CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Proyecto Final de Ciclo

ILERNA Online

Weather compare

Autor: Adrián Fraga Cortés

Tutor: Jorge Juan Delgado Durán

Fecha de entrega: 02/12/2020

Convocatoria: 1S 2020 2021

Índice de contenidos

Índice de imágenes

Índice de tablas

1. Introducción.

El principal objetivo de este proyecto fin de CFGS de Desarrollo de Aplicaciones Multiplataforma es crear una app Android que nos permita hacer una comparación de la climatología en 2 ciudades que elijamos. *Weather compare* nos permitirá observar de forma simultánea algunos datos meteorológicos como las temperaturas mínimas y máximas, el porcentaje de lluvia, nubosidad o si el día está soleado.

Esta app podrá utilizarse en dispositivos Android como pueden ser smartphones o tablets.

Imagen 1: Ejemplo de predicción meteorológica.

1.1. Motivación.

Tras un análisis previo se observó que la gran mayoría de la gente consulta la predicción meteorológica para los próximos días desde su smartphone.

Existen una multitud de aplicaciones Android para consultar la climatología, unas nos aportan más datos que otras, llegando algunas a ofrecernos información sobre las mareas, previsión en playas y otros datos de interés. Sin embargo, si lo que deseamos es realizar una comparación del tiempo en 2 localidades distintas, no disponemos de tantas opciones.

Weather compare puede ayudarnos, por ejemplo, si estamos indecisos a la hora de hacer un pequeño viaje de fin de semana, a decidir destino ya que el clima puede variar mucho en localidades no muy lejanas y elegir un destino con un clima más agradable puede ser un factor a tener en cuenta.

Otra posible utilidad de esta app es para una persona que desea emprender un negocio o actividad que se desarrolle al aire libre. Tras una comparativa previa de la climatología entre diversas localizaciones de su interés, puede decantarse por escoger una ubicación u otra para el desarrollo de su actividad.

1.2. Abstract.

Weather compare is an app developed for Android devices such as smartphones or tablets.

The main objective of this application is to allow us to compare different meteorological data such as, for example, the minimum and maximum temperatures, cloudiness, the percentage of rain probability or whether the day is sunny or not in 2 locations that we enter.

The idea of developing this app appears after observing that there are many meteorological applications, some very complete, but, if what we want is to compare the weather in 2 different cities, we have so many options.

Weather compare can be useful for us if we want to compare the weather between two locations for a short weekend trip.

Another use that we can give to this app is to choose the location in which there is the best climate to carry out an activity or business outdoor.

The application will be developed with the Android Studio IDE in Java language.

To obtain the meteorological data, the free API of AEMET will be used and the records of the comparisons will be stored in an SQLite database which will have 3 tables (user, search and comparison).

The graphical user interface will be simple and intuitive to offer a pleasant navigation to the user.

1.3. Objetivos propuestos (generales y específicos).

El principal objetivo de Weather compare es ofrecer al usuario final una app Android 100% funcional con la que poder hacer una comparación del tiempo en las 2 ubicaciones diferentes que el usuario decida.

Nos dará la posibilidad de revisar otras comparaciones que hayamos realizado con anterioridad.

2. Metodología utilizada.

El modelo de desarrollo elegido para realizar este proyecto es el modelo en cascada con retroalimentación ya que es un modelo que ofrece una fácil planificación del proyecto, es el más adecuado para realizar por personal poco cualificado, los requisitos del proyecto son estables, no se necesitan versiones intermedias durante el desarrollo y la retroalimentación nos permite volver a fases anteriores en el que caso de que se necesite hacer alguna modificación.

Por estos motivos es el modelo que más encaja para el desarrollo del presente proyecto ofreciendo la máxima garantía posible.

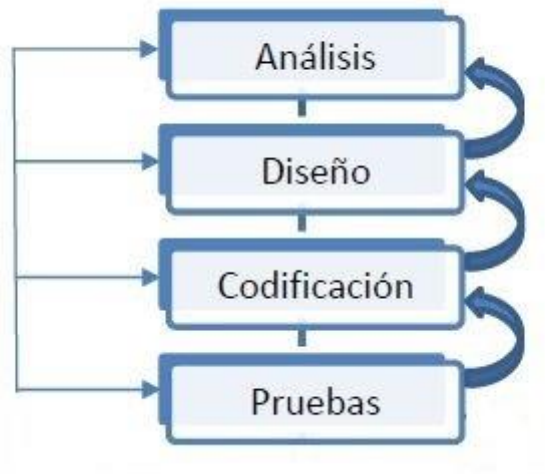

Imagen 2: Modelo en cascada con retroalimentación.

3. Tecnologías y herramientas utilizadas en el proyecto.

Para realizar el diagrama de Gantt se ha utilizado la herramienta teamgantt [4]. Con esta herramienta se hace el diagrama con el que exponemos el tiempo de dedicación estimado para cada tarea.

La app se desarrollará con el IDE Android Studio con el lenguaje de programación Java ya que es el entorno de desarrollo integrado estudiado y utilizado en el módulo de programación multimedia y dispositivos móviles y además nos ofrece un emulador para ir probando la app durante el desarrollo como la posibilidad de probarla también en un dispositivo Android real.

La base de datos para la aplicación se realizará con SQLite y contendrá las tablas necesarias para el correcto funcionamiento de la app.

Para la obtención de los datos meteorológicos se hará uso de la API gratuita que nos ofrece AEMET (Agencia Estatal de Meteorología) [17].

Durante todo el proyecto se hará un control de versiones con la herramienta cliente Git y el servidor Github.

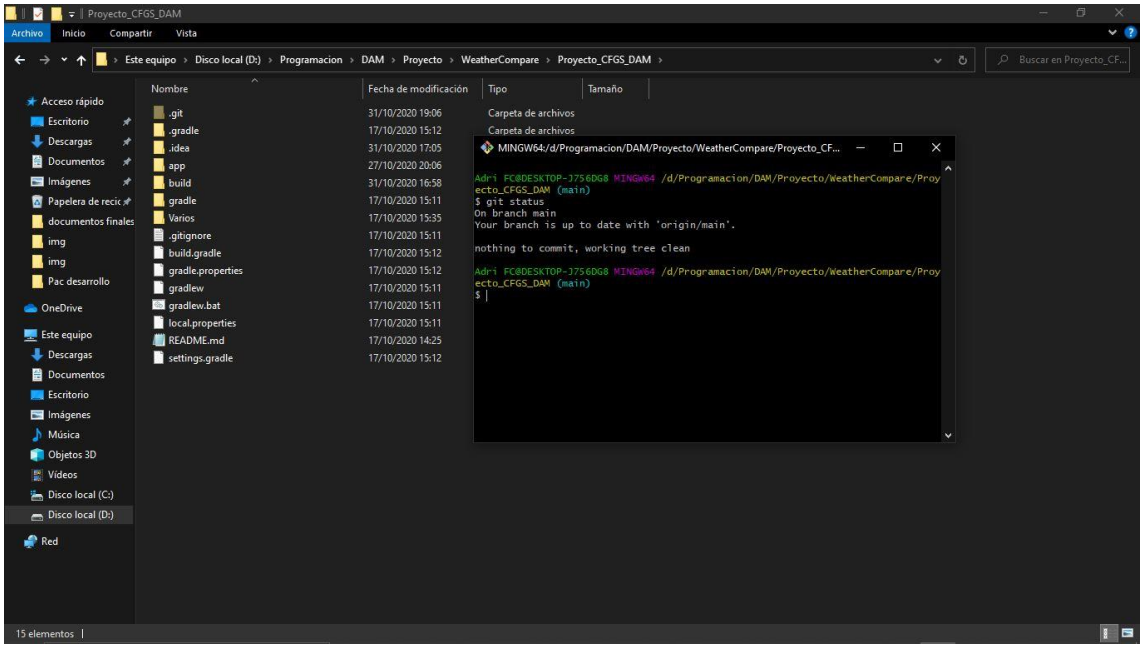

Imagen 3: Consola de Git.

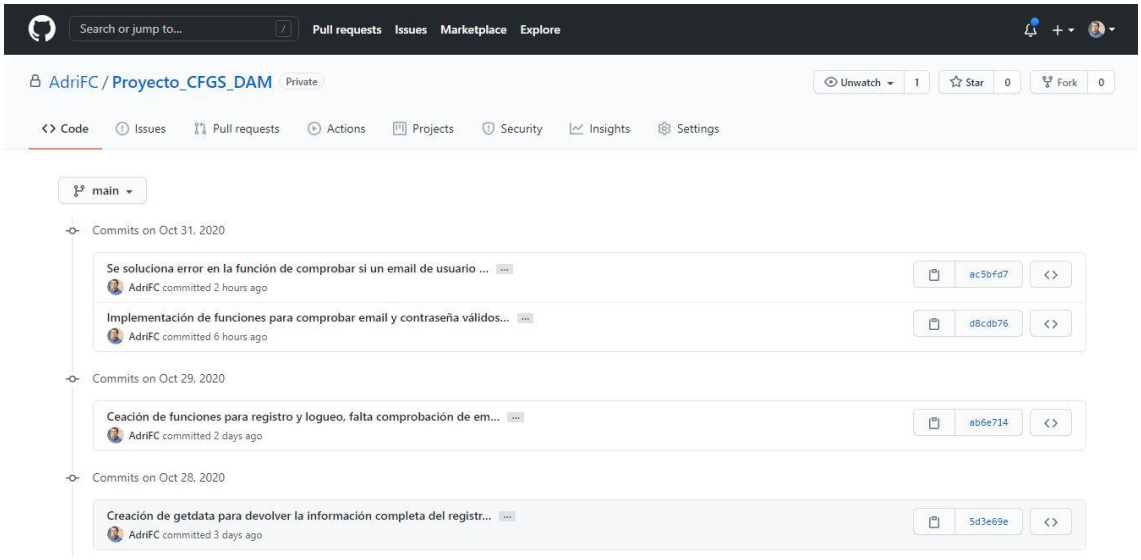

Imagen 4: Repositorio de Git en web.

4. Estimación de recursos y planificación.

En la realización de este proyecto solo hay 1 persona asignada para la ejecución de todas las tareas, la disponibilidad para trabajar en él es de 10 horas entre semana y 12 los fines de semana.

Para seguir una buena planificación durante el desarrollo, se ha optado por realizar un diagrama de Gantt el cual nos servirá de guía a la hora de distribuir las diferentes tareas desde el comienzo hasta la entrega del proyecto.

Este diagrama se irá actualizando a medida que se vayan cumpliendo los objetivos establecidos.

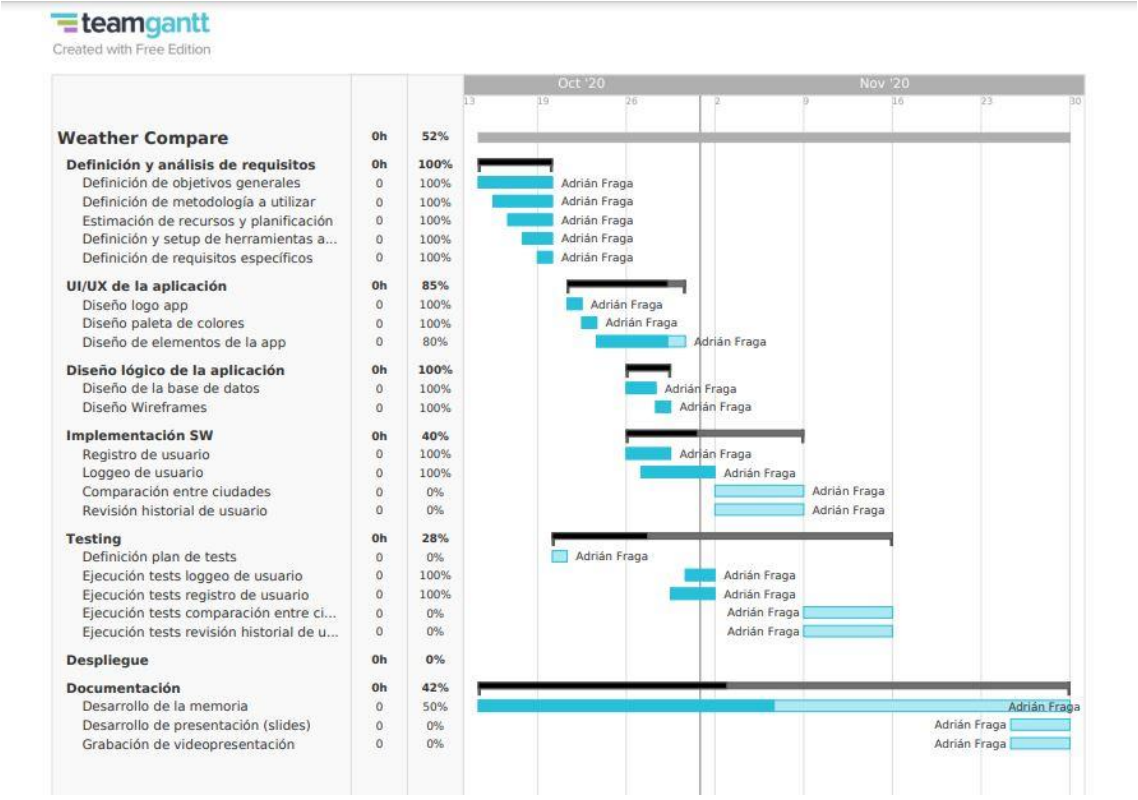

Imagen 5: Diagrama de Gantt durante el desarrollo del proyecto.

5. Desarrollo del proyecto.

En este apartado se analizan y detallan las fases en las que hemos implementado nuestro proyecto.

5.1. Análisis.

En este aparto se especifican los requisitos funcionales y no funcionales, así como el diagrama entidad-relación necesario para entender la estructura de la base de datos.

5.1.1. Definición de requisitos.

Requisitos funcionales:

1. El usuario ha de poder registrarse para usar la app, introduciendo una dirección email válida y una contraseña formada por 8 caracteres alfa numéricos y al menos 1 carácter en mayúscula.
2. Los datos de registro de usuario quedarán guardados en una base de datos SQLite interna.
3. El usuario debe poder loguearse introduciendo sus datos de registro.
4. El usuario debe poder realizar una comparativa meteorológica entre 2 ubicaciones.
5. La app debe mostrar por pantalla los datos obtenidos de la comparativa.
6. Los datos obtenidos de la comparativa deben guardarse en la base de datos.
7. La app debe facilitar al usuario un botón con el que poder revisar el historial de las 6 últimas comparativas que realizó.
8. El usuario debe de poder hacer logout.

Requisitos no funcionales:

1. La app debe poder visualizarse de forma correcta en diversos dispositivos con diferentes resoluciones y orientación de pantalla.

5.1.2. Modelo Entidad-Relación.

El diagrama Entidad-Relación nos muestra la estructura interna de la base de datos. Tenemos 2 entidades principales:

- El usuario, que cuenta con 3 atributos (Nombre, email y password).

- La búsqueda que cuenta con diferentes atributos (Id, temperatura mínima, temperatura máxima, temperatura media, sol, precipitación, velocidad media del viento y velocidad de racha del viento, provincia y localidad).

La entidad búsqueda, a su vez, tiene una relación reflexiva de comparación, que representa la comparación de 2 búsquedas de ubicaciones diferentes, y esta relación tiene un atributo fecha, que representa la fecha en la que se realiza la comparación entre dos ubicaciones.

Un ejemplo para ilustrar esta relación sería la comparación realizada el día 19/10/2020 que nos muestra los datos meteorológicos del día 21/10/2020

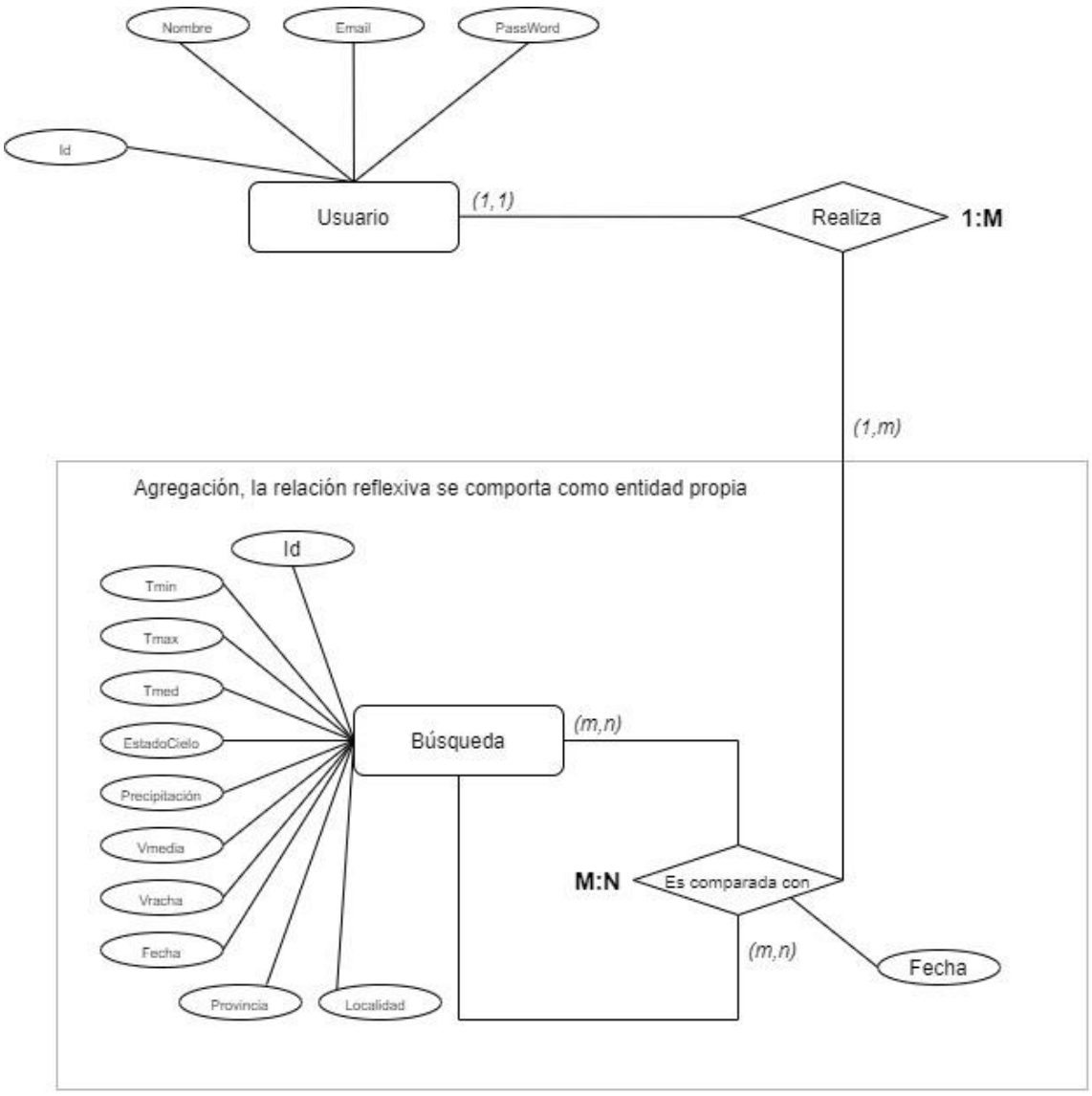

Imagen 6: Esquema Entidad-Relación.

5.2. Diseño.

En este apartado se muestran los diagramas de clases y de casos de uso resultantes del análisis previo, así como los mockups para el diseño gráfico de la app.

Estos diagramas nos ayudarán con la futura implementación de la aplicación.

5.2.1. Diagrama de casos de uso.

Con los casos de uso se pueden modelar el sistema desde el punto de vista del usuario. De esta forma obtenemos los requisitos de software.

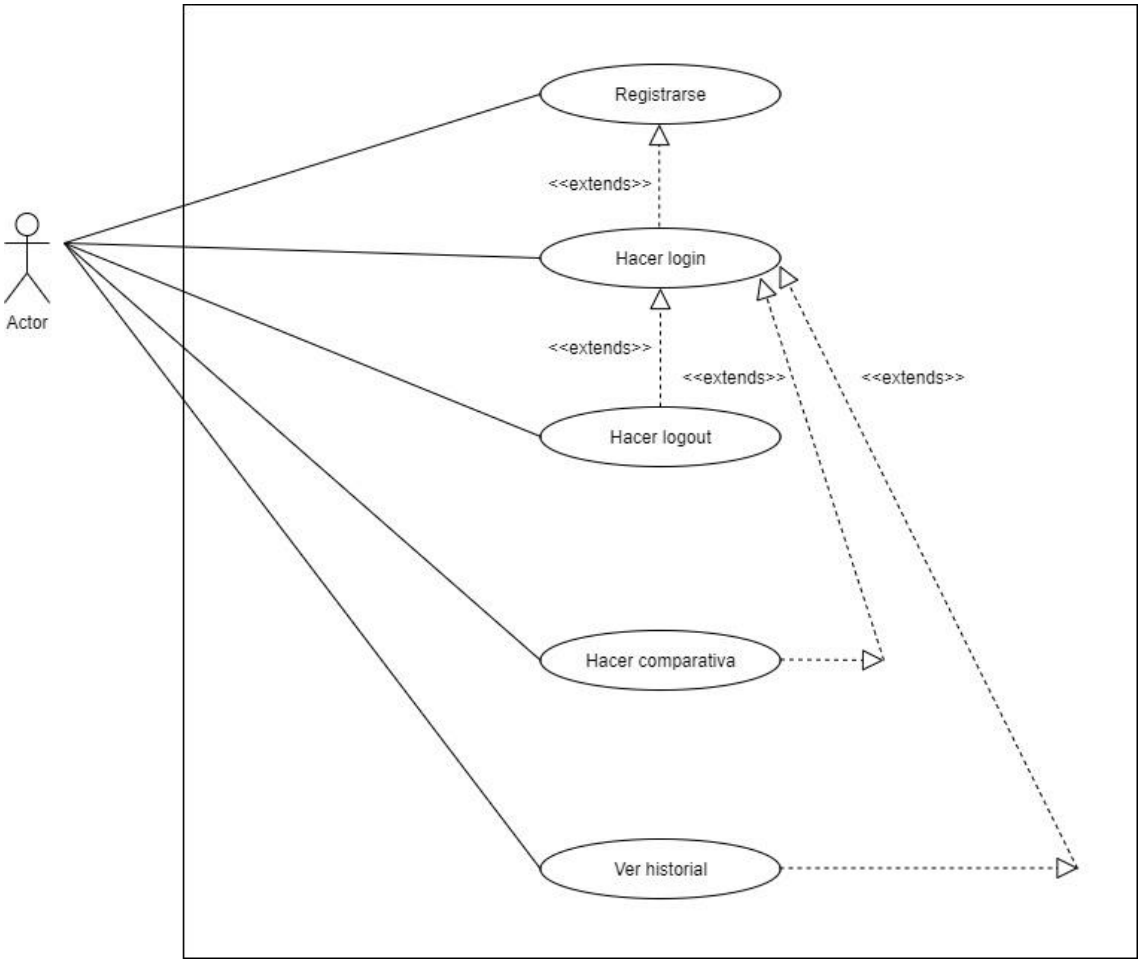

Imagen 7: Diagrama de casos de uso.

UC-1.	Registro
Descripción	El usuario se registra para usar la app
Actores implicados	El usuario final
Precondición	Haber accedido a la app
Curso normal	1. El usuario introduce su nombre 2. El usuario introduce un email válido 3. El usuario introduce un password formado por al menos 6 caracteres alfanuméricos.
Postcondición	El registro del usuario se guardará en la base de datos en caso de que los datos de registro sean correctos.
Alternativas	2. Si el email no es válido, un mensaje indicará al usuario que debe de introducir un email válido. 3. Si el password no contiene al menos 6 caracteres alfanuméricos, un mensaje indicará al usuario que debe introducir un password que si los contenga.

Tabla 1: descripción del UC-1

UC-2.	Login
Descripción	El usuario se loguea para usar la app
Actores implicados	El usuario final
Precondición	Haber realizado un registro correcto en la app
Curso normal	1. El usuario introduce su email 2. El usuario introduce su password 3. El usuario pulsa el botón de login
Postcondición	La app da acceso al usuario para realizar una comparativa meteorológica o revisar su historial de las 6 últimas comparativas realizadas.
Alternativas	1. Si el email no existe en la base de datos, un mensaje indicará al usuario que debe de introducir un email registrado. 2. Si el password no se corresponde con el registrado con el email, un mensaje informará al usuario de que la contraseña es incorrecta.

Tabla 2: descripción del UC-2

UC-3.	Logout
Descripción	El usuario cierra sesión en la app
Actores implicados	El usuario final
Precondición	Haber hecho login en la app
Curso normal	1. El usuario pulsa un botón para cerrar su sesión 2. Un mensaje informa al usuario de que cerró su sesión
Postcondición	El usuario tendrá la opción de volver a loguearse o hacer un nuevo registro de usuario
Alternativas	

Tabla 3: descripción del UC-3

UC-4.	Comparativa meteorológica
Descripción	El usuario realiza una comparación del tiempo de 2 localidades
Actores implicados	El usuario final
Precondición	Haber iniciado sesión con sus datos de usuario
Curso normal	1. El usuario introduce los datos de la primera localidad que desea comparar. 2. El usuario introduce los datos de la segunda localidad que desea comparar. 3. El usuario pulsa un botón para que se realice la comparación. 4. La app muestra por pantalla el resultado de la comparación de las 2 localidades 5. La app guarda en la base de datos la comparativa
Postcondición	La app se queda mostrando los resultados de la comparación
Alternativas	1. La app devuelve un mensaje de error si los datos introducidos para la primera localidad son incorrectos 2. La app devuelve un mensaje de error si los datos introducidos para la segunda localidad son incorrectos 4. La app muestra un mensaje de error si no consigue realizar la comparación 5. La app muestra un mensaje de error si no consigue guardar la comparación en la base de datos

Tabla 4: descripción del UC-4

UC-5.	Ver historial
Descripción	El usuario pulsa un botón para revisar su historial de búsquedas
Actores implicados	El usuario final
Precondición	Haber iniciado sesión
Curso normal	1. El usuario podrá pulsar un botón para que se muestren las últimas 6 comparativas. 2. La app busca en la base de datos las 6 últimas comparativas realizadas por el usuario 3. La app muestra por pantalla el resultado
Postcondición	La app se queda mostrando en pantalla el historial hasta que el usuario interactúe con ella
Alternativas	2. Si no dispone de ninguna comparativa previa, un mensaje lo advertirá por pantalla 2. Si hay comparativas previas, estas se mostrarán por pantalla hasta un máximo de 6

Tabla 5: descripción del UC-5

5.2.2. Diagrama de clases

Tras un análisis previo, las clases resultantes son las siguientes:

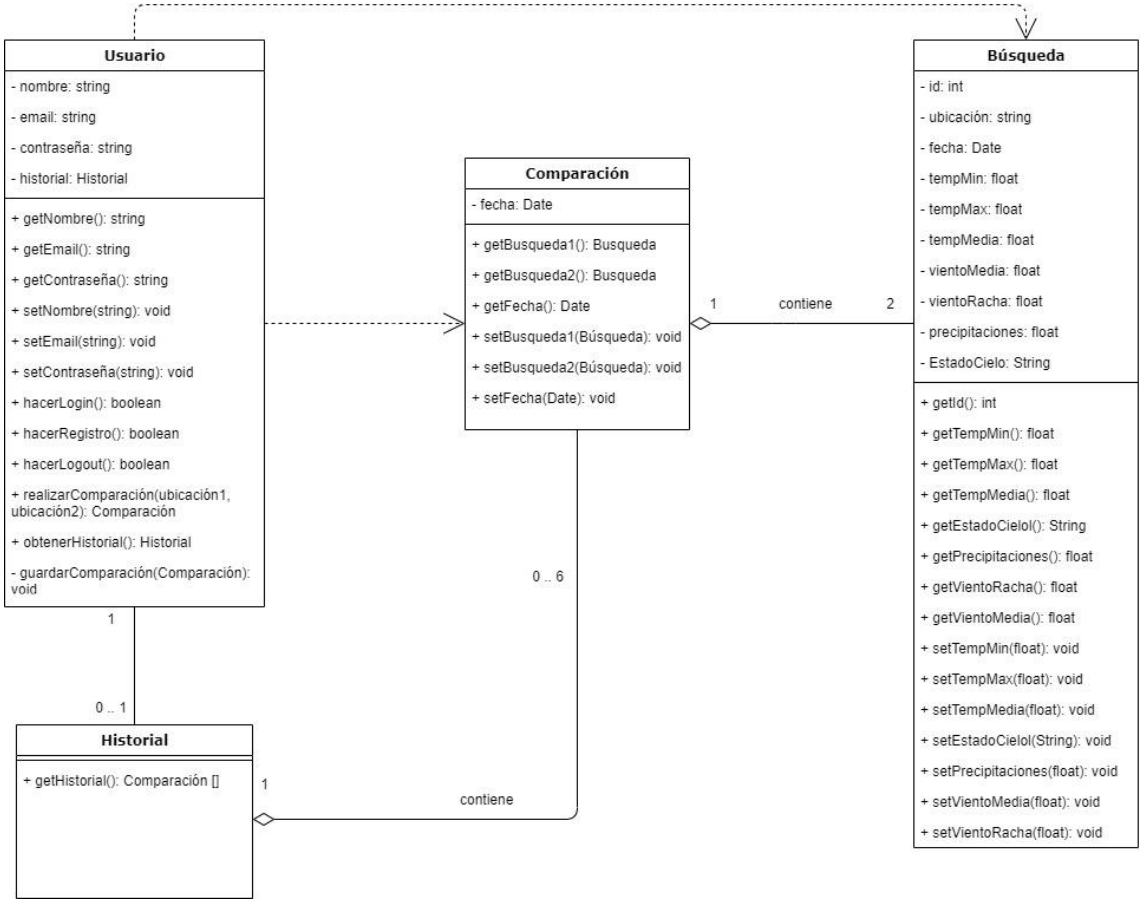

Imagen 8: Propuesta inicial de diagrama de clases.

5.2.3. Mockups de la aplicación

Tras hacer el diseño de las pantallas de usuario, el resultado es el siguiente:

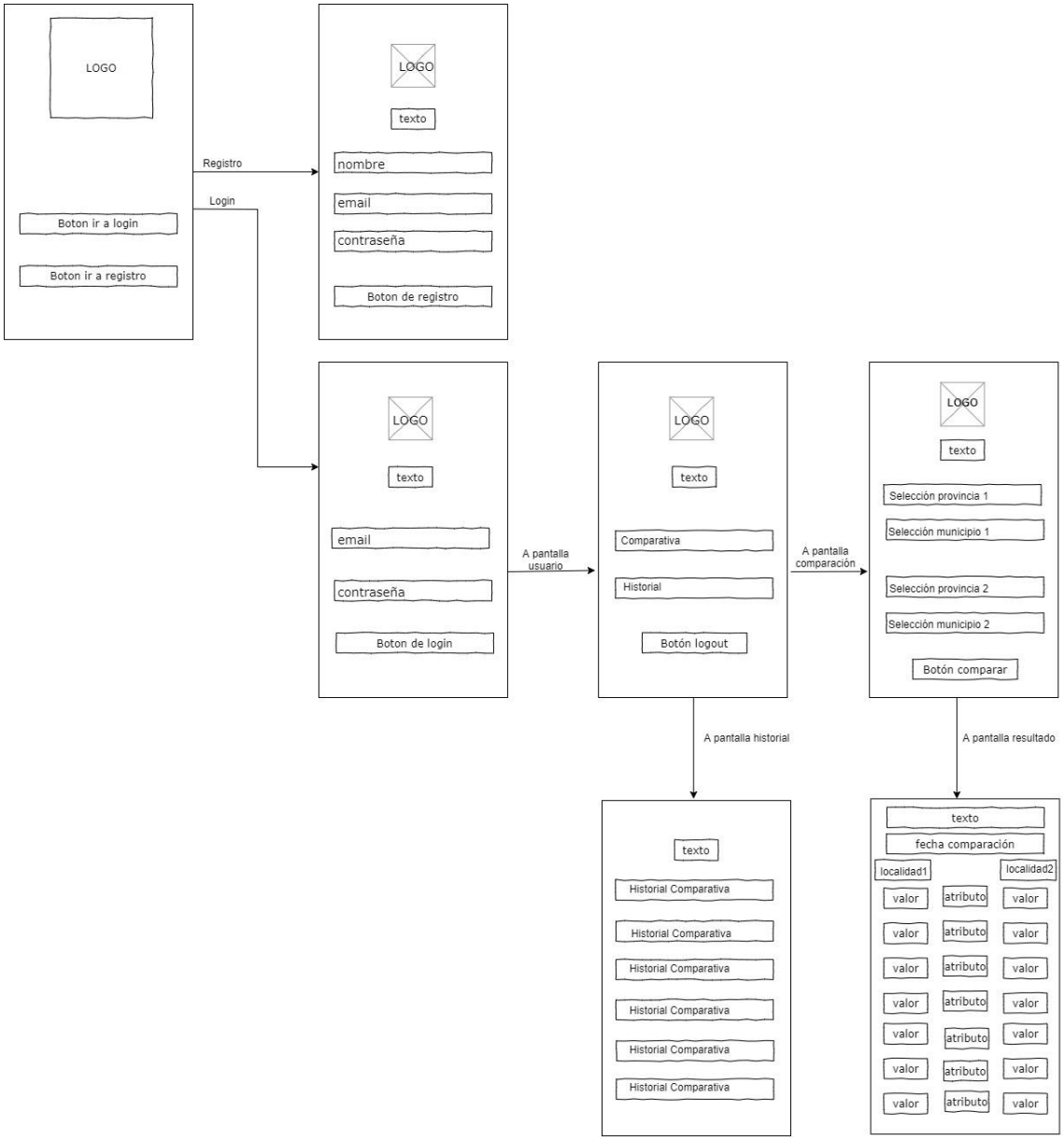

Imagen 9: Mockups.

5.3. Implementación.

En este apartado se desarrollará el proceso de implementación de la aplicación, detallando el funcionamiento de cada uno de los activities, así como los métodos y atributos pertenecientes a las clases que los implementan.

Se comienza la implementación del proyecto desde un empty activity llamado **activity_main**. Este activity es el destinado para la primera pantalla al entrar en la app. Como todo activity cuenta con su archivo .xml encargado del diseño de la interfaz gráfica y su archivo .java en el cual se desarrolló la parte lógica.

Tras crear el primer activity, se procede a la definición de los colores que se utilizaran para el desarrollo de la app en el el archivo **colors.xml.** Estos colores mantienen una temática oscura creando contraste entre ellos para facilitar la lectura de los elementos de tipo texto y los contrastes necesarios para su cómoda visualización.

Tras tener el código de colores establecido como variables, lo que nos permitirá no tener que hardcodearlos, se procede con la creación de los elementos drawable que nos permitirán tener un fondo background común para todas las pantallas, así como un estilo común para todos los botones de la aplicación, archivos **background_gradient.xml y button_design.xml**

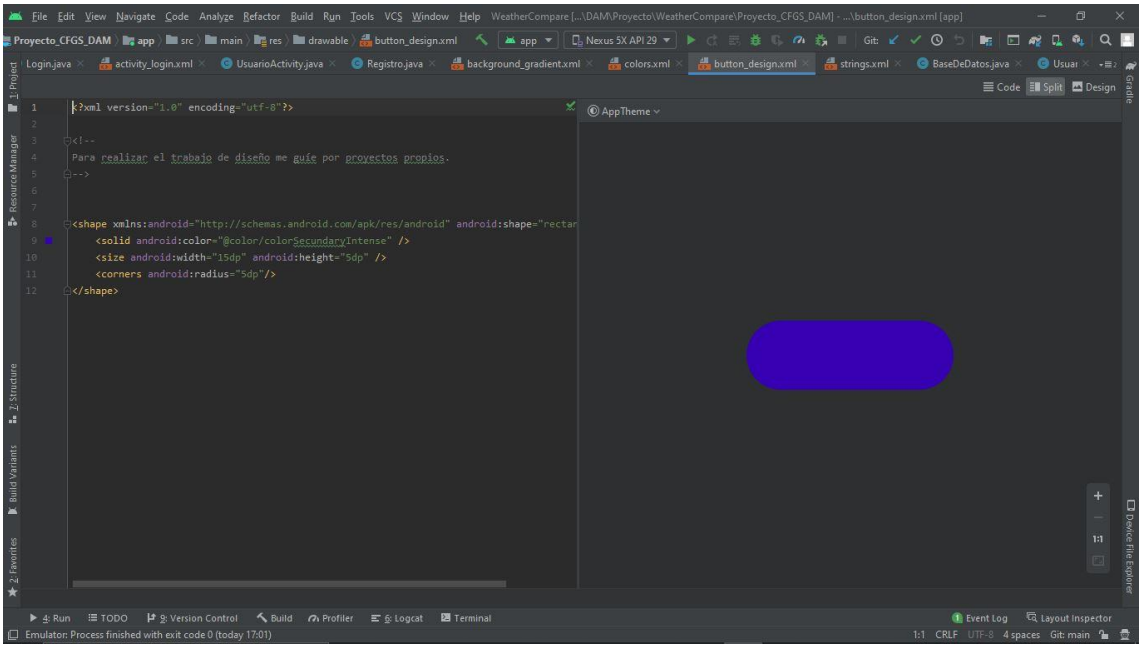

Imagen 10: Diseño establecido para todos los botones de la app.

En cuanto a los strings que usaremos durante el desarrollo de la app, se configura el archivo **strings.xml** y se van añadiendo los diferentes strings para toasts, textviews, texto de botones y todos los elementos necesarios con ayuda del **Translations Editor**, de esta forma, evitamos el uso de strings hardcodeados y tenemos mayor organización, control y reusabilidad.

Se establece como idioma predefino el **Español_ES**.

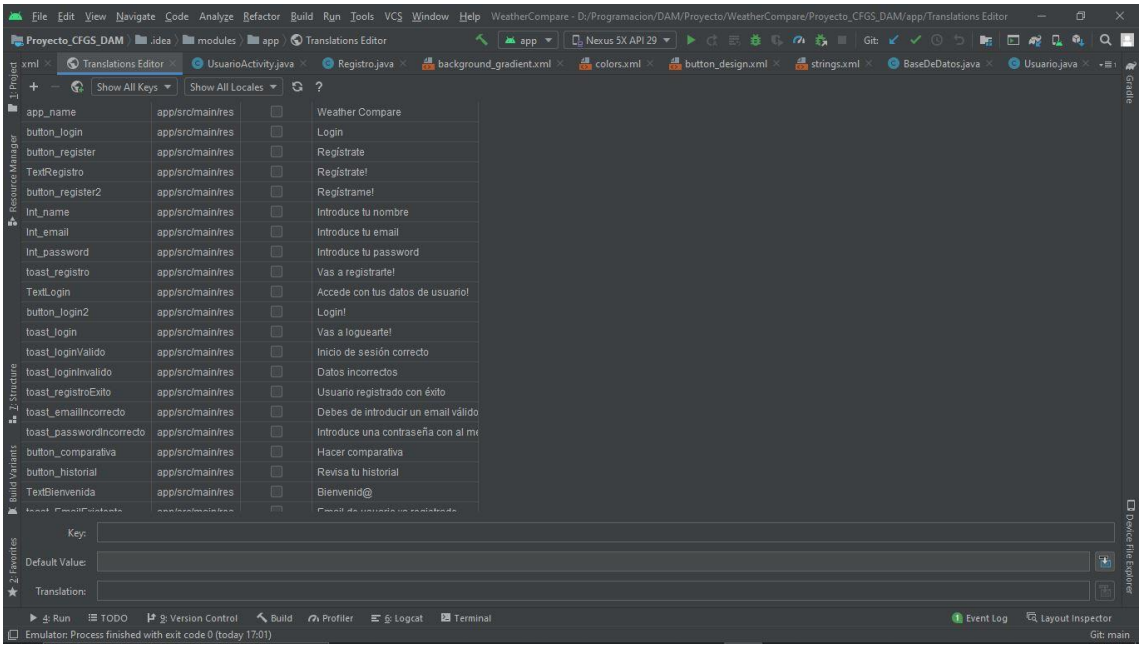

Imagen 11: Translations Editor.

Tras realizar estas primeras configuraciones, se procede a realizar el diseño de las interfaces gráficas a las que tenemos acceso desde la main_activity, es decir, las pantallas de registro de usuario y la pantalla para login de usuario, siempre realizando la versión portrait y landscape. En ambas pantallas se hace uso de los elementos de diseño ya creados con el fin de mantener coherencia en la app. En cuanto a los archivos .java de estas clases se realizan los enlaces correspondientes de sus atributos con los elementos físicos de la interfaz, como es el caso de los botones. También se programan los diferentes intents que nos permiten navegar entre los diferentes layouts de la aplicación.

A continuación, se comienza a implementar las clases definidas en el diagrama de clases. Conforme se avanza en el desarrollo, se detecta alguna necesidad adicional en cuanto a clases a implementar. Concretamente, es necesaria una clase para trabajar con la base de datos. Por tanto, se crea la **clase BaseDeDatos**, que hereda de **SQLiteOpenHelper** y cuenta con diversas funciones que nos permiten gestionar el registro y login de un usuario, además de guardar la información de las comparaciones realizadas por él. De esta forma, el diagrama de clases actualizado quedaría de la siguiente manera:

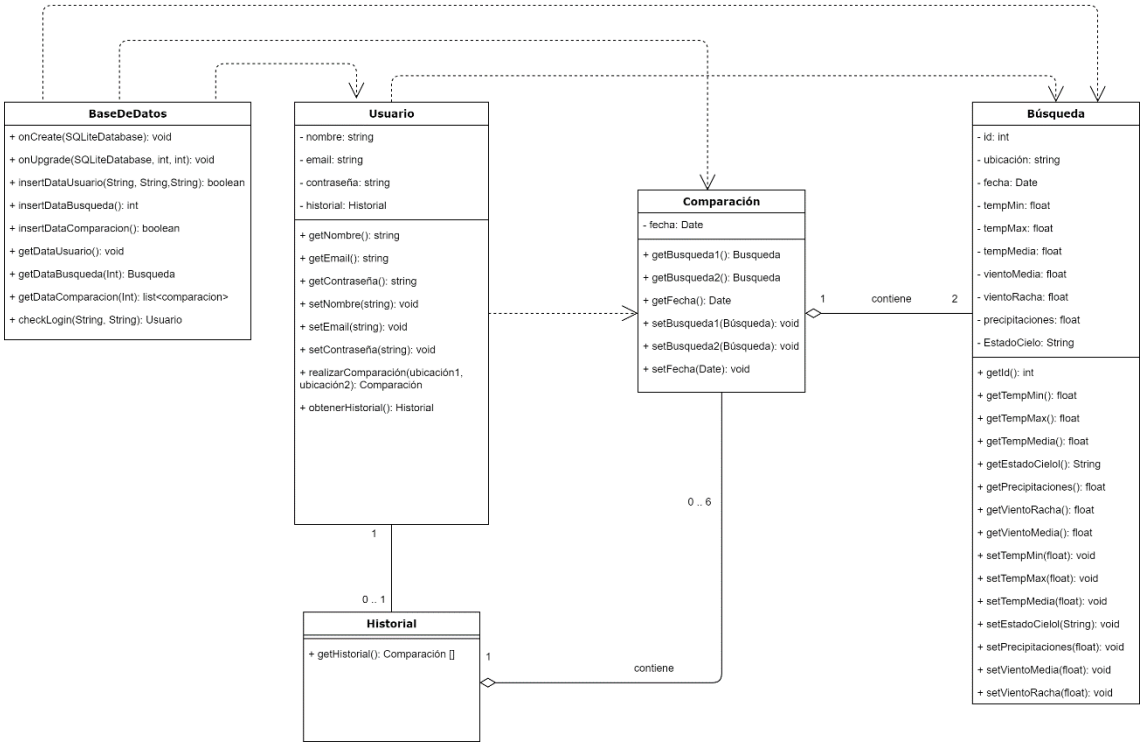

Imagen 12: Propuesta final del diagrama de clases.

Se implementa la clase **usuario** según lo descrito en el diagrama de clases. Además, esta clase implementa la interfaz **Serializable** para que sea posible intercambiar información entre activities.

A continuación, se crean la clase Búsqueda y la clase **Comparación**, que almacena las dos búsquedas junto a la fecha de realización de dicha comparación. Al ser el número de búsquedas fijo e igual a 2, se deciden guardar por separado como atributos independientes (y no usar estructuras de datos como arrays o listas).

Se implementa también la clase **Historial** y se decide guardar las comparaciones que un usuario realizó en un objeto de tipo lista. Esta es la estructura más adecuada puesto que un usuario puede tener de 0 a 6 comparaciones guardadas y nos permite guardar un número indeterminado de comparaciones.

A continuación, se implementa el activity **Registro.** Este activity tiene 3 EditText para introducir los datos de Email, contraseña y nombre, y un botón que, al ser pulsado, realiza el registro con los datos introducidos (este activity hace uso de la clase BaseDeDatos detallada anteriormente).

Para el registro, se tienen en cuenta algunas restricciones, como que el email siga un patrón válido y la contraseña contenga a los 8 caracteres alfanuméricos (mayúscula, minúscula y cifra). Para ello, se emplean la clase Patter de Java [5] [6].

Posteriormente, se procede a la implementación del activity **Login**. Este cuenta con dos EditText para introducir el email y la contraseña del usuario y un botón que, al ser pulsado, hace uso de los métodos de la clase BaseDeDatos para comprobar si el usuario está o no registrado y proceder a realizar el Login. Si el login es exitoso, se llama al activity UsuarioActivity, pasando como extra en el intent la información del usuario como objeto de la clase Usuario.

Después de implementar el activity Login, se procede a la implementación de **UsuarioActivity,** la pantalla que aparece cuando un usuario se ha logueado correctamente. Esta pantalla muestra un mensaje de bienvenida personalizado con el nombre del usuario, además de 3 botones que nos permiten realizar una nueva comparación, revisar el historial, o hacer el logout del usuario. Este activity recibe los datos del usuario para mantener la sesión abierta. La técnica empleada para mantener esa información disponible entre todos los activities es la de una clase Singleton o de instancia única [7]. De esta forma, en esta activity se guarda la información del usuario, y ésta estará disponible en todas las demás, usando los métodos *get* y *set* correspondientes sobre la instancia única al objeto *usuario* almacenado.

De las tres opciones que tiene el usuario, se procede en primera instancia a implementar el activity ComparacionActivity. Esta pantalla permite al usuario escoger las provincias/municipios a comparar, mediante el uso de spinners [11], [12], [13], y cuenta con un botón para lanzar la comparación. Es la activity más compleja, ya que, dentro de ella, se implementan funciones para poder configurar los spinners leyendo los datos a visualizar desde un archivo CSV, así como el lanzamiento de las peticiones HTTP y la recepción de los json con los datos de cada una de las búsquedas.

Para poder hacer las peticiones que nos devuelven los datos climatológicos de cada municipio, se necesita proporcionar en la petición como parámetro un código asociado a cada municipio. Estos códigos los obtenemos de la página web del INE **www.ine.es**

Por otra parte, para la configuración de los spinners se utiliza un archivo con la relación de todos los municipios de cada una de las provincias, disponible en la misma web. Este archivo, en definitiva, contiene una relación de todos los municipios de España, asociados a su provincia y con su código único de municipio.

A partir del CSV original, se deciden crear diferentes CSV derivados, con el objetivo de optimizar la búsqueda de elementos a mostrar en los spinners. De esta forma, se genera:

- Un CSV con la relación de todas las provincias disponibles.
- Un CSV por provincia, con la relación de todos sus municipios.

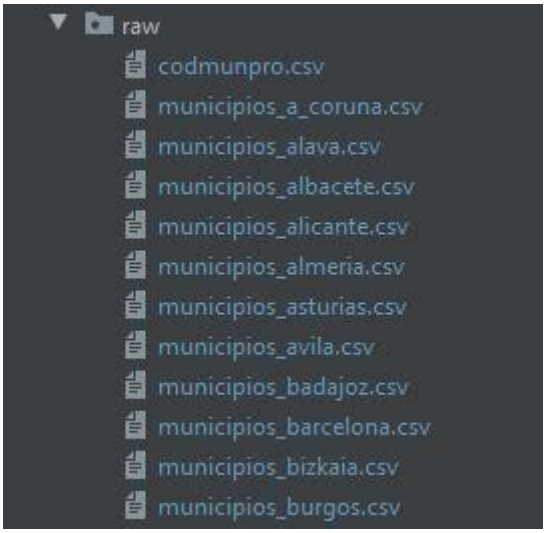

Imagen 13: Archivos .csv.

Para poder hacer uso del archivo CSV se procede a crear la clase **CSVFile** [14], que devuelve el contenido de dicho archivo como una lista de arrays de strings, donde cada elemento de la lista se corresponde con una fila del archivo CSV, y cada elemento del array de strings se corresponde con cada uno de los elementos separados por coma dentro de dicha fila del CSV.

La lógica de configuración de los spinners consiste en los siguientes pasos:

- Se habilita el uso de los spinners asociados a las provincias, cuyos elementos son leídos del CSV con el nombre de todas las provincias.
- Si el usuario escoge una provincia, el spinner de municipio asociado se habilita, y se configura leyendo el CSV asociado a esa provincia.
- Si el usuario cambia de provincia, el spinner se resetea y se vuelve a leer otro CSV, con los municipios de la nueva provincia.
- Si el usuario pone el spinner de provincia al valor por defecto (“seleccione una provincia”), el spinner del municipio se vuelve a deshabilitar y se elimina el posible elemento visualizado en ese momento, poniendo por defecto “seleccione un municipio”.

Para conocer más detalles sobre el funcionamiento de esta pantalla, consultar el manual de usuario en la sección 11.

Si el usuario acciona el botón a su disposición y ha seleccionado dos municipios de dos provincias, el activity se encarga de realizar dos peticiones HTTP a la API de AEMET.

Para la implementación de las peticiones HTTP se hace uso de las librerías **Retrofit** [18] y **Gson** [19] (se añaden dichas librerías al archivo build.gradle para poder trabajar con ellas).

```
40        implementation 'com.squareup.retrofit2:retrofit:2.5.0'
41        implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
```

Imagen 14: Librerias Retrofit y Gson.

La librería **Retrofit** nos permite hacer las **peticiones** a la API REST de AEMET para traer de esta los objetos en formato Json. Por su parte, con la librería Gson podemos serializar y deserializar los objectos para su conversión a formato Json.

Para poder utilizar la librería Retrofit, es necesario implementar una interfaz que nos permite configurar y estructurar las peticiones a realizar (en nuestro proyecto dicha interfaz se llama **JsonPlaceHolderApi**). Siguiendo la documentación del API de AEMET, se configuran dos peticiones independientes. La primera necesita del código único de municipio, además de una clave que se ha de generar desde la propia web de AEMET, y nos devuelve un json básico que contiene, entre otros atributos, uno denominado “datos”, que es otra dirección http con los datos propiamente de la predicción a 7 días para el municipio indicado en la primera petición. Por tanto, en la interfaz, se configura una segunda petición con la url indicada como resultado de la primera de ellas.

Request URL

```
https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/36039
```

Response Body


```
{
  "descripcion": "exito",
  "estado": 200,
  "datos": "https://opendata.aemet.es/opendata/sh/89315117",
  "metadatos": "https://opendata.aemet.es/opendata/sh/dfd88b22"
}
```


Imagen 15: Petición inicial con el código único de municipio y respuesta recibida.

Imagen 16: Petición secundaria con el resultado del campo "datos" de la petición inicial.

Cada uno de los json devueltos por las peticiones, tiene que tener asociada una clase que contenga exactamente los mismos atributos y con los mismos nombres. Por tanto, se crean todas las clases necesarias para cada una de las peticiones.

- La clase Model200 modela el json devuelto por la primera de las peticione HTTP (incluye la url para la segunda petición).
- La clase PrediccionMunicipio modela el json devuelto por la segunda de las peticiones HTTP (la predicción en sí).

Como el json devuelto por la segunda petición tiene una complejidad considerable, se busca alguna alternativa que facilite la implementación de la clase. Mediante el uso de una herramienta web [20] se consigue obtener la definición de la clase JAVA que modela la predicción de manera automática. Para poder modelar el json completamente, se (auto)generan las siguientes clases:

- PrediccionMunicipio (incluye a todo el resto de clases)
- CotaNieveProv
- Dato
- Dium
- EstadoCielo
- HumedadRelativo
- Origen
- Prediccion
- ProbPrecipitacion
- RachaMax
- SensTermica
- Temperatura
- Viento

Las peticiones HTTP se realizan de manera asíncrona, utilizando el método *enqueue* que nos proporciona Retrofit (el recomendado por la propia librería). Como se lanzan dos peticiones, es necesario implementar un mecanismo de sincronización, de forma que el activity sea capaz de darse cuenta cuando han llegado las respuestas a las dos peticiones. Dicho mecanismo utiliza un índice de petición que permite identificarla (petición 1 y petición 2), de forma que, con ese índice, se establece un flag de llegada de respuesta y se comprueba si la otra petición ya ha llegado (comprobando el valor del flag de la otra petición). Si se encuentra un *true* quiere decir que ambas respuestas han llegado, y, por tanto, ya se pueden mostrar los resultados al usuario. Es en este punto cuando se lanza un nuevo intent a la activity ResultadoComparacionActivity.

En el activity ResultadoComparacionActivity se muestran los resultados de las predicciones para los dos municipios seleccionados en el activity anterior. Para representar los resultados se utilizan diferentes TextView a modo de tabla, donde tenemos 3 columnas: a la derecha los valores de uno de los municipios, a la izquierda los valores del otro, y en el medio, los valores a comparar (temperatura mínima y máxima, velocidad de viento de racha, probabilidad de precipitación, …).

Además de mostrarse los resultados, estos se guardan en la base de datos. Para ello se emplea el método insertComparacion de la clase base de datos. Este método necesita el id del usuario (se obtiene usando el get correspondiente a la clase singleton), y las dos búsquedas comparadas. Primero, se inserta cada uno de las búsquedas en la tabla de búsquedas, luego se obtiene el ID asignado por la propia base de datos (columna autoincrement) para ser introducido en la tabla comparación, junto al id del

usuario y a la fecha actual (fecha del sistema). En este punto, para poder trabajar con las fechas correctamente, se utiliza la clase SimpleDataFormat [16], que nos permite convertir entre tipos de datos Date y numéricos, así como convertir a strings para ser representados en la interfaz gráfica.

En este punto, la funcionalidad de realizar una comparación, mostrar el resultado y guardarlo en la base de datos está implementado. Se procede a continuación a la implementación de la funcionalidad de revisión del historial de un usuario.

Para ello, se implementa un nuevo activity, HistorialActivity, al que se accede pulsando el botón de revisar historial en el activity UsuarioActivity. En este activity, se muestran las seis comparaciones más recientes realizadas por el usuario. Para mostrar los resultados, se utilizan seis Textview, en los que se visualiza la fecha de visualización y los municipios comparados. Si el número de comparaciones realizadas por el usuario es menor a 6, los Textview restantes se quedan vacíos. Además, se habilita un onClickListener para aquellos que contienen alguna información. En la función setOnClickListener se programa el uso de un nuevo intent para pasar a una nueva activity, ResultadoHistorialActivity, cuyo layout coincide plenamente con el de ResultadoComparacionActivity, y en el que se muestran los datos de la comparación recuperados de la base de datos (en este caso no es necesario guardar ningún dato nuevo en la base de datos).

Por último, se programa la funcionalidad de logout, a la que se accede también desde el activity UsuarioActivity accionando el botón correspondiente. Al pulsar este botón, se borra el usuario logueado, eliminando la información de la clase Singleton, y se vuelve a la pantalla inicial, que permite realizar el registro o el login de un usuario registrado.

6. Despliegue y pruebas

A medidas que se avanza en el desarrollo del proyecto se van realizando diversos test y pruebas funcionales basadas en los casos de uso.

6.1. Plan de pruebas.

Test 1	**El email introducido es valido**	
UC asociado	**UC1**	
Requisitos verificados	1	
Pruebas a realizar	• EL usuario introduce un email sin @. La aplicación devuelve un toast indicando que el email no es válido	Verificacion: OK (30/10/2020)
	• El usuario introduce un email sin dominio. LA aplicación devuelve un toast indicando que el email no es válido	Verificacion: OK (30/10/2020)
	• El usuario introduce un email correcto. La aplicación nos informa con un toast que el registro es correcto	Verificacion: OK (30/10/2020)

Tabla 6: definición test 1.

Test 2	**El password introducido es valido**	
UC asociado	**UC1**	
Requisitos verificados	**1**	
Pruebas a realizar	• EL usuario introduce un password sin ningún número. La app nos informa con un toast como debe de ser la contraseña.	Verificacion: OK (31/10/2020)

	• EL usuario introduce un password sin ninguna letra mayúscula. La app nos informa con un toast como debe de ser la contraseña.	Verificacion: OK (31/10/2020)
	• EL usuario introduce un password sin ninguna letra. La app nos informa con un toast como debe de ser la contraseña.	Verificacion: OK (31/10/2020)

Tabla 7: definición test 2.

Test 3	**El login se realiza correctamente**	
UC asociado	**UC2**	
Requisitos verificados	**2**	
Pruebas a realizar	• EL usuario introduce un email que no está registrado para iniciar sesión. La app nos informa con un toast de que los datos son incorrectos.	Verificacion: OK (31/10/2020)
	• EL usuario introduce un password que no está registrado para iniciar sesión. La app nos informa con un toast de que los datos son incorrectos.	Verificacion: OK (31/10/2020)
	• EL usuario introduce un email y password correctos. La app nos informa con un toast de que el inicio de sesión es	Verificacion: OK (31/10/2020)

	correcto y pasa a la pantalla de usuario.	

Tabla 8: definición test 3.

Test 4	**El usuario cierra sesión correctamente**	
UC asociado	**UC3**	
Requisitos verificados	**8**	
Pruebas a realizar	• EL usuario cierra su sesión al pulsar el botón logout. La app informa con un toast de que la sesión se cerró correctamente.	Verificacion: OK (03/11/2020)
	• Tras cerrar la sesión, se carga automáticamente la Main_activity o pantalla principal	Verificacion: OK (03/11/2020)

Tabla 9: definición test 4.

Test 5	**El usuario puede realizar la comparación meteorológica de 2 localidades**	
UC asociado	**UC4**	
Requisitos verificados	**4**	
Pruebas a realizar	• EL usuario Puede seleccionar 2 provincias y 2 localidades de dichas provincias para realizar la comparación del tiempo.	Verificacion: OK (21/11/2020)
	• Tras seleccionar las localidades, puede pulsar un botón el cual conduce a la pantalla en la que se	Verificacion: OK (21/11/2020)

	muestran los datos requeridos de las 2 localidades comparadas.	

Tabla 10: definición test 5.

Test 6	**Las comparaciones se guardan de forma automática en la base de datos de la aplicación**	
UC asociado	**UC6**	
Requisitos verificados	**6**	
Pruebas a realizar	• Tras realizar una comparación, ésta se guarda de forma automática en la base de datos del aplicativo.	Verificacion: OK (21/11/2020)

Tabla 11: definición test 6.

Test 7	**El usuario puede acceder a las 6 últimas comparaciones que realizó**	
UC asociado	**UC7**	
Requisitos verificados	**7**	
Pruebas a realizar	• Existe un botón en la pantalla de usuario desde el cual se puede acceder al historial de las 6 últimas comparaciones realizadas por el usuario.	Verificacion: OK (21/11/2020)
	• La pantalla que nos muestra las 6 últimas comparaciones realizadas, nos permite seleccionar cualquiera de ellas para visualizarla.	Verificacion: OK (21/11/2020)

Tabla 12: definición test 7.

Tras la realización de los test, se puede comprobar que la aplicación cumple de forma satisfactoria todos los casos de uso.

7. Conclusiones.

En este apartado se analiza el proceso de desarrollo de la aplicación y se tratará de sacar algunas conclusiones tanto del desarrollo como del aplicativo en sí.

7.1. Objetivos alcanzados.

A rasgos generales se han alcanzado los principales objetivos propuestos para el desarrollo de esta app. Se consiguió crear una aplicación que nos permite un correcto registro de usuario para su uso, la posibilidad de iniciar sesión como usuario y lo que es más importante, realizar una comparativa meteorológica de 2 localidades guardando un historial para futuras consultas.

Por otra parte, durante el desarrollo del proyecto surgieron algunas complicaciones, siendo la más relevante la dificultad para hacer la previsión del tiempo a varios días desde la búsqueda, este problema, vino dado por la complejidad a la hora de transformar los datos en formato Json que nos proporciona la API de Aemet, es por ello que se decidió que la previsión sería para el día siguiente al que se realiza la búsqueda ya que otro factor importante fue el plazo de entrega del proyecto, escaso para las horas que se le pudo dedicar al desarrollo del mismo.

7.2. Conclusiones del trabajo.

Este desarrollo permitió al alumno adquirir más conocimiento sobre las tecnologías utilizadas, mejorar los métodos para buscar y encontrar la información necesaria para llevar a cabo un proyecto de desarrollo y también adquirir buenas costumbres de trabajo.

7.3. Vías futuras.

Se plantean en este apartado algunas mejoras que se podrían realizar sobre la aplicación, así como nuevas funcionalidades a implementar:

- La interfaz gráfica podría mejorarse haciéndola más atractiva, y gráfica, ya que, en esta implementación inicial, casi toda la interacción con el usuario es mediante el uso de EditText y TextViews, y no se utilizan información gráfica (iconos, símbolos, gifs, …)

- Podrían implementarse mejoras en la realización de las predicciones. Actualmente, la aplicación solo permite comparar el tiempo para el día siguiente al de la comparación, por lo que la funcionalidad se ve limitada. Esta flexibilización podría realizarse teniendo en cuenta las restricciones que impone la API empleada, la de AEMET, que solo devuelve predicciones a una semana. Si se quisiera aumentar ese rango de días, habría que buscar una alternativa para la fuente de datos.
- También existe una restricción en cuanto al ámbito geográfico de aplicación, ya que actualmente solo nos sirve para comparar municipios de España. Y dentro de España, solo contamos con una predicción por municipio y no una predicción especifica dentro de un municipio. Se podrían explorar diferentes alternativas para poder ampliar esta funcionalidad.
- La búsqueda de los municipios a comparar se podría realizar seleccionando sobre un mapa, para que la experiencia de usuario se viese mejorada.
- Se podría implementar una funcionalidad de “usuario invitado” de forma que se puedan realizar comparaciones sin realizar registro ni login, y sin que los datos de esa comparación se guarden en ninguna base de datos.
- La base de datos en la que se guardan las comparaciones es una base de datos en local, de forma que un usuario no puede tener los mismos datos en dos dispositivos diferentes. Se podrían migrar los datos a una base de datos externa a la que la aplicación se conectase independientemente del dispositivo en el que se esté ejecutando.
- Se podría incluir funcionalidades relativas a compartir y exportar datos de comparaciones, para ser guardados externamente o para ser compartidos en redes sociales o mediante aplicaciones de mensajería (email, mensajería instantánea).
- No se han incluido ningún recurso para poder monetizar la aplicación (servicios premium, publicidad, …) que nos permita sacar rendimiento económico de la aplicación.

8. Glosario.

- AEMET: Agencia Estatal de Meteorología.
- API: *Application Programming Interface.*
- APK: *Android Application Package.*
- HTTP: *Hypertext Transfer Protocol.*
- IDE: *Integrated Development Environment.*
- JSON: *JavaScript Object Notation.*
- URL: *Uniform Resource Locator.*

9. Bibliografía.

[1] Material de didáctico del módulo de programación multimedia y dispositivos móviles.

[2] Videotutorias del módulo programación multimedia y dispositivos móviles.

[3] Proyecto propio del módulo programación multimedia y dispositivos móviles.

[4] Teamgantt: https://www.teamgantt.com/

[5] ¿Cuál es la mejor forma de validar un email en Android? (recurso web): https://es.stackoverflow.com/questions/29870/cu%C3%A1l-es-la-mejor-forma-de-validar-un-email-en-android

[6] How to validate a Password using Regular Expressions in Java (recurso web): https://www.geeksforgeeks.org/how-to-validate-a-password-using-regular-expressions-in-java/

[7] Singleton (recurso web): https://es.wikipedia.org/wiki/Singleton

[8] Android Studio setText with String set in Code (recurso web): https://stackoverflow.com/questions/46713418/android-studio-settext-with-string-set-in-code

[9] java.lang.RuntimeException: Parcelable encountered IOException writing serializable object (recurso web): https://stackoverflow.com/questions/13493052/java-lang-runtimeexception-parcelable-encountered-ioexception-writing-serializa/44082022

[10] Eliminar la pila de actividades (Back Stack) en Android (recurso web): https://elbauldelprogramador.com/eliminar-la-pila-de-actividades-back-stack-en-android/

[11] Disabling Spinner in android (recurso web): https://stackoverflow.com/questions/5986130/disabling-spinner-in-android#:~:text=you%20can%20set%20android%3Aclickable,the%20spinner%20for%20click%20event%20.&text=You%20can%20set%20this%20in,setEnabled(boolean)%20from%20View%20.

[12] Change color of the Spinner Android – Androchunk (recurso web): https://androchunk.blogspot.com/2019/02/change-color-of-spinner-android.html

[13] Cómo personalizar un Spinner en Android Studio (recurso web): https://www.youtube.com/watch?v=rCT7EDJ3em4

[14] How to parse CSV file into an array in Android Studio (recurso web): https://stackoverflow.com/questions/38415680/how-to-parse-csv-file-into-an-array-in-android-studio/38415815

[15] How can I pad an integer with zeros on the left? (recurso web): https://stackoverflow.com/questions/473282/how-can-i-pad-an-integer-with-zeros-on-the-left

[16] Java Convert Date to String (recurso web): https://www.javatpoint.com/java-date-to-string

[17] Documentación de AEMET OpenData (recurso web): https://opendata.aemet.es/dist/index.html?

[18] Retrofit - A type-safe HTTP client for Android and Java (recurso web): https://square.github.io/retrofit/

[19] GSON - A Java serialization/deserialization library to convert Java Objects into JSON and back (recurso web): https://github.com/google/gson

[20] jsonschema2pojo - Generate Plain Old Java Objects from JSON or JSON-Schema (recurso web): http://www.jsonschema2pojo.org/

10. Manual de instalación.

Para la correcta instalación de la app, es necesario descargar el archivo .apk en un dispositivo Android (Tablet o smartphone) con una versión de Android 10.0. Tras la descarga, la instalación de la app es automática.

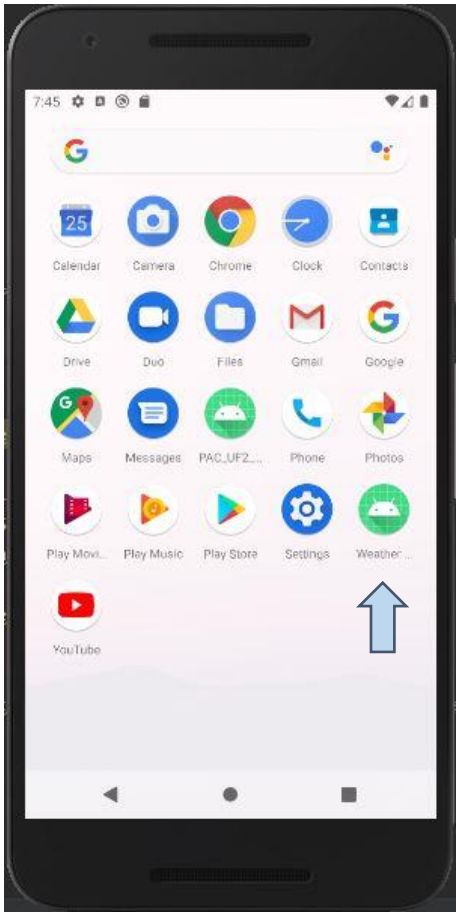

Imagen 17:App instalada en smartphone.

11. Manual de usuario.

Cuando entramos en la app, nos encontramos con la pantalla de inicio, desde la que podemos registrarnos como usuario (necesario para su utilización) o identificarnos como usuario.

Al pulsar sobre el botón de registro, vamos a la pantalla desde la cual podremos registrarnos introduciendo un nombre de usuario, un email válido y una contraseña formada por 8 caracteres alfa numéricos y al menos 1 mayúscula.

Imagen 18:Pantalla de registro de usuario.

Después de realizar un registro correcto, podemos identificarnos como usuario introduciendo nuestro email y contraseña. Una vez que nos identificamos como, la app nos lleva a la pantalla de usuario desde la que tenemos la opción de hacer una nueva **comparativa,** revisar el **historial** de las 6 últimas comparativas realizadas o **cerrar sesión**, para ellos disponemos de 3 botones debidamente identificados.

Si pulsamos sobre el botón de realizar comparativa, este nos llevará a una pantalla desde la cual podremos seleccionar las provincias y municipios que queremos

comparar, deberemos seleccionar primero la provincia para que se habiliten los municipios disponibles y posteriormente pulsar el botón disponible para hacer la comparación.

Imagen 19:Pantalla de selección de localidades.

La siguiente pantalla nos muestra los datos meteorológicos de las 2 localidades seleccionadas anteriormente, mostrándonos también la fecha para dicha comparación (siempre el día siguiente al que se realiza la búsqueda).

Imagen 20:Pantalla de resultado de la comparación.

Si queremos revisar nuestro historial de búsquedas, basta con pulsar el botón historial que tenemos disponible desde nuestra pantalla de usuario, de esta forma, se nos mostrará una pantalla con nuestras últimas comparativas realizadas (hasta un máximo de 6) y pinchando sobre cada una, podremos volver a visualizarla.

Imagen 21:Pantalla de historial de usuario.