Código Del Sur

Evaluación técnica Android

Martina Severo

Julio de 2021

Contenido

1. Prototipo de la aplicación

A continuación, se presenta un prototipo de como luciría la aplicación:

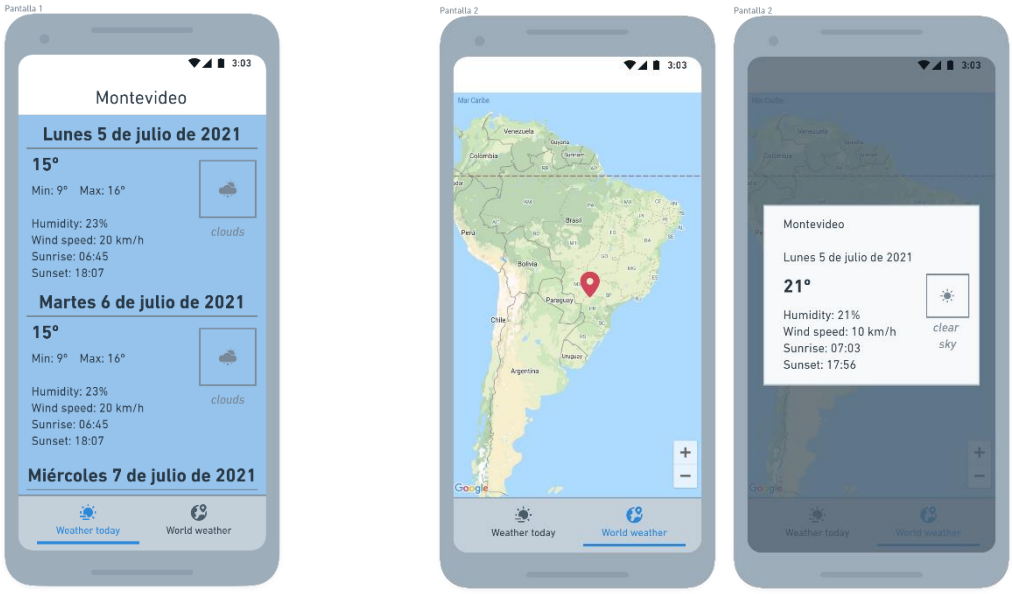

Imagen 1 - Pantalla 1 (izquierda) y pantalla 2 (derecha)

En la pantalla 1 se muestra una lista con los datos del clima del día actual y los siguientes 7 días. También se muestra una imagen correspondiente a la condición del clima y una breve descripción.

En la pantalla 2 se muestra un mapa mundial, donde se puede seleccionar un punto para luego mostrar una ventana emergente con los datos del clima correspondientes a dicha ubicación.

Además, ambas pantallas están ligadas a una Bottom Navigation Bar y los iconos a utilizar para el clima serán en formato SVG.

2. Flujo de trabajo en Git

Para trabajar en Git, se utilizó el flujo de trabajo GitFlow, haciendo uso de una rama master, develop y features. Las features utilizadas fueron las siguientes:

- **Feature backend-connection**: se trabajó todo lo relacionado a la conexión de la aplicación con el backend. En este caso, con la API de OpenWeather para conseguir los datos del clima y con los servicios de Google Maps para cargar el mapa mundial.

- **Feature frontend-design**: se trabajó con lo relativo al diseño de las pantallas y permisos, incluyendo los iconos en formato SVG.

3. Cumplimiento de RF y RNF

Requerimientos funcionales (RF)

- La aplicación consiste en una Bottom Navigation Bar, la cual tiene 2 pantallas. La primera muestra el clima de hoy y los siguientes 7 días. La segunda pantalla consta de un mapa mundial, donde se puede seleccionar un punto especifico y se mostrarán los datos del clima correspondientes a dicho lugar.
- Los iconos utilizados son en formato SVG.

Requerimientos no funcionales (RNF)

- La aplicación se adapta a orientación portrait y landscape.
- Para la pantalla 1 (día actual y siguientes 7 dias) se utilizó un RecyclerView.
- La aplicación fue escrita en el lenguaje Kotlin y se usó Android Studio como IDE.
- La comunicación con la API fue realizada utilizando la librería Retrofit.
- Las dependencias fueron agregadas mediante Gradle.
- No se utilizaron wrappers de la API de OpenWeatherMap.
- En cuanto al diseño y código, se intentó seguir buenas prácticas y ser lo más prolijo posible, dentro del tiempo con el que se dispuso.
- A la hora de probar con versiones (a partir de la 5.0) ciertas versiones más viejas (por ejemplo, la 5.0 o la 5.1) no funcionan con la librería Dexter. Por ello, esta aplicación cumple parcialmente con el requerimiento de obtener datos para el día actual y los próximos 7 días, ya que, para estos casos, carga los datos de la ubicación predeterminada (Montevideo) y no la actual del usuario.

4. Recursos utilizados y aclaraciones

Librerías utilizadas

- **Retrofit**: para poder realizar peticiones HTTP en Android. En este caso, para poder obtener los datos de la API de OpenWeather.
- **okHttp**: para gestionar las peticiones HTTP y crear interceptores. En este caso, se creó uno para agregar la API key como query param.
- **Dexter**: para gestionar los permisos más fácilmente (por ejemplo, los de ubicación).
- **Google Play Services**: necesarios para la utilización de la API de Google Maps.

Proyecto en Android Studio

- Dentro de la carpeta **common** se definieron dos clases, una para almacenar las constantes (url, API key y nombre del parámetro correspondiente para pedir datos a la API de OpenWeather) y otra para la instancia de la aplicación.
- Clase **OpenWeatherRepository:** para definir las funciones a llamar para realizar las requests para pedir los datos del clima proporcionados por la API de OpenWeather. Una función es para el caso de la pantalla 1 (clima de hoy y próximos 7 días), donde se excluyen los campos *hourly* y *minutely* de la response obtenida. La otra es para el caso de la pantalla 2 (clima de punto especifico del mapa), donde se excluye también el campo *daily*.
- En cuanto a lo relacionando a Retrofit:
    - Como modelos, se crearon diversas clases, siendo OpenWeatherResponse la representación de la response completa. El campo *daily* de dicha response fue usado para mostrar los datos del clima en la pantalla 1, ya que dentro de esta información se incluye el día actual y los próximos 7. El campo *current* fue utilizado para mostrar los datos en la pantalla 2 del mapa, ya que solo se muestran los del día actual.
    - Clase **OpenWeatherInterceptor**: donde se agregan parámetros a la request. La misma se intercepta y luego sigue su camino con los nuevos datos agregados. En

este caso, se le adiciona la API key proporcionada por OpenWeather como query param en la URL, para cumplir con el formato de la request.

- Clase **OpenWeatherClient**: permite obtener el acceso a las peticiones definidas en la interfaz de Retrofit, de manera que cada vez que queramos hacer una petición, accederemos a ese cliente. En esta clase también se le agrega el interceptor al cliente, para poder agregar los parámetros deseados a la request. Aquí se aplica el patrón *Singleton*; cada vez que se invoque a esta clase, si ya existe una instancia de ella, la retorna, manejando una sola instancia durante todo el proceso. Solo crea la instancia de la clase si ésta es nula (que solo ocurriría la primera vez).
- Interfaz **OpenWeatherService:** donde se define la request de tipo GET con sus respectivos parámetros, para obtener los datos del clima proporcionados por la API de OpenWeather.

- Respecto a la interfaz de usuario (ui):
  - Para la pantalla 1, se hizo uso de un **Fragment**, el cual se comunica con un **ViewModel**, donde se llama al repositorio para hacer la petición y, un **RecyclerViewAdapter**, donde se conectan los elementos del layout con los datos del clima de la response obtenida, manejando cuales son mostrados.
  - Para la pantalla 2, se utilizó un **Fragment**, el cual se comunica con un **ViewModel**, donde se llama al repositorio para hacer la petición y, un **DialogFragment** para poder mostrar los datos del clima de un punto especifico seleccionado del mapa. En este último se hacen las conexiones con los elementos del layout con los datos de la response recibida y el manejo de los mismos para mostrarlos.
- **MainActivity**: donde se hacen las conexiones del layout con los elementos correspondientes para la navegación.

Manejo de errores de la aplicación

- En la pantalla 1, para el caso en que haya problemas al obtener los datos del clima para la ubicación actual del usuario, se cargarán los datos de la locación por defecto (Montevideo), indicándose con un mensaje.

- Para ciertas versiones más viejas se tuvieron problemas con el dialogFragment, mostrado al hacer clic en un punto del mapa en la pantalla 2. En la parte superior se muestra un espaciado extra que no debería aparecer (encima del timezone).
  También se aplicaron diferentes algoritmos según ciertas versiones debido a problemas de compatibilidad con ciertos recursos utilizados (comparando Build.VERSION.SDK_INT con *Build.VERSION_CODES.[Version]*):
  - En el OpenWeatherClient y WeatherTodayFragment, para versiones iguales o inferiores a Marshmallow se hace uso de otros métodos debido a problemas con los permisos en dichas versiones.
  - En el WeatherTodayRecyclerViewAdapter y WeatherMapDialogFragment, para versiones inferiores a Oreo, la conversión de los datetime proporcionados por la API de OpenWeather es realizada de forma distinta debido a problemas de compatibilidad con los Instant. Por esto, para dichas versiones, las fechas se muestran diferente (formato “05/07/2021”) que las versiones iguales o superiores a Oreo (formato “lunes, 5 de julio de 2021”).