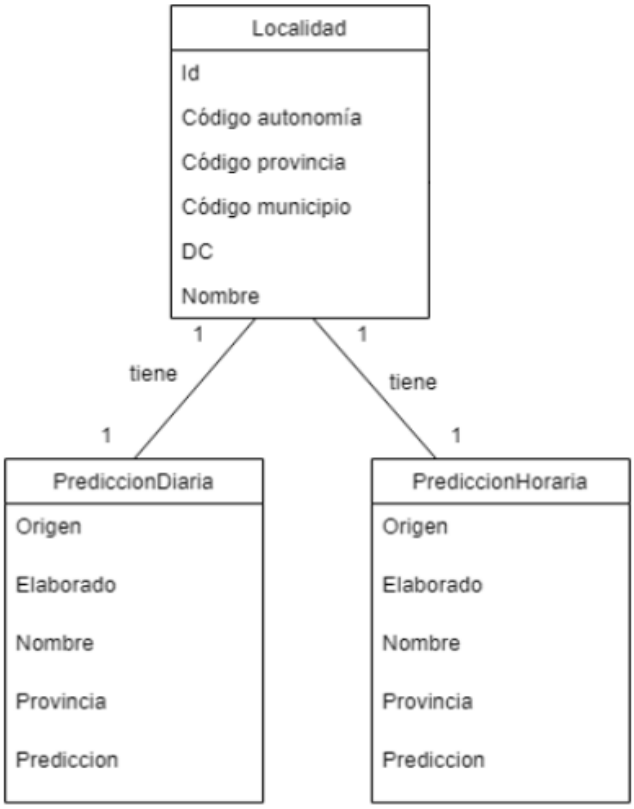

Figura 21. Modelo de dominio

En primer lugar, podemos observar la entidad Localidad que contiene una serie de atributos (Código autonomía, código provincia, código municipio y DC) necesarios para formular la petición a la API y obtener las predicciones.

Las entidades PrediccionDiaria y PrediccióнHoraria contienen la información que extraemos de la API de AEMET. Concretamente el atributo Prediccion es la fuente de todos los datos que se muestran en la aplicación. Los atributos Nombre y Provincia hacen referencia a la localidad.

Existen más clases diseñadas para poder obtener toda la información proporcionada por la API: temperatura, pluviosidad, estado del cielo, etc. Sin embargo, al ser demasiadas, se ha decidido obviarlas para no complicar el diagrama.

Más en adelante, estudiaremos la arquitectura de la base de datos.

5. Diseño

5.1. Esquema de componentes

En la figura 22 se muestran los principales componentes de nuestra aplicación.

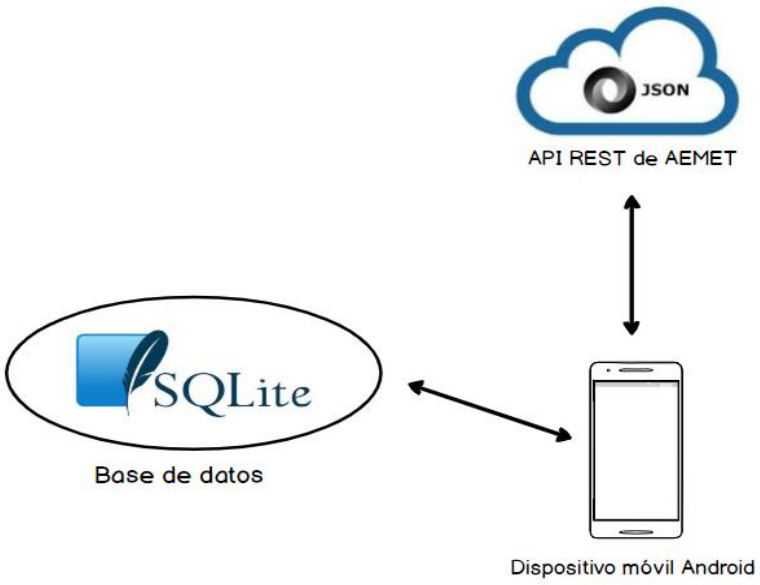

Figura 22. Arquitectura general de la aplicación

Posee una base de datos SQLite donde tenemos almacenada información de localidades de las cuales podemos obtener una predicción climatológica. De esta manera, podemos realizar consultas a la base de datos para poder obtener los códigos de cada localidad y así, poder realizar la petición a la API con la información meteorológica.

Como hemos dicho, la API de AEMET provee la información requerida. A través del servicio REST[6], realizamos una serie peticiones a dicha API, y esta nos envía los datos en formato JSON[7], los cuales son tratados y posteriormente mostrados al usuario.

5.2. Arquitectura

La arquitectura software se corresponde con el diseño de más alto nivel de la estructura de un sistema.

[6] REST: protocolo de manipulación de datos.
[7] JSON: formato de texto simple utilizado para el intercambio de información.

En el libro Clean Arquitecture [14]**,** escrito Robert C. Martin, se habla de la importancia de la arquitectura y los beneficios que proporciona. El autor cita ejemplos reales tales como un proyecto real, que a medida que aumentaba su tamaño, el coste por línea de código aumentaba hasta 100 veces, al no haber cuidado la arquitectura desde un principio. De esta manera, podemos llegar a entender la importancia que conlleva.

Según "Uncle Bob", así es como se conoce al autor de este libro, una "arquitectura limpia" [15] se caracteriza por:

- Ser independiente de cualquier framework.
- Ser independiente de la IU.
- Ser independiente de la base de datos.
- Ser testeado con facilidad.

Una arquitectura influye a la hora de desarrollar nuevos módulos y en la manera de organizar el código. Dependiendo del tipo de aplicación que queramos desarrollar es conveniente utilizar un tipo de arquitectura software u otra.

5.2.1. Arquitectura de la aplicación

Nuestra aplicación está construida en torno a una arquitectura de 3 capas. Esto quiere decir que está organizada en 3 subsistemas o partes. Cada una de ellas hace referencia a componentes de la capa inferior y a la vez sirve como base de implementación de la capa superior.

Dichos subsistemas están representados en la figura 23.

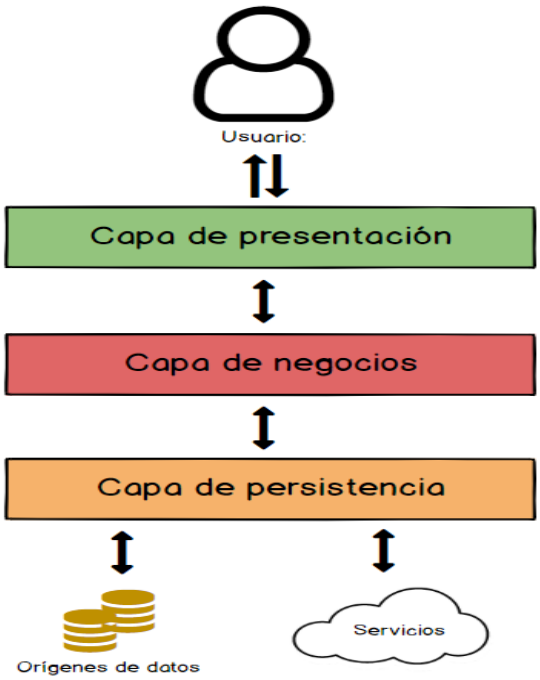

Figura 23. Arquitectura de 3 capas de nuestra aplicación

- La capa de presentación muestra el sistema al usuario mediante una interfaz. Captura la interacción con el usuario y muestra los resultados.
- La capa intermedia o lógica de negocios proporciona la funcionalidad de la aplicación. Recibe solicitudes de la capa de presentación y se comunica con la capa de datos para almacenar o recuperar información.
- La capa de persistencia o capa de acceso es donde residen los datos. Proporciona mecanismos para la obtención o el almacenamiento de información desde la capa de negocio.

Como podemos observar, el usuario interactúa con la capa de presentación, a través de una interfaz. Esta envía peticiones a la capa de la lógica de negocio, que obtiene la información necesaria de la capa de persistencia, ya sea de una base de datos o de otro servicio.

El trabajar con esta arquitectura nos ofrece mayor flexibilidad a la hora de añadir nuevas funcionalidades a la aplicación, ya que al tener los módulos separados y organizados es más sencillo saber dónde debemos realizar las inserciones de código. Además, aumenta la capacidad de mantenibilidad de la aplicación, haciendo que el tiempo para ubicar el fallo sea menor.

Del mismo modo, si necesitamos cambiar la base de datos, es decir, el modelo de datos, el resto de las capas no se verían afectadas. Por otro lado, si en lugar de una interfaz para teléfonos móviles queremos una para SmartTV o SmartWatch, solo tenemos que cambiar la capa de presentación.

Aunque en este trabajo todo el peso de la implementación haya caído sobre una persona, sería muy beneficioso si se elaborara de manera conjunta, es decir, varios desarrolladores. De esta manera, se podría trabajar en paralelo desarrollando a la vez las 3 capas y llegado el momento, enlazar las partes.

En resumen, trabajar con una arquitectura nos permite tener un proyecto mantenible. Una vez dicho esto, en el siguiente punto, vamos a ver más en detalle cómo está construida nuestra aplicación.

5.2.2. Patrón modelo-vista-presentador (MVP)

Para el desarrollo de nuestra aplicación hemos utilizado el patrón modelo-vista-presentador. Es un patrón arquitectónico muy empleado en el desarrollo de interfaces de usuario. Se trata de una derivación del modelo-vista-controlador (MVC), que nos permite separar las vistas de la lógica de la aplicación [16].

De este modo, todo el peso de la lógica cae sobre el presentador y las consultas a bases de datos, APIs u otros servicios, al modelo.

De manera general podemos representar este patrón mediante la figura 24.

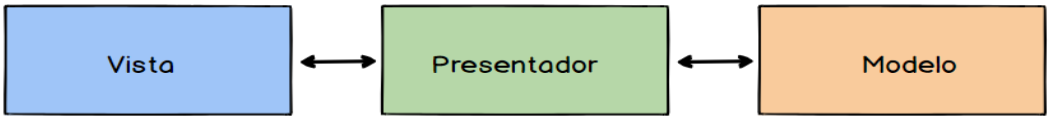

Figura 24. Patrón MVP

La vista es una interfaz que va mostrando datos al usuario durante su interacción con él. Cualquier acción sobre la interfaz es notificada al presentador. Esta capa no contiene nada de la lógica de la aplicación.

El presentador se sitúa entre la vista y el modelo, conectando la interfaz con los datos. Ante las notificaciones de la vista, el presentador es el encargado de gestionar la información y transformar los datos necesarios, ya sea almacenando nuevos registros en la base de datos o modificando la vista para mostrar una lista de favoritos. Por ello, tiene acceso a la capa de acceso a datos o modelo. En MVP toda la lógica de la aplicación se coloca en el presentador.

Por último, el modelo es el que contiene las operaciones necesarias para gestionar los datos.

En nuestra aplicación hemos aplicado una variante de este modelo-vista-presentador, introduciendo además el personaje "interactor". Este contiene el código desarrollado para completar los casos de uso de aplicación, es decir, las acciones que podemos realizar en ella: guardar un favorito, mostrar la predicción, etc. Esto supone el desacoplamiento del presentador disminuyendo sus líneas de código y haciéndola más limpia.

De este modo, la estructura del patrón viene representada por la figura 25.

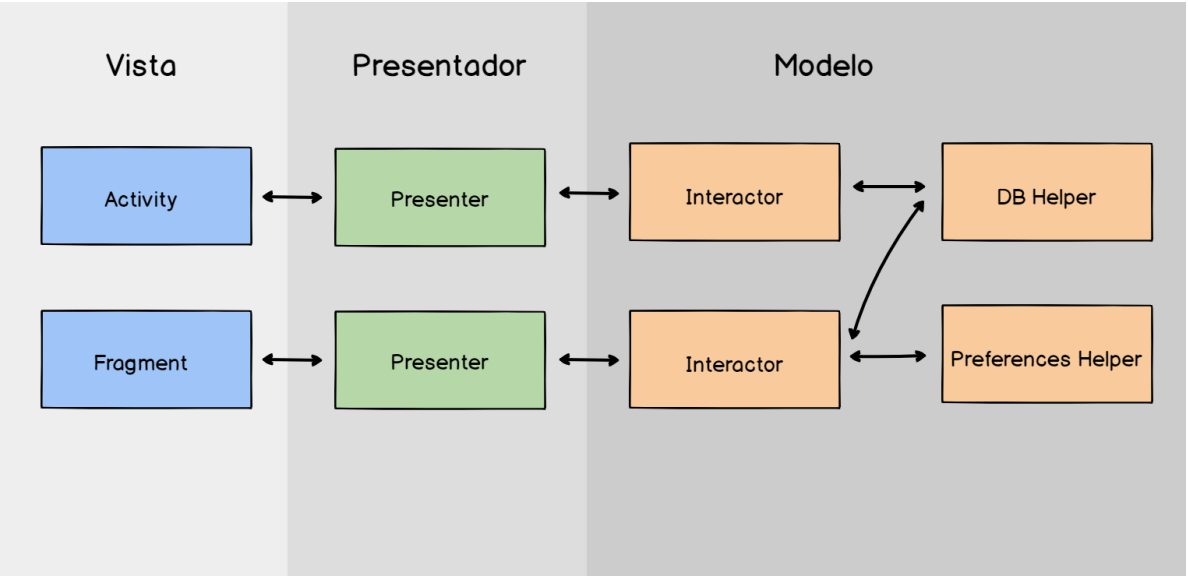

Figura 25. Adaptación al patrón MVP con interactor

En este caso, las activities[8] y fragments[9] que conforman la vista de nuestra aplicación poseen su propio presentador. Cada presentador tiene una clase interactor asociada, con una serie de métodos que corresponden con los casos de uso y son accesibles desde el presentador. Estos acceden a las fuentes de información: base de datos, preferencias de Android o a la API REST.

Las entidades DB Helper y Preferences Helper contienen los métodos necesarios para insertar, eliminar o modificar datos del modelo. Además, el interactor tiene la funcionalidad para comunicarse con el servicio API web de AEMET.

Utilizar este tipo de arquitectura nos ha permitido disminuir la aglomeración de código, evitando así clases con demasiadas líneas de código, las cuales son muy pesadas y difíciles de tratar. Vamos

[8] Activity: componente la aplicación que contiene una pantalla con la que el usuario interactúa.
[9] Fragment: porción de una interfaz de usuario, que puede trabajar de manera independiente.

a ver un ejemplo, utilizando una comparativa del número de líneas antes y después de aplicar el patrón MVP en una de las vistas principales de nuestra aplicación.

ANTES

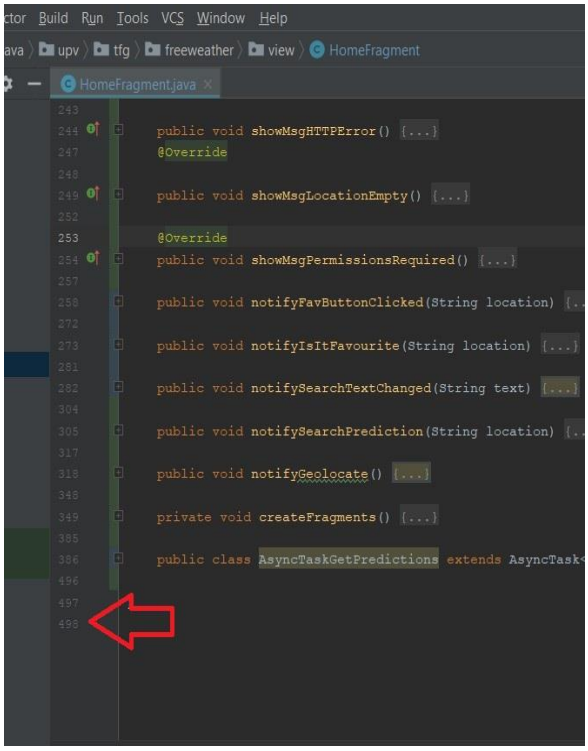

Figura 26. Ventana Home sin MVP

DESPUÉS

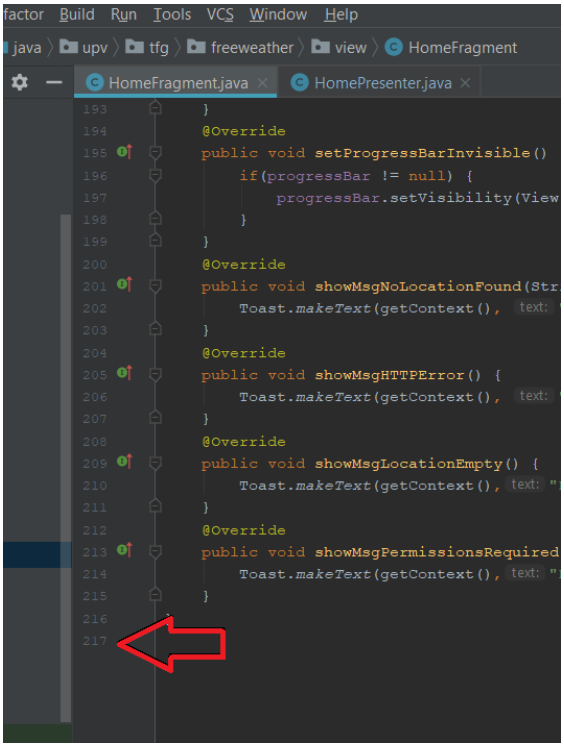

Figura 27. Ventana Home con MVP

Si nos fijamos en las figuras 26 y 27, podemos observar que tras haber extraído la lógica de la clase HomeFragment, perteneciente a la vista, e introducirla en otra clase denominada HomePresenter, que pertenece a la figura del presentador, las líneas de código se han decrementado más de la mitad. Obteniendo así, clases ligeras y fáciles de manipular.

5.3. Estructura del proyecto

Tras la aplicación del patrón modelo-vista-presentador hemos decidido organizar el proyecto de una manera en la que se puedan distinguir fácilmente las capas que conforman la arquitectura de nuestra aplicación. La estructura se muestra en la figura 28.

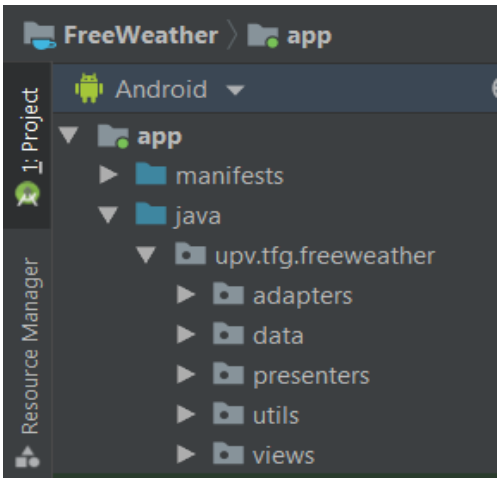

Figura 28. Estructura general del proyecto Android

El proyecto consta de un total de 5 carpetas. La capa de datos está representada por la carpeta "data", representada en la figura 29. En ella se encuentran los "interactors" o mediadores con los casos de uso de la aplicación, la base de datos ("local") y las entidades serializadas ("model") que contienen la información meteorológica, es decir el modelo de la aplicación. Además, aquí también se encuentra el código necesario para realizar las llamadas a la API. Una especie de conexión con el servicio REST de AEMET que veremos más en detalle en el apartado 7.2.

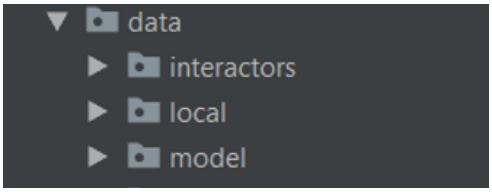

Figura 29. Contenido de la carpeta data

Las carpetas "views" y "presenters" contienen las vistas y sus respectivos presentadores que comunican con el modelo. Las vistas están divididas en las subcarpetas "fragments" y "activities".

Para mostrar la información de manera más visual y entendible se ha utilizado lo que en mundo Android se conoce como Adapter, un mecanismo de Android que permite mostrar e interactuar con información a través listados y de manera eficiente. Hemos decidido agrupar todos ellos en la carpeta "adapters".

Finalmente, en la carpeta "utils" se encuentran aquel código de utilidad que nos han hecho falta durante la implementación, y que se ha decidido extraer en un lugar aparte para disminuir el tamaño de ciertas clases.

5.4. Arquitectura de base de datos

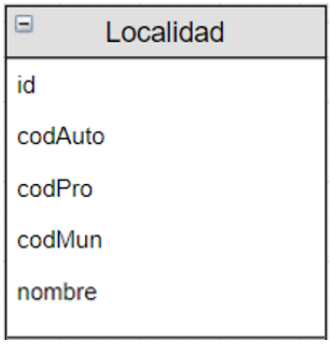

Figura 30. Modelo de datos

Esta aplicación utiliza una base de datos SQLite, compuesta por una tabla denominada Localidad, representada en la figura 30. Dicha tabla contiene los siguientes atributos:

- id: es el identificador único de la localidad, autogenerado al crear la base de datos.
- codAuto: es el código de la autonomía a la que pertenece dicha localidad.
- codPro: representa al código de la provincia al que pertenece.
- codMun: es el código del municipio.
- nombre: es el nombre con el que se conoce a la localidad.

Se crea al lanzar la aplicación por primera vez, por lo que el tiempo de carga suele ser algo más costoso.

Su funcionalidad es proporcionar los códigos de la localidad a partir del nombre introducido en el buscador de localidades de la aplicación.

Existe mucha información meteorológica obtenida de la API, que hemos decidido no almacenar en base de datos, ya que no se ha visto necesario mostrar el clima de días anteriores a la fecha actual. Por lo que tampoco es necesario guardar dicha información.

Otra manera de persistir datos es mediante las preferencias de Android. Dichas preferencias es información que se almacena, durante la interacción del usuario con la aplicación, en un archivo XML, y no en una base de datos como el caso anterior.

Dichos datos son la gestión de las notificaciones activas, y el estado actual de la aplicación.

6. Herramientas y tecnologías

En esta sección vamos a tratar las herramientas que hemos ido utilizando durante el desarrollo de la aplicación. A través de la figura 31, podemos visualizar las principales tecnologías.

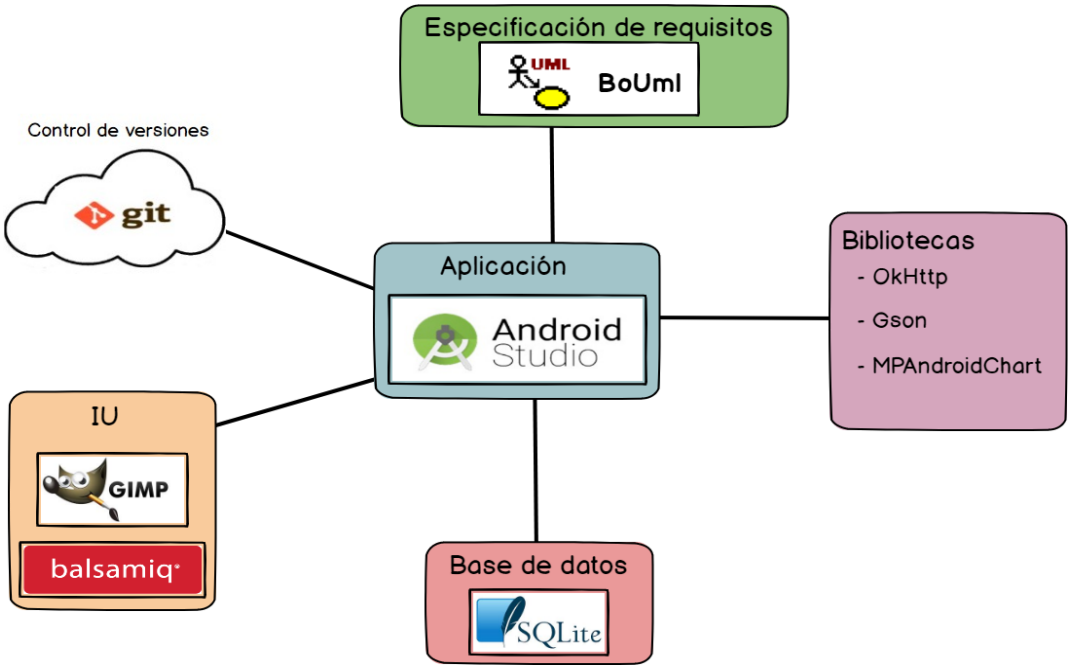

Figura 31. Herramientas y tecnologías utilizadas

6.1. Android Studio

Es el entorno de desarrollo integrado (IDE) oficial de Android [17]. Su primera versión fue lanzada en 2014. Está basado en Intellij IDEA, un software para el desarrollo de aplicaciones. Permite compilar, ejecutar y depurar aplicaciones, entre otras muchas funcionalidades. También dispone de un emulador propio.

Es un framework que conozco desde hace tiempo y con el que he ido cogiendo soltura estos últimos años, razón que me ha llevado a elegirlo cómo principal herramienta de programación en este trabajo.

6.2. SQLite

Es la base de datos más usada en el mundo. SQLite [18] es un sistema de bases de datos relacional, escrito en lenguaje C. Esta librería implementa la mayoría de las funciones del estándar SQL-92, incluye transacciones atómicas, consistencia, aislamiento, modelo de transacciones ACID, consultas complejas, etc.

Es totalmente gratuita, de dominio público y no requiere instalación alguna. Actualmente cuenta con un trillón de bases de datos en activo.

6.3. GitHub

GitHub [19] es una plataforma de trabajo colaborativo que permite gestionar las versiones de un producto software. Proporciona herramientas para la gestión y el seguimiento del trabajo, permitiendo almacenar, descargar, revisar y trabajar en paralelo, entre otras muchas funciones.

En nuestro caso, nos ha permitido tener la certeza de que el código que hemos ido implementando estaba a salvo, y también nos ha facilitado la revisión del código en ciertos puntos del desarrollo de la aplicación.

Android Studio la tiene integrada a través de un menú de opciones, por lo que nos ha facilitado mucho su utilización.

6.4. BoUml

Es una herramienta de modelado UML que permite realizar todo tipo de diagramas: casos de uso, diagramas de secuencias, clases, etc. BoUml [20] permite la opción de generar código en diversos lenguajes, aunque en nuestro caso no le hemos dado uso. Sin embargo, nos ha permitido diseñar el diagrama de casos de uso de nuestra aplicación.

6.5. GIMP

GIMP [21] es un programa para el tratamiento de imágenes, con cierto nivel profesional. Es una buena alternativa para aplicaciones de pago como Photoshop, ya que es gratuita y ofrece buenos servicios.

Durante trabajo se ha utilizado para crear, editar y retocar todas las imágenes, iconos meteorológicos y el logo de la aplicación.

6.6. Balsamiq

Balsamiq [22] es una herramienta para el diseño de prototipos. Nos permite crear, de manera rápida y sencilla, mockups que definan la interfaz de nuestra aplicación. Es de pago, aunque el primer mes de uso es gratuito.

6.7. Bibliotecas

- OkHttp [23]: Es una librería de código abierto que facilita la gestión de las comunicaciones HTTP con una API, la sincronización de hilos de ejecución en paralelo y la gestión de colas de peticiones, proporcionando una serie de métodos para ello.

- Gson [24]: Nos permite la transformación del código JSON obtenido de la API a entidades del sistema, como una predicción del tipo horaria o diaria.
- MPAndroidChart [25]: Es una librería software gratis que permite construir todo tipo de gráficas. Desde diagramas de barras o líneas hasta gráficos animados y escalables.

7. Implementación

En este apartado vamos a comentar en detalle el desarrollo de las funcionalidades de la aplicación y las razones pertinentes que nos han llevado a implementarlo de esta manera.

7.1. Ventanas de la aplicación

Para su mejor comprensión, hemos dividido la aplicación en varios bloques según su utilidad.

7.1.1. Navigation drawer

Desde el primer momento se ha buscado dotar de mayor accesibilidad y sencillez a nuestra aplicación. La solución ha sido utilizar un navigation drawer [26]. Es un patrón de diseño perteneciente al grupo de Material Design[10] de Google, que provee una navegación rápida entre ventanas, una buena organización y un aspecto más profesional. Se trata de un menú horizontal desplegable con diversas opciones, en nuestro caso, 4 opciones.

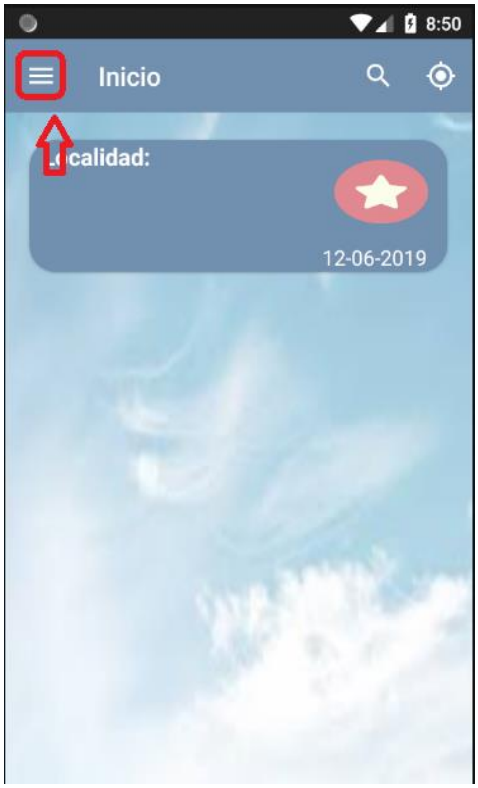

Figura 32. Ventana de inicio

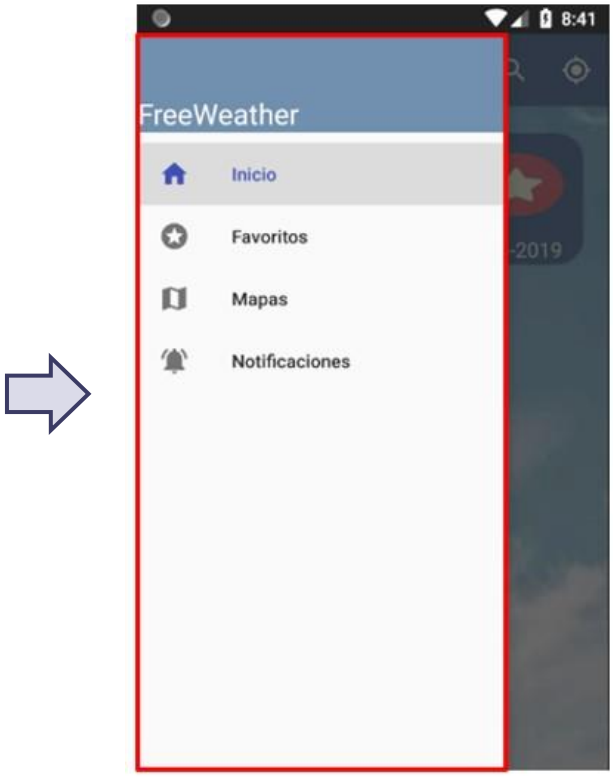

Figura 33. Menú desplegable

Como se observa en la figura 33, el menú desplegable está compuesto por:

- Cabecera: con el nombre y el logo de aplicación.

[10] Material Design: guía para el diseño visual en Android, creada por Google.

- Listado de opciones:
    - Inicio: ventana de consulta de predicciones meteorológicas.
    - Favoritos: consulta de favoritos y acceso directo a predicciones.
    - Mapas: visualización de mapas e imágenes de radar.
    - Notificaciones: para la activación de avisos en un rango horario.

Una manera de crear diseños de interfaces de usuario más dinámicos es mediante el uso de fragmentos o *fragments*, que permiten modificar parte de una vista sin alterar al resto. El navigation drawer está compuesto por una actividad principal, la cual alberga diferentes fragmentos. De este modo funciona como una especie de contenedor, reemplazando el contenido actual por el contenido de otra opción escogida del menú desplegable. Tal y como podemos observar en la figura 34.

La figura 35 indica que cada menú de opciones posee su propio fragmento.

Figura 34. Contenedor de fragmentos

Figura 35. Opciones del navigation drawer

7.1.2. Ventana de inicio

Tras iniciar la aplicación, se muestra la ventana de inicio por defecto. Esta vista se corresponde con la figura 32 y permite consultar el clima en un gran número de localidades españolas.

El usuario debe seleccionar, bien el buscador (icono con forma de lupa) e introducir el nombre de una localidad, o bien el botón de geolocalización (situado a la derecha del buscador), que permite obtener la predicción de la localidad donde se encuentra el usuario en ese momento.

Tras introducir una cadena de texto en el buscador, aparece una lista de localidades sugeridas que comiencen por dicha sentencia, así al usuario le resulta más fácil realizar la búsqueda. Todo esto es posible gracias a la carga previa en la base de datos de unas 8124 localidades, tarea que se realiza la primera vez que se inicia la aplicación. En este caso, el tiempo de carga suele ser un poco mayor.

En cambio, si el usuario selecciona la opción de geolocalización, esta cargará directamente la predicción del lugar donde está ubicado.

Desde un principio, el objetivo de esta ventana de inicio era mostrar las predicciones meteorológicas obtenidas de la API. Aunque en un primer momento se preveía mostrar por separado los dos tipos de predicciones: horaria y diaria, finalmente se decidió representarlas en una misma actividad, ya que ambas poseen características muy similares. De esta manera, podemos reducir código redundante (actividades, métodos, etc.).

El objetivo se logra mediante el uso de un ViewPager [27], que nos permite la transición entre los distintos tipos de predicciones de manera fácil, deslizando con los dedos o seleccionando encima de la pestaña.

Figura 36. Tipos de predicciones climatológicas

Para lograr esto hemos tenido que añadir la biblioteca “android.support.v4.view.ViewPager” a nuestro proyecto.

Seguidamente, hemos creado tres fragmentos, uno para cada una de las tres predicciones que se muestran en la figura 36 y los hemos asociado al elemento ViewPager, para permitir el deslizamiento horizontal.

Para mostrar una evolución del clima a lo largo de los días nos apoyamos en el uso de gráficas.

Google no facilita la creación de estas gráficas, por lo que, tras un periodo de investigación se ha decidido utilizar MPAndroidChart [25], una librería muy fácil de usar que proporciona gran variedad de gráficas, como diagramas de barras, líneas y circulares. El resultado puede verse en la figura 37.

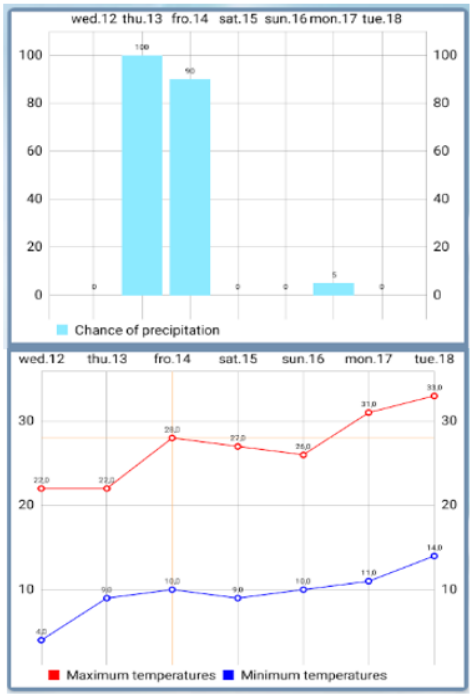

Figura 37. Gráficas para las predicciones diarias

Cabe destacar que la ventana ofrece retroalimentación de manera que el usuario se pueda dar cuenta de lo qué está haciendo mal y cómo puede solucionarlo. Por ejemplo, al pulsar el botón de favoritos sin haber seleccionado ninguna localidad previamente, se genera un mensaje avisando de que debe seleccionarla, o tras realizar una consulta de una localidad inexistente, la aplicación avisa de que no se encuentra en la base de datos. En la figura 38 se muestran algunos de estos mensajes informativos.

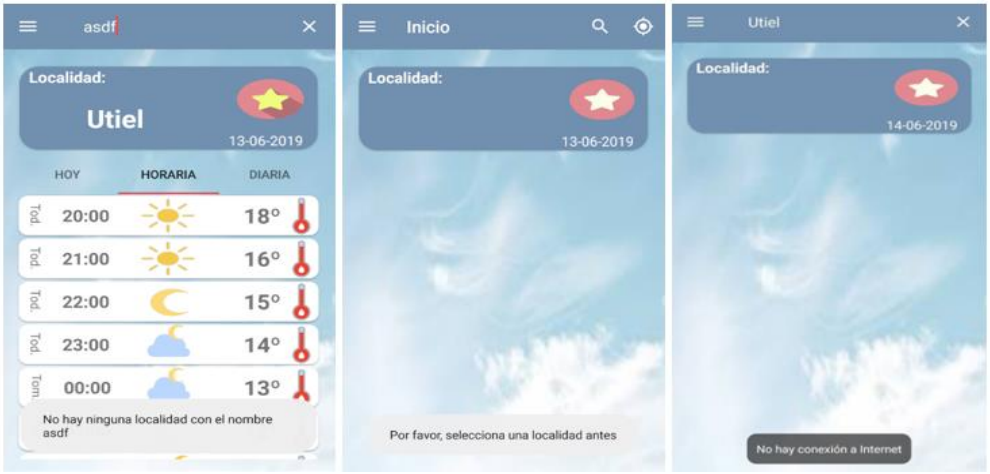

Figura 38. Ejemplos de mensajes del sistema

7.1.3. Ventana de favoritos

Para esta funcionalidad de la aplicación nos hemos ayudado de una lista vertical, que muestra las localidades favoritas guardadas tras haber seleccionado el botón de añadir a favoritos de la ventana de inicio.

Figura 39. Ventana favoritos

Si nos fijamos en la figura 39, cada elemento de la lista contiene un campo de texto con el nombre de la localidad y un botón que lo elimina de la lista de favoritos y de la base de datos para que no se muestre la próxima vez que se consulten.

Para el almacenamiento de los favoritos nos hemos ayudado de las preferencias de Android, que proporcionan una forma de persistir información y recuperarla cuando sea necesaria.

Cada vez que el usuario pulsa el botón de añadir a favorito, se crea un nuevo registro en el fichero XML con su nombre y el valor “true”, como podemos observar en el fragmento 1. Y cuando se elimina de la lista, este registro toma el valor de “false”.

```
<map>
    <boolean name="Madrid" value="true" />
    <boolean name="Ciudad Real" value="true" />
    <boolean name="Barcelona" value="true" />
    <boolean name="Utiel" value="true" />
    <boolean name="Cádiz" value="true" />
    <boolean name="València" value="true" />
</map>
```

Fragmento 1. Fichero XML con las localidades favoritas almacenadas

Para obtener las localidades favoritas guardadas, utilizamos el método getAllFavourites() de la clase PreferencesHelper, que devuelve un conjunto de pares <clave, valor> como los de la imagen anterior, donde la clave es el lugar y el valor indica si es o no favorito.

7.1.4. Ventana de mapas

Para implementar la visualización de los mapas significativos, imágenes de radar y mapas de predicción se ha utilizado un menú con opciones desplegables que permiten seleccionar una opción cada vez.

Tanto las imágenes como su correspondiente descripción las hemos obtenido de la propia API de AEMET. Aunque la manera de obtener dicha información lo veremos más en adelante, en el apartado 7.2. En la figura 40 se muestran ejemplos de los diferentes mapas e imágenes de radar a los que podemos acceder.

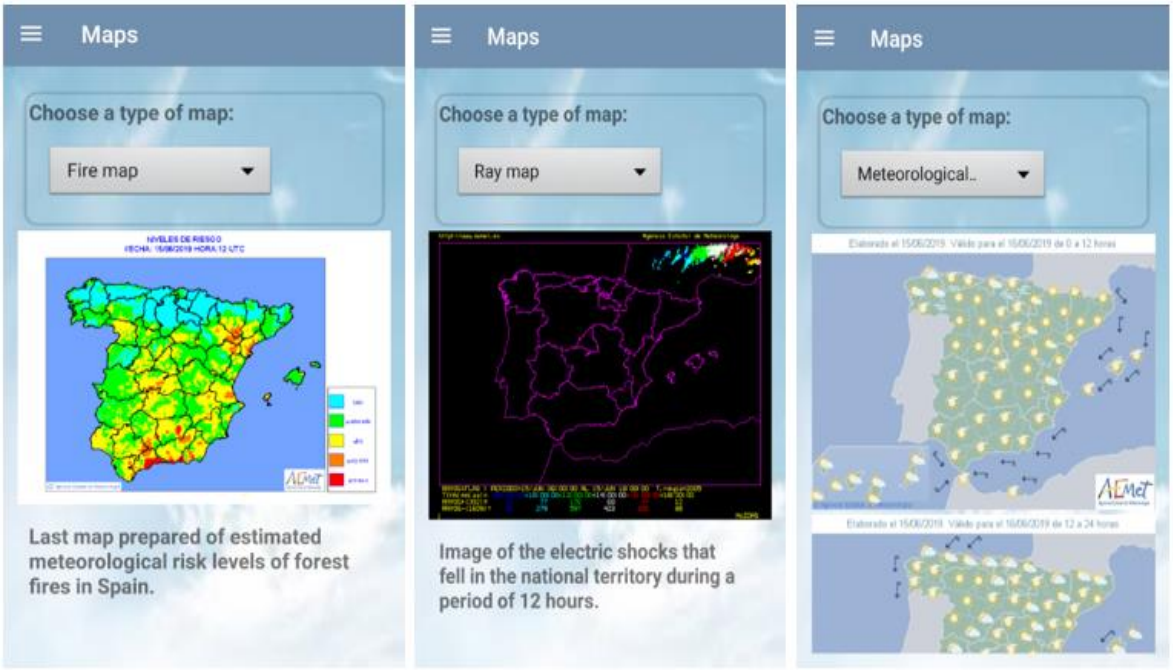

Figura 40. Ejemplos de mapas e imágenes de radar

7.1.5. Ventana de notificaciones

Para lograr que la aplicación pueda notificar al usuario del tiempo climatológico de una localidad, se ha implementado en una sección denominada Notificaciones una ventana que permite configurar una notificación diaria, idéntica a la que vemos en la figura 41.

Figura 41. Ventana de notificaciones

Está compuesta por un botón, en la parte superior derecha, que permite activar o desactivar la notificación y, justo debajo, un conjunto de CheckBox[11] de selección única que contiene opciones acerca de la hora de envío del mensaje y una lista desplegable las localidades favoritas.

Al activar la notificación se muestra un mensaje informando que la alarma ha sido configurada.

En la parte superior aparece un mensaje que muestra la notificación activa en ese momento. Al desactivarla, este mensaje desaparece.

Para su creación tenemos que importar la API NotificationCompat, proporcionada por la biblioteca de soporte de Android.

En primer lugar, necesitamos construir el cuerpo del mensaje creando una instancia de NotificationCompat.Builder y darle valor a una serie de parámetros.

El contenido de la notificación, representada en la figura 42, viene configurado a partir de los siguientes parámetros:

1. Una pequeña imagen, establecida por el método setSmallIcon(), que hace referencia al icono de la aplicación.
2. El nombre de la aplicación, el cual que aparece por defecto en el mensaje.
3. La localidad notificada, fijada por setContentTitle().
4. Descripción breve del tiempo y temperatura actual, a través del método setContentText().

Figura 42. Contenido de la notificación

[11] CheckBox: tipo de botón de doble estado, seleccionado o no seleccionado.

5. Una imagen representativa del estado del cielo en la localidad notificada, establecida con setLargeIcon().

Por último, para mostrar la notificación por pantalla llamamos al método notify(), del mismo modo que en el fragmento de código 2.

```
notificationManager.notify(0, mNotifyBuilder.build());
```

Fragmento 2. Código necesario para mostrar la información

7.2. Comunicación con la API REST de AEMET

La Agencia Estatal de Meteorología proporciona un servicio Open Data, al cual tenemos acceso a través del protocolo HTTP.

Para obtener la información meteorológica en la aplicación necesitamos establecer una conexión con su API. Por ello, durante su implementación nos hemos ayudado de la biblioteca OkHttp, comentada en el apartado 6.7 y, además, hemos utilizado tareas asíncronas para conseguir dicha información y evitar bloqueos en la aplicación.

7.2.1. Tarea asíncrona o AsyncTask

Las tareas asíncronas nos permiten llevar a cabo operaciones en segundo plano y evitar que el hilo principal de la aplicación (donde se ejecutan todos los procesos) se bloquee mientras corra uno de estos procesos. Cuando una de estas operaciones termina, comunica su resultado al hilo principal, este aplica los cambios oportunos y continua con su ejecución.

En lo que a nuestra aplicación respecta, las predicciones climáticas, la información contenida en las notificaciones y los mapas e imágenes de radar son obtenidas gracias al uso de estos subprocesos.

7.2.2. Peticiones a la API

La API de AEMET requiere de una clave para poder acceder a los datos que proporciona. Esta clave, conocida como API Key la solicitamos al comienzo del proyecto a su página web oficial.

Una vez obtenida esta clave, podemos realizar peticiones del tipo “GET” para obtener la información.

En primer lugar, tenemos que formar la URL de la petición. Contiene la dirección web del servicio, un código que depende del tipo de información que queramos obtener y la API Key. Todo esto unido conforma la URL. Si nos fijamos en el fragmento de código número 3, podemos ver un ejemplo.

```
URL url = new URL(
"https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/46249?api_key=ey…");
```
Fragmento 3. Ejemplo de URL para una petición GET

Este caso es el utilizado para obtener la predicción diaria del municipio de código 46249. El resto de los casos son similares, cambiando la dirección web (color azul) y el código de la localidad (verde). El número de la API Key en el fragmento 3 ha sido truncado por temas de seguridad.

Una vez hemos construida la URL abrimos la conexión, seleccionamos el tipo de petición (GET) y la codificación (ISO_8859_1) del mensaje, tal y como vemos en el fragmento 4.

```
connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
connection.setRequestProperty("Content-Type", "application/json;
charset=ISO_8859_1");
connection.setDoInput(true);
connection.connect();
```
Fragmento 4. Ejemplo de conexión a la API

Para obtener información del fichero proporcionado por la API, establecemos un objeto InputStreamReader que lee de él y almacena el resultado en una variable. Seguidamente, mediante el método fromJson() que nos proporciona la biblioteca Gson, podemos deserializar este código JSON en un objeto Java. En este caso, en la clase DailyPrediction, la cual hemos modelado para poder obtener los datos que mostramos en la predicción diaria. El fragmento 5 es un ejemplo de uso del método fromJson().

```
DailyPrediction[] dp = gson.fromJson(reader, DailyPrediction[].class);
```
Fragmento 5. Deserialización del objeto JSON

El fragmento 6 representa una pequeña parte del JSON devuelto por la API a partir del código anterior. Para poder construir un objeto Java a partir de esta información, es necesario definir los atributos adecuados. Estos deben de coincidir con los valores que aparecen en JSON.

```
      "estadoCielo" : [ {
        "value" : "12",
        "descripcion" : "Poco nuboso"
      } ],
      "viento" : [ {
        "direccion" : "C",
        "velocidad" : 0
      } ],
      "temperatura" : {
        "maxima" : 25,
```

```
        "minima" : 8,
        "dato" : [ ]
      },
```

Fragmento 6. Ejemplo JSON de información meteorológica

El fragmento 7 representa la clase EstadoCielo. Contiene los atributos "value" y "description", además de los métodos (getters) necesarios para poder recuperar estos valores. Esta clase nos permitirá obtener el valor "12" y la descripción "Poco Nuboso" que aparecen en el fragmento 6, a través de los métodos getValue() y getDescription().

```
import java.io.Serializable;

public class EstadoCielo implements Serializable {

    private String value;
    private String description;

    public EstadoCielo() {

    }

    public String getValue() {
        return value;
    }

    public String getDescription() {
        return description;
    }
}
```

Fragmento 7. Clase Java 'EstadoCielo'

8. Pruebas

En este apartado se busca comprobar que la aplicación funciona correctamente y cumple con los requisitos establecidos. Veremos pruebas de carga, pruebas de aceptación realizadas a un grupo de usuarios y mediante unas guías que nos proporciona la guía de desarrollo de Android mediremos la calidad de nuestra aplicación.

8.1. Pruebas de carga

Las primeras pruebas realizadas corresponden a las mediciones del tiempo de ejecución y carga de la aplicación. Para abarcar el mayor número de situaciones posibles, hemos decidido utilizar tres dispositivos móviles con distintas capacidades y sistemas operativos. La tabla 18 incluye las características de los modelos de telefonía móvil empleados durante la prueba.

Tabla 18. Dispositivos móviles utilizados para las pruebas de carga

	LG Q6	**Google Nexus 5**	**Google Nexus S**
Versión API	28	25	22
Sistema Operativo	Oreo (Android 8.1)	Nougat (Android 7.1)	Lollipop (Android 5.1)
Procesador	Qualcomm Snapdragon 435	Qualcomm Snapdragon 800	Samsung S5PC110
RAM	3 GB	1.5 GB	512 MB
Dimensión	5.5 pulgadas (1080x2160)	4.95 pulgadas (1.920 × 1080)	4 pulgadas (480x800)

Para este estudio se han tomado medidas, mediante un cronómetro digital, de los siguientes procesos:

- En primer lugar, se ha medido el tiempo que tarda la aplicación en arrancar. Hay que decir que la primera vez que se inicia, el tiempo es algo superior a la media, ya que necesita cargar la base de datos en memoria con todas las localidades. Por ello, hemos decidido medir tanto el primer como los siguientes arranques.
- En segundo lugar, hemos medido los tiempos de carga de las predicciones de una localidad. Es necesario comprobarlo, ya que el requisito no funcional 8 establece que este tiempo no debe superar los 6 segundos.

- Finalmente, hemos medido también los tiempos de carga de los mapas e imágenes de radar.

Los resultados se muestran en la tabla 19. Hay que tener en cuenta que para realizar el cálculo de la carga de predicciones hemos decidido hacer la media aritmética del tiempo de búsqueda de tres ciudades: Barcelona, Madrid y Valencia, para que la medida sea más correcta.

Tabla 19. Mediciones del tiempo de carga en segundos

RESULTADOS (en segundos)	Primer arranque	Arranque normal	Carga de predicciones	Carga de mapas
LG Q6	2.48	2.20	2.99	2.38
Google Nexus 5	2.72	1.33	1.84	2.75
Google Nexus S	2.18	1.42	3.13	2.84

Analizando la tabla, observamos que en ningún momento se ha sobrepasado el límite de tiempo de carga mencionado en los requisitos no funcionales 7 y 8.

Además, se ha comprobado también que la aplicación se adapta correctamente al tamaño de pantalla de cualquiera de los dispositivos móviles estudiados, como puede verse en la figura 43. Esto satisface el requisito no funcional 1.

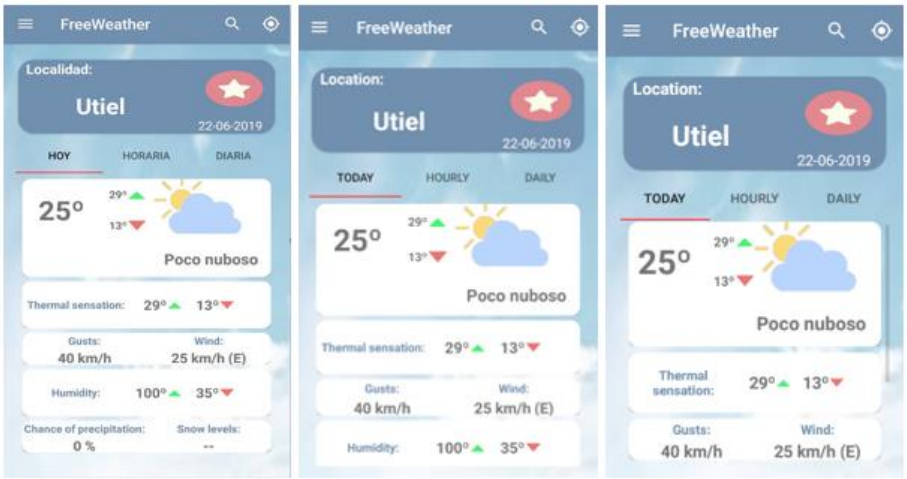

Figura 43. Ejemplo de adaptación al tamaño de pantalla en distintos dispositivos

8.2. Pruebas de usuario

En esta sección, buscamos saber si nuestra aplicación cumple funcionalmente de cara al cliente Hemos realizado un estudio, para comprobar su correcto funcionamiento, en una pequeña población compuesta por 5 personas, que abarcan las edades: 20, 22, 27, 45 y 60 años.

Para ello, hemos instalado la aplicación en sus dispositivos móviles Android. Y tras haber pasado un par de semanas, a través de un cuestionario con varias preguntas, hemos recogido información y opiniones acerca de ella. Las respuestas del cuestionario están basadas en la escala Likert [28], la cual nos permite conocer el grado de conformidad del encuestado respecto a la afirmación planteada. A continuación, vamos a ver las respuestas obtenidas.

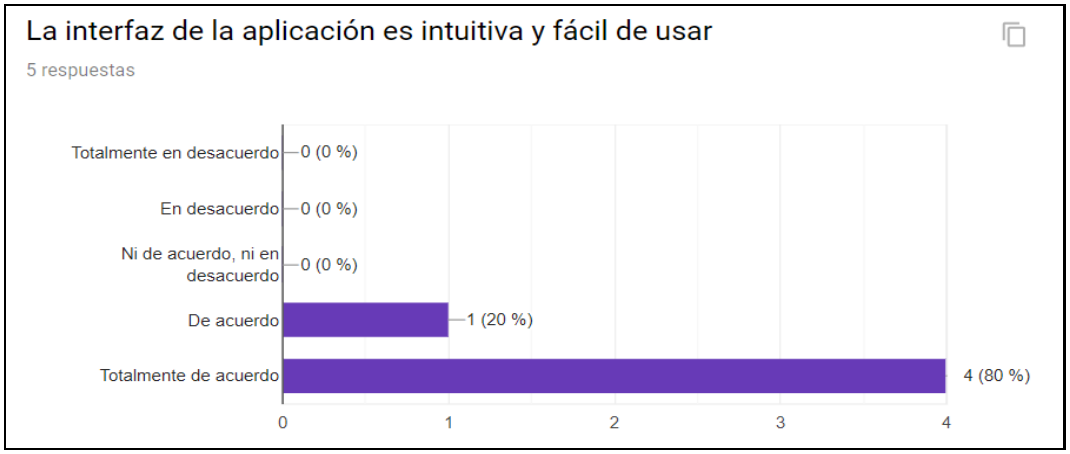

Figura 44. Primera pregunta de las pruebas de usuario

La figura 44 demuestra que los usuarios están muy de acuerdo en que les resulta fácil de entender. Algo muy importante teniendo en cuenta que uno de los objetivos de la aplicación es que sea apta para personas de cualquier edad.

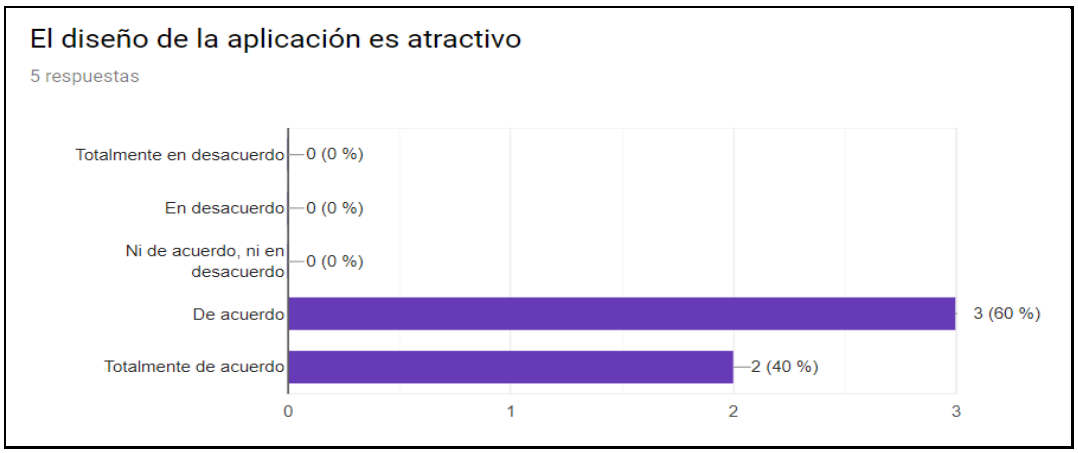

Figura 45. Segunda pregunta de las pruebas de usuario

Como podemos ver en la figura 45, según los usuarios, el diseño de la aplicación les resulta atractivo. Esto es debido a la utilización del Material Design de Google, el cual hace que las interfaces tengan un aspecto más profesional.

Tras estas dos primeras cuestiones se puede entender que los requisitos no funcionales 2 y 3, acerca de la sencillez de la aplicación y del uso de los patrones de diseño recomendados por Google, están cubiertos. Ya que todos ellos coinciden como mínimo en estar de acuerdo en ambas afirmaciones.

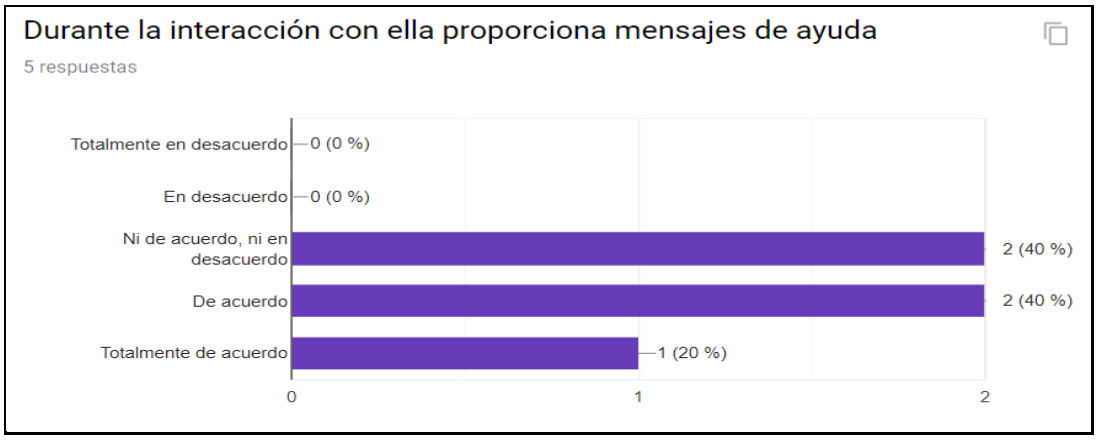

Figura 46. Tercera pregunta de las pruebas de usuario

Si observamos detenidamente la figura 46, vemos que ciertos usuarios no están de acuerdo en que la aplicación proporciona una buena retroalimentación. Posiblemente, no le hayamos dado la suficiente importancia a este requisito. Algo a tener muy en cuenta para futuras modificaciones.

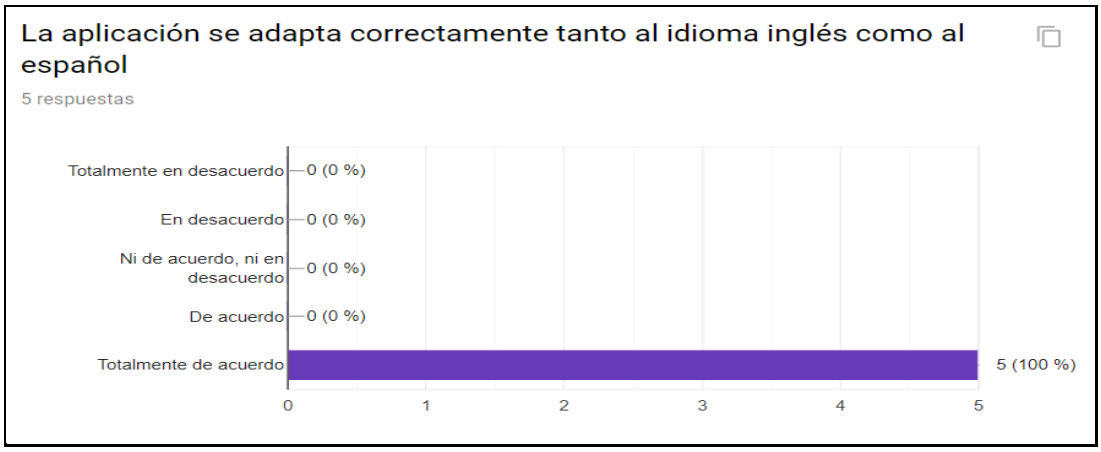

Figura 47. Cuarta pregunta de las pruebas de usuario

Según la figura 47, tenemos poco que reprochar a la capacidad de la aplicación para adaptarse a usuarios de habla inglesa o española. Todos coinciden en la misma respuesta.

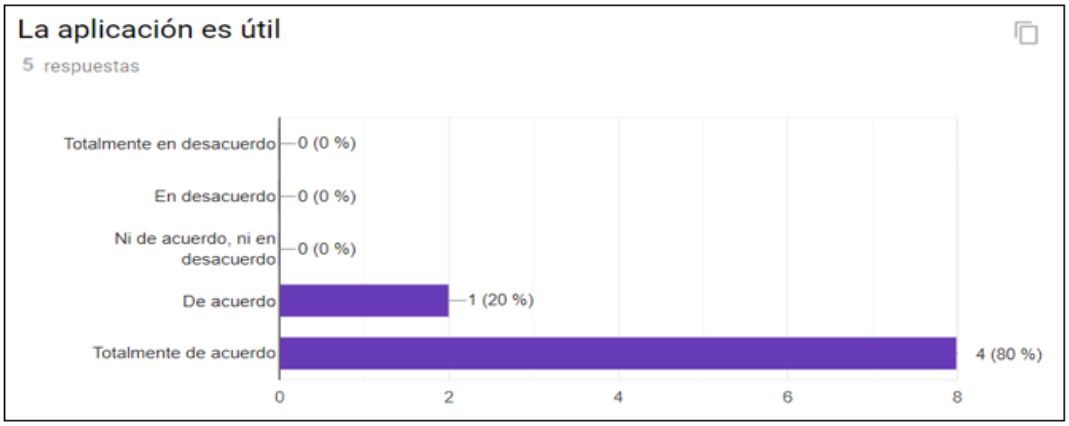

Figura 48. Quinta pregunta de las pruebas de usuario

Si observamos la figura 48, todos los usuarios consideran, en mayor o menor medida, que la aplicación es útil.

Podemos estar satisfechos al conseguir crear una aplicación de utilidad que cumple con la mayoría de los requisitos, exceptuando el requisito no funcional 4, acerca de la retroalimentación proporcionada por la aplicación, con el que ciertos usuarios se muestran algo indecisos.

8.3. Guías de calidad

Los usuarios de Android esperan aplicaciones de gran calidad. Antes de publicarla en el mercado es recomendable probarla para asegurarse que funciona correctamente. Por esta razón, Google proporciona una serie de guías o cuestionarios, que permiten medir la calidad de una aplicación.

Estas pruebas están agrupadas según el contexto de la aplicación: diseño visual e interacción con el usuario, funcionalidad, compatibilidad, rendimiento, estabilidad, seguridad, etc. Y poseen pruebas específicas para cada contexto. Además, no solo existen guías para dispositivos móviles, sino que también se pueden encontrar para otros dispositivos Android como una Tablet o una TV.

A continuación, vamos a probar algunas de las características de nuestra aplicación. Para ello, hemos utilizado tablas que agrupan las pruebas realizadas a un área de la aplicación.

En primer lugar, vamos a comenzar por el diseño visual y la interacción con el usuario.

Tabla 20. Prueba de diseño visual e interacción con el usuario

Área: Diseño visual e interacción	**Descripción**	**Test**	**Nota**
Estándar de diseño	La aplicación sigue las guías de Android Design y utiliza patrones de interfaz de usuario e iconos comunes.	CR-Todos	
Navegación	La aplicación soporta el sistema de navegación "Back Button" y no lo modifica.	CR-3	
	Todos los diálogos son ocultables usando el botón "Back button".		
	Presionando el botón "Inicio" en cualquier punto de la aplicación regresamos a la ventana de inicio.	CR-1	
Notificaciones	Notificaciones siguen las guías de Android Design: • Múltiples notificaciones se acumulan en una, mientras sea posible. • Las notificaciones son persistentes en caso de tratarse de eventos presentes, como música activa o una llamada entrante. • No contienen publicidad o contenido no relacionado con la aplicación, a menos que el usuario lo autorice.	CR-11	

En esta prueba se han realizados diversos ejercicios del tipo CR (del inglés, Core Suite). Estos ejercicios ayudan a descubrir errores en su diseño y los producidos durante la interacción con el usuario. Engloban tareas como:

- Navegación hacia todas las ventanas de la aplicación, cuadros de diálogo, etc.
- Cambios en la conectividad de red, batería o ubicación.
- Navegación hacia atrás, mediante el uso de botones.
- Cambios bruscos entre aplicaciones para comprobar su buen funcionamiento.
- Control de las notificaciones entrantes de la aplicación y la interacción con ellas.

Si nos fijamos en la tabla 20, podemos ver los resultados favorables que hemos obtenido tras realizar este examen.

En segundo lugar, trataremos el área funcional de la aplicación.

Tabla 21. Prueba de funcionalidad

Área: Funcionalidad	**Descripción**	**Test**	**Nota**
Permisos	La aplicación solicita los permisos estrictamente necesarios para realizar sus funciones.	SC-4	
	La aplicación no solicita permisos para acceder a datos confidenciales (Contactos) o servicios que puedan costar dinero (SMS).		
Localización de la instalación	La aplicación funciona correctamente cuando se instala en la memoria externa (SD).	SD-1	
Interfaz de usuario y gráficos	La aplicación es compatible con las orientaciones vertical y horizontal.	CR-5	
	Utiliza toda la pantalla en ambas orientaciones.		
	La aplicación maneja las transiciones entre orientaciones sin problemas.		
Estado de la aplicación	La aplicación no deja ningún servicio en segundo plano, como puede ser mantener una conexión Bluetooh o el GPS activo.	CR-6	
	Conserva el estado del usuario al salir de la aplicación y evita pérdidas de datos u otros cambios en la ventana.	CR-1, CR-3, CR-5	

Esta prueba combina ejercicios del tipo CR, SC (del inglés, Security) y SD (del inglés, Secure Digital) para comprobar el comportamiento funcional que se espera de la aplicación. Los ejercicios realizados afectan a la solicitud de los permisos por parte de la aplicación, a su lugar de instalación, al manejo de cambios de orientación y a su estado. Si observamos la tabla 21, vemos que tanto los permisos como el estado son correctos. Sin embargo, no ha superado la prueba de interfaz de usuario y gráficos. Hay que decir, que esto era de suponer, ya que desde un principio no se contaba con que la aplicación fuera compatible con la orientación horizontal. Esto es debido a que, la información mostrada no ocupa tanto espacio como para necesitar ampliar horizontalmente. Este es un aspecto para tener en cuenta en aplicaciones de dispositivos de mayor tamaño como una tableta.

En tercer lugar, nos centraremos en la compatibilidad, el rendimiento y la escalabilidad de la aplicación.

Tabla 22. Prueba de compatibilidad, rendimiento y estabilidad

Área: Compatibilidad, rendimiento y escalabilidad	**Descripción**	**Test**	**Nota**
Estabilidad	La aplicación no se bloquea, ni fuerza el cierre, ni se congela o funciona inadecuadamente.	CR-Todos, HA-1	
Actuación	La aplicación se carga rápidamente o proporciona un indicador del estado hasta que termina el proceso.	CR-Todos	
Calidad visual	La aplicación muestra texto, gráficos, imágenes y otros elementos sin distorsión o pixelados.	CR-Todos	
	Muestra texto y bloques de texto de forma adecuada: • No hay palabras incorrectas en botones u otros elementos gráficos. • No hay frases o palabras truncadas. • Hay espacio suficiente entre elementos.	CR-Todos	

En esta área se combinan pruebas de tipo CR y HA (del inglés, Hardware Acceleration), las cuales ofrecen una serie de cuestiones enfocadas a la confiabilidad y la eficiencia de la aplicación. Hemos comprobado, por ejemplo:

- Que la aplicación no se bloquea, cierra o congela mientras se está utilizando.
- Que muestra el estado de un proceso que tarda en cargarse.
- La correcta visualización de los elementos que conforman la interfaz.

Tras realizar estas cuestiones, y según la tabla 22, concluimos que nuestra aplicación cumple con los requisitos vistos, solicitando los permisos únicamente necesarios(en nuestro caso, el permiso para obtener la ubicación del dispositivo), proporcionando indicadores de progreso en procesos costosos y controlando que los elementos se visualizan correctamente.

Por último, queda decir que existen más tipos de pruebas sobre la seguridad de la información en la aplicación, media (vídeo, audio…) y Google Play. Sin embargo, no hemos creído conveniente realizarlas, ya que nuestra aplicación no trabaja con información privada de los usuarios, ni reproduce vídeos o música, y por el momento no nos hemos planteado publicar la aplicación en Google Play debido al coste que requiere.

9. Conclusiones

El movimiento Open Data nos provee gran cantidad de información que no es fácil de acceder para ciertas personas, pues muchas provienen en formatos difíciles de interpretar (JSON, XML, etc.). La gran tendencia por parte de las empresas a exponer datos de este tipo públicamente hace necesario el desarrollo de programas que faciliten el acceso a esta información, para que cualquier persona pueda disfrutar de ella. Por esta razón, en este proyecto se ha desarrollado una aplicación que facilita el acceso a las previsiones meteorológicas de AEMET.

Existe un gran número de aplicaciones en el mercado que tratan este tipo de "datos abiertos" y muestran la información de manera entendible para las personas, pero suelen presentar una estructura caótica, donde es difícil aplicar cambios sin afectar al resto de las partes. Sin embargo, nuestra aplicación está construida en torno a un modelo de tres capas, lo cual aporta cierto grado de flexibilidad y adaptabilidad. De este modo, si necesitáramos realizar algún cambio en cualquiera de las capas (presentación, lógica de negocio y acceso a datos) nos resultaría más sencillo. Para ello, hemos utilizado el patrón modelo-vista-presentador, el cual es una derivación del conocido patrón arquitectónico modelo-vista-controlador (MVC), solo que más encaminado al desarrollo de aplicaciones en dispositivos móviles.

Para construir una aplicación útil y con un toque de madurez hemos tenido que realizar un estudio del mercado de aplicaciones meteorológicas y ampliar nuestro conocimiento a través de la documentación que nos proporciona la API de Android, acerca del empleo de buenas prácticas de programación y el uso del Material Design de Google, el cual era otro de los objetivos establecidos. Esta guía de diseño nos permite obtener una navegación fluida entre las ventanas de la aplicación y una correcta visualización de la información.

Como hemos podido ver en las pruebas de usuario, es apta, en mayor o menor medida, para usuarios de distintas edades, debido a una interfaz sencilla a la vez que atractiva.

En cuanto a los demás objetivos, la aplicación permite obtener predicciones meteorológicas en localidades españolas de manera horaria y diaria, obtener mapas e imágenes de radar, gestionar las predicciones de los lugares favoritos y configurar una notificación con información climática de una localidad en particular. La intención es la de ampliar las funcionalidades para darle mayor magnitud a la aplicación, algo que trataremos en el apartado 9.2, acerca del trabajo futuro.

Hay que destacar que es una aplicación estable, que no se cuelga ni bloquea mientras el usuario interactúa con ella.

Además, por el momento hemos conseguido adaptar la aplicación para usuarios de habla inglesa o española. Del mismo modo, también sería sencilla su adaptación a cualquier otro idioma que creamos conveniente en un futuro.

En general, se han cumplido con la mayoría de los objetivos fijados, quedando pendiente su lanzamiento al mercado a través de Google Play (otro de los objetivos establecidos), para que la gente pueda beneficiarse gratuitamente de ella.

De este trabajo, he aprendido a autoorganizarme, siendo capaz de gestionar el tiempo y de solventar cualquier problema que se me presentaba durante su desarrollo. Además, me ha servido para valorar la cantidad de trabajo y esfuerzo que requiere un proyecto de este tipo.

9.1. Relación del trabajo desarrollado con los estudios cursados

En la elaboración de este proyecto se han aplicado muchos conocimientos adquiridos durante el grado en Ingeniería Informática. Principalmente, asignaturas como Ingeniería del Software y Análisis y Especificación de Requisitos, nos han ayudado a realizar la parte de especificación de los requisitos del sistema, empleando las técnicas de modelado vistas en estas asignaturas.

La elección de Android Studio como herramienta de desarrollo se debe mayormente a la preferencia por el lenguaje Java, el cual ha sido, desde el inicio del grado, el lenguaje base que me ha introducido en el mundo de la programación. Asignaturas como: Introducción a la Programación, Programación, Estructuras de Datos y Algoritmos o Interfaces Persona Computador me proporcionaron la formación necesaria para comenzar con esta andadura.

Finalmente, hay que destacar ciertas competencias transversales que hemos puesto práctica en este trabajo. Principalmente, tras un esfuerzo de meses de trabajo para conseguir los objetivos establecidos, la competencia de diseño y proyecto es la que más presente ha estado. Sin embargo, también hemos necesitado del análisis y resolución de los problemas, para aquellos obstáculos que nos hemos ido encontrando durante el desarrollo de la aplicación. Además, la competencia transversal de planificación y gestión del tiempo nos ha ayudado a tener controlado el proyecto y llegar a las fechas de entrega. Por último, y no menos importante, en este trabajo se han utilizado herramientas y conceptos desconocidos, por lo que otra competencia que cabe destacar es la del aprendizaje permanente.

9.2. Trabajo futuro

Al tratarse de una primera aproximación de la aplicación existe gran cantidad de trabajo posible o con aire de mejora.

En primer lugar, debido a que los tiempos de entrega se acercaban no hemos tenido tiempo para mejorar visualmente las ventanas de favoritos y notificaciones, algo que hubiese dado un toque más profesional. Del mismo modo, también hemos tenido que dejar fuera una opción extra del menú principal que hubiera proporcionado ayuda, a modo de imágenes, para la utilización de la aplicación.

En segundo lugar, se podría ampliar la funcionalidad de la aplicación añadiendo nuevas secciones con la información que nos proporciona la API de AEMET, como predicciones marítimas y de montaña, avisos meteorológicos o valores climatológicos de las estaciones. Además, podríamos implementar un mapa interactivo de España que permita obtener las predicciones seleccionando comunidades o provincias, algo novedoso pero que se ha decidido no realizar por el momento, ya que nos llevaría mucho tiempo desarrollarlo.

Tampoco estaría de más adaptar la aplicación a otros idiomas, aparte del inglés y español.

En tercer lugar, deberíamos disminuir más los tiempos de carga de información, imágenes y mapas por pantalla para mejorar la experiencia del usuario y evitar un posible descontento con la aplicación por su parte.

Dentro del apartado de mejoras, en vistas a que uno de los objetivos es el de diseñar una aplicación atractiva a la vez que intuitiva, podríamos emplear más Material Design de Google y estudiar más opciones para mejorar su interfaz. Para ello, tendríamos que investigar y obtener más conocimientos acerca del desarrollo de interfaces móviles.

Bibliografía

[1] «Budapest Open Access Initiative,» 2012. [En línea]. Disponible: https://www.budapestopenaccessinitiative.org/. [Último acceso: 20 febrero 2019].

[2] R. Muñoz, «El número de líneas móviles supera por primera vez a la población mundial,» 27 febrero 2018. [En línea]. Disponible: https://elpais.com/tecnologia/2018/02/27/actualidad/1519725291_071783.html. [Último acceso: 24 febrero 2019].

[3] AEMET, «Agencia Estatal de Meteorología - AEMET. Gobierno de España,» 1980. [En línea]. Disponible: wwww.aemet.es. [Último acceso: 1 marzo 2019].

[4] Anónimo, «Blog Historia de la Informática,» 14 diciembre 2012. [En línea]. Disponible: https://histinf.blogs.upv.es/2012/12/14/android/. [Último acceso: 4 marzo 2019].

[5] Google, «Arquitectura de la plataforma,» 2019. [En línea]. Disponible: https://developer.android.com/guide/platform. [Último acceso: 24 marzo 2019].

[6] «El Tiempo,» [En línea]. Disponible: https://www.eltiempo.es. [Último acceso: 29 marzo 2019].

[7] «AccuWeather,» 1962. [En línea]. Disponible: https://weather.com/. [Último acceso: 28 marzo 2019].

[8] IBM, «The Weather Channel,» [En línea]. Disponible: https://weather.com/. [Último acceso: 28 marzo 2019].

[9] P. Mulder, «MoSCoW Method: for setting Requiremets by order of Priority,» ToolsHero, 2013. [En línea]. Disponible: https://www.toolshero.com/project-management/moscow-method/. [Último acceso: 30 marzo 2019].

[10] R. M. Agut, «Especificación de Requisitos Software según el estándar IEEE 830,» 2001. [En línea]. Disponible:

http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/ERS_IEEE830.pdf. [Último acceso: 1 abril 2019].

[11] «PMOinformatica.com,» 6 mayo 2015. [En línea]. Disponible: http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html. [Último acceso: 10 abril 2019].

[12] G. Kotonya y I. Sommerville, «Requirements engineering : processes and techniques,» 1997. [En línea]. [Último acceso: 15 abril 2019].

[13] A. v. Lamsweerde, «Requirements engineering : from system goals to UML models to software specifications,» 2009. [En línea]. [Último acceso: 19 abril 2019].

[14] R. C. Martin, «Clean Architecture: A Craftsman's Guide to Software Structure and Design.,» Prentice Hall, 2017. [En línea]. [Último acceso: 26 abril 2019].

[15] C. P. Martinez, «DevExperto,» 9 febrero 2016. [En línea]. Disponible: https://devexperto.com/arquitectura-del-software/. [Último acceso: 10 mayo 10].

[16] Develapss, «Modelo Vista Presentador (MVP) en Android,» 20 julio 2016. [En línea]. Disponible: http://www.develapps.com/es/noticias/modelo-vista-presentador-mvp-en-android. [Último acceso: 19 mayo 2019].

[17] Google, «Android Studio,» Android Developers, 2019. [En línea]. Disponible: https://developer.android.com/studio. [Último acceso: 28 mayo 2019].

[18] «SQLite Home Page,» [En línea]. Disponible: https://www.sqlite.org/index.html. [Último acceso: 3 junio 2019].

[19] «GitHub,» [En línea]. Disponible: https://github.com. [Último acceso: 3 junio 2019].

[20] «BoUml,» [En línea]. Disponible: https://www.bouml.fr/. [Último acceso: 3 junio 2019].

[21] «GNU Image Manipulation Program,» [En línea]. Disponible: https://www.gimp.org/. [Último acceso: 4 junio 2019].

[22] «Balsamiq,» [En línea]. Disponible: https://balsamiq.com/. [Último acceso: 4 junio 2019].

[23] «OkHttp,» [En línea]. Disponible: https://square.github.io/okhttp/. [Último acceso: 4 junio 2019].

[24] «Gson,» [En línea]. Disponible: https://github.com/google/gson. [Último acceso: 4 junio 2019].

[25] P. Jahoda, «MPAndroidChart,» 2014. [En línea]. Available: https://github.com/PhilJay/MPAndroidChart#creators. [Último acceso: 4 junio 2019].

[26] Google, «Navigation drawer,» Material Design, 2019. [En línea]. Available: https://material.io/design/components/navigation-drawer.html#. [Último acceso: 8 junio 2019].

[27] Google, «ViewPager,» Android Developers, 2019. [En línea]. Disponible: https://developer.android.com/reference/android/support/v4/view/ViewPager. [Último acceso: 8 junio 2019].

[28] O. Llauradó, «La escala de Likert: qué es y cómo utilizarla,» 12 diciembre 2014. [En línea]. Disponible: https://www.netquest.com/blog/es/la-escala-de-likert-que-es-y-como-utilizarla. [Último acceso: 28 junio 2019].

Lista de acrónimos

AEMET: Agencia Estatal de Meteorología

AOT: del inglés, Ahead Of Time

API: del inglés, Application Programming Interface

ART: del inglés, Android Runtime

GSMA: del inglés, Global System for Mobile Communications

HAL: del inglés, Hardware Abstraction Layer

HTTP: del inglés, Hypertext Transfer Protocol

JSON: del inglés, JavaScript Object Notation

REST: del inglés, Representational State Transfer