THE METEO

Android Capstone Project

UDACITY

Android Kotlin Developer Nanodegree Program

Autor: Maria José Rojas Medrano

Email: maria.rojasmedrano@gmail.com

THE METEO

Introduction

“The Meteo” is an application that shows information regarding the weather and air pollution in the current location of the device. The application contains one main screen which shows several features:

- Current temperature.
- Current weather description (sunny, clear sky, rainy, etc).
- Current data values of uv index, wind speed, humidity, pressure, visibility and temperature “feels like”.
- Time of the current day sunrise and sunset.
- Daily forecast for the next 7 days (only max a min temperature).
- Current air quality (according to the European Union AQI). You can click on “see more” to check more details regarding the current air quality.
- It can show alerts if there are any. You can click on “see more” to check more details regarding the alert.

The application gets the information from “openweathermap”. It calls the following 2 API’s from the server to function correctly:

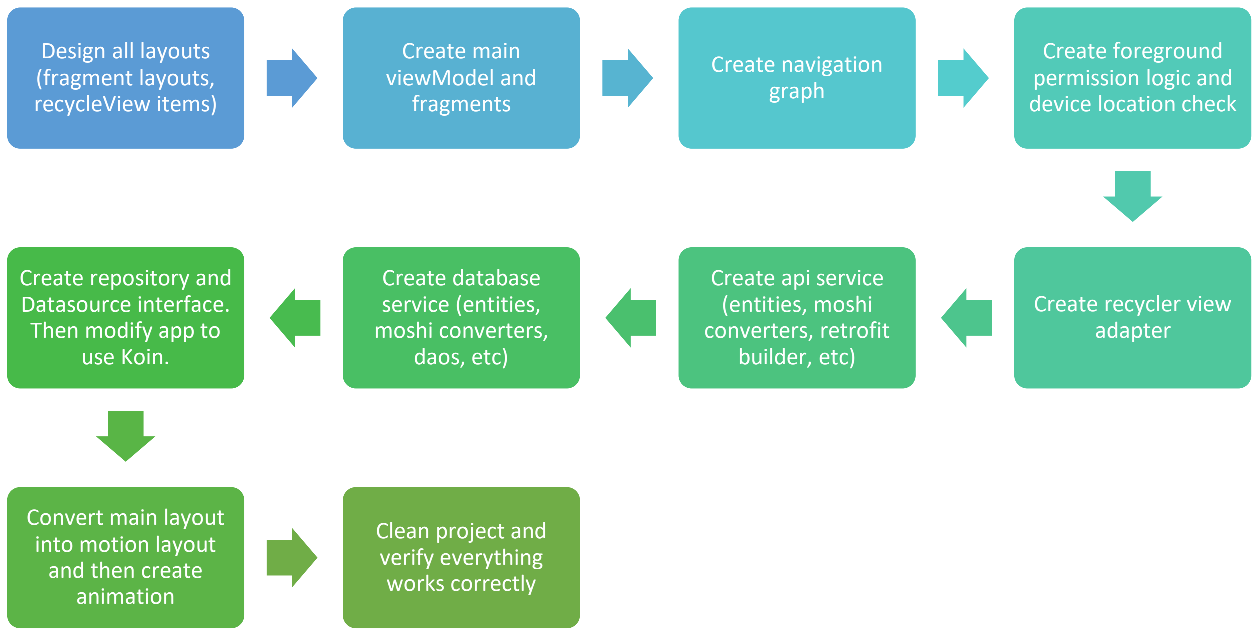
- To get the current and forecast weather data:
  https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API key}
- To get the current air pollution data:
  https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API key}

To understand better how these API’s works, you can check the following documentation:

- For air pollution data: https://openweathermap.org/api/air-pollution
- For weather data: https://openweathermap.org/api/one-call-api

Milestones

The following chart shows the milestones achieved to complete the project successfully:


Design all layouts (fragment layouts, recycleView items)
Create main viewModel and fragments
Create navigation graph
Create foreground permission logic and device location check
Create recycler view adapter
Create api service (entities, moshi converters, retrofit builder, etc)
Create database service (entities, moshi converters, daos, etc)
Create repository and Datasource interface. Then modify app to use Koin.
Convert main layout into motion layout and then create animation
Clean project and verify everything works correctly

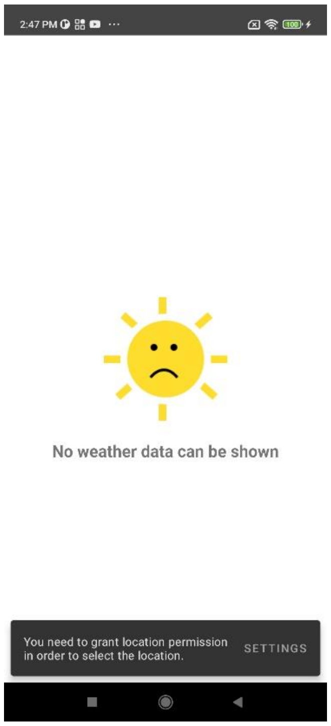
Project features and guideline

The following will show the major and minor features of the project with a guideline of the application functionality:

LOCATION

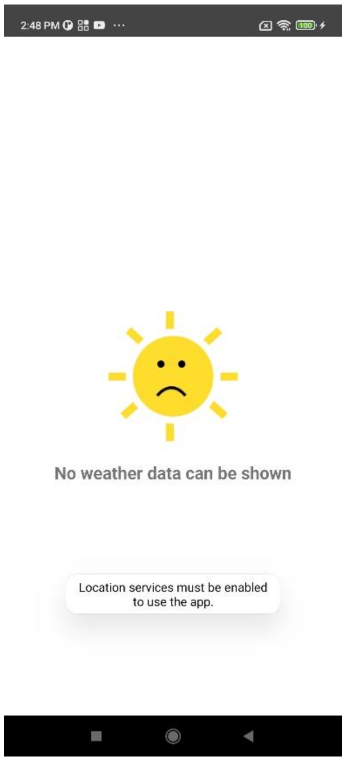
When the application is opened for the first time it will ask to the user to allow the foreground permission. There are two possibilities here:

1. *If denied*, a snack bar will appear to inform the user it's necessary to grant permission to use the app. The screen should look like this:



You need to grant location permission in order to select the location.

2. *If granted*, the app will check the if the device location is on. Again, there are two options in here:
    a. The location is *on*, so the app immediately goes to the main screen.
    b. The location is *off*, so the app will ask the user to able the location, if denied, a toast will show (image below) to inform the user the location is necessary for the app to function correctly, otherwise the app will go to the main screen.


No weather data can be shown

Location services must be enabled to use the app.

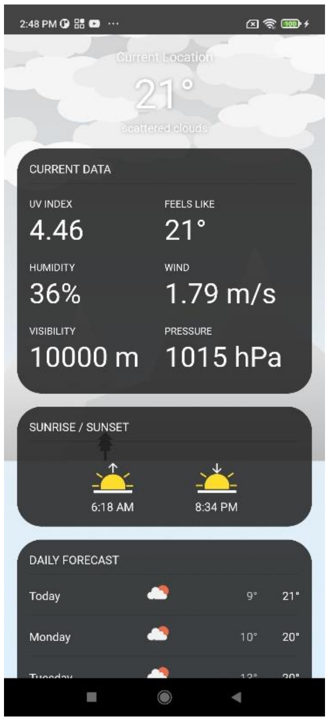
MAIN SCREEN

Once the permission is granted and location is on, the application will execute several steps before showing the main screen:

1. A progress bar will be shown until the data is ready to be shown in the screen.
2. Once the location is “on” the app gets the latitude and longitude values of the current location of the device.
3. Then it’s triggered the process to refresh the data. The main function is in the repository and it calls 2 API’s, one to get the weather data and one to get the air pollution data. Both values are then saved in their respective tables in the database.
4. The next step is to get the data from the database. There are two functions in the repository that are called: one to get the Weather data and another to get the air pollution data.
5. Once the Weather and AirPollution results are returned, then is created the weatherRecyclerView list which will contain the data to be shown on each item of the recycler view.
6. The weather data is then observed to define the background of the main screen. For example. If the current weather condition is cloudy, then the background image wil be set to “drawable.cloudy”. This was able to do because the server returns an “id” for each

weather condition. If the “id” goes from 802 to 804 then, the current weather is cloudy. For more information, you can find it in the following link: https://openweathermap.org/weather-conditions#Weather-Condition-Codes-2

7. The recycler view then is created for each item the list and the result of the main screen is the following:



2:48 PM

Current Location

21°

scattered clouds

CURRENT DATA

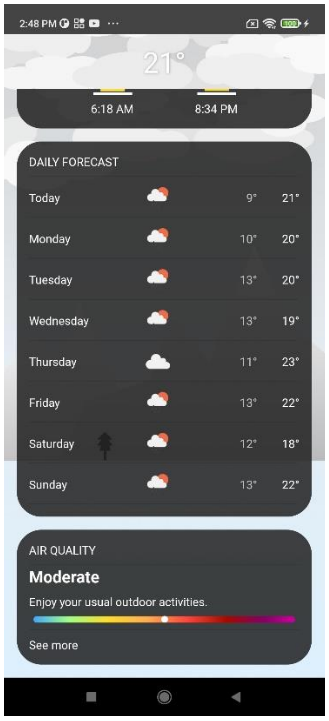
UV INDEX	FEELS LIKE
4.46	21°
HUMIDITY	WIND
36%	1.79 m/s
VISIBILITY	PRESSURE
10000 m	1015 hPa

SUNRISE / SUNSET

6:18 AM 8:34 PM

DAILY FORECAST

Today	9°	21°
Monday	10°	20°



2:48 PM

21°

6:18 AM 8:34 PM

DAILY FORECAST

Today	9°	21°
Monday	10°	20°
Tuesday	13°	20°
Wednesday	13°	19°
Thursday	11°	23°
Friday	13°	22°
Saturday	12°	18°
Sunday	13°	22°

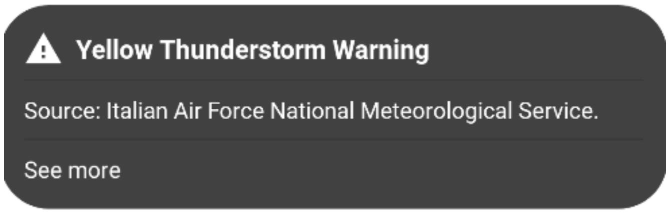
AIR QUALITY

Moderate

Enjoy your usual outdoor activities.

See more

8. As it can be seen in the image above, when it’s scrolled the recycler view there’s a change in the current temperature shown in the top of the screen. There’s an animation that decrease the size of the temperature and takes the transparency of the “current location” textView and the weather description to 0.0 (this way they are not visible at the end of the animation).
9. There’s another item in the recycler view that can or cannot be shown, it depends on the json response of the server. The alerts form part of the “current and forecast weather” API response when there’s any alert, otherwise is not returned, hence, the app does not show the “alert” item. The “alert” item that is added when needed in the recycler view is like this:

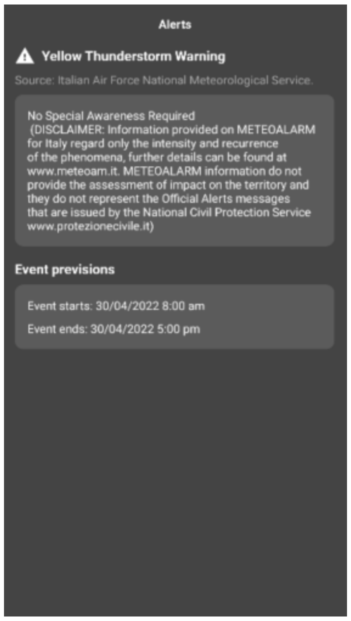


Yellow Thunderstorm Warning

Source: Italian Air Force National Meteorological Service.

See more

When you click on “See more” the application will navigate to another screen which shows more details regarding the alert. The image below shows an example of it:



Alerts

Yellow Thunderstorm Warning

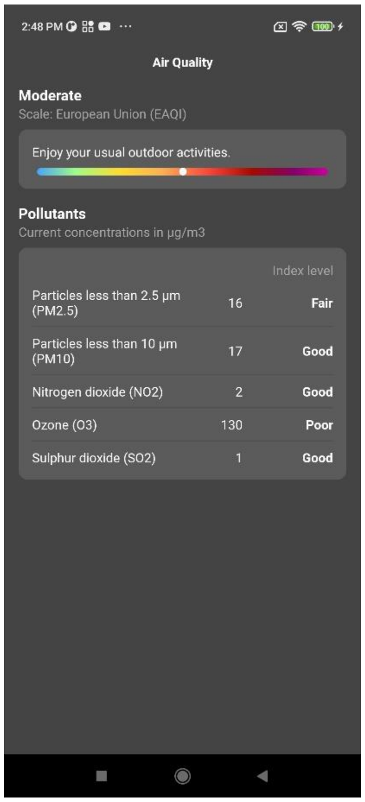
Source: Italian Air Force National Meteorological Service.

No Special Awareness Required
(DISCLAIMER: Information provided on METEOALARM for Italy regard only the intensity and recurrence of the phenomena, further details can be found at www.meteoam.it. METEOALARM information do not provide the assessment of impact on the territory and they do not represent the Official Alerts messages that are issued by the National Civil Protection Service www.protezionecivile.it)

Event previsions

Event starts: 30/04/2022 8:00 am
Event ends: 30/04/2022 5:00 pm

10. The last item of the recycler view contains also a clickable “See more”. When clicked, the app navigates to another screen which contains more detailed information of the current air quality. The image below shows an example of this screen.



Air Quality

Moderate

Scale: European Union (EAQI)

Enjoy your usual outdoor activities.

Pollutants

Current concentrations in μg/m3

		Index level
Particles less than 2.5 μm (PM2.5)	16	Fair
Particles less than 10 μm (PM10)	17	Good
Nitrogen dioxide (NO2)	2	Good
Ozone (O3)	130	Poor
Sulphur dioxide (SO2)	1	Good

Project rubrics

Criteria	specifications	result
Android UI/UX		
Build a navigable interface consisting of multiple screens of functionality and data.	Application includes at least three screens with distinct features using either the Android Navigation Controller or Explicit Intents.	The application contains three screens: the main screen (fragment_weather), the airPollution screen (fragment_air_quality) and the alerts screen (fragment_daily_alerts).
	The Navigation Controller is used for Fragment-based navigation and intents are utilized for Activity-based navigation.	The navigation controller is used to navigate between fragments.
	An application bundle is built to store data passed between Fragments and Activities.	This is also done to pass the data between the fragments
Construct interfaces that adhere to Android standards and display appropriately on screens of different size and resolution.	Application UI effectively utilizes ConstraintLayout to arrange UI elements effectively and efficiently across application features, avoiding nesting layouts and maintaining a flat UI structure where possible.	Constraint layout were created for all layouts designed in the app and were properly used to show the data
	Data collections are displayed effectively, taking advantage of visual hierarchy and arrangement to display data in an easily consumable format.	
	Resources are stored appropriately using the internal res directory to store data in appropriate locations including string* values, drawables, colors, dimensions, and more.	
	Every element within ConstraintLayout should include the id field and at least 1 vertical constraint.	
	Data collections should be loaded into the application using ViewHolder pattern and appropriate View, such as RecyclerView.	
Animate UI components to better utilize screen real estate and create engaging content.	Application contains at least 1 feature utilizing *MotionLayout* to adapt UI elements to a given function. This could include animating control elements onto and off screen, displaying and hiding a form, or animation of complex UI transitions.	Motion layout was used to animate the "current temperature" in the top of the main screen (fragment_weather). It was also created a motionScene xml to set the start and end of animation
	MotionLayout behaviors are defined in a *MotionScene* using one or more *Transition* nodes and *ConstraintSet* blocks.	
	Constraints are defined within the scenes and house all layout params for the animation.	
Local and Network data		

Connect to and consume data from a remote data source such as a RESTful API.	The Application connects to at least 1 external data source using **Retrofit** or other appropriate library/component and retrieves data for use within the application.	Retrofit was used to call the API's from openweatehr.org and Moshi was successfully used to convert the data that arrived from the server to local entities. All request handled in the appropriate thread
	Data retrieved from the remote source is held in local models with appropriate data types that are readily handled and manipulated within the application source. Helper libraries such as **Moshi** may be used to assist with this requirement.	
	The application performs work and handles network requests on the appropriate threads to avoid stalling the UI.	
Load network resources, such as Bitmap Images, dynamically and on-demand.	The Application loads remote resources asynchronously using an appropriate library such as **Glide** or other library/component when needed.	The application uses Glide in different occasions. From the network it's used to shown the weather description image in the Daily forecast.
	Images display placeholder images while being loaded and handle failed network requests gracefully.	
	All requests are performed asynchronously and handled on the appropriate threads.	
Store data locally on the device for use between application sessions and/or offline use.	The application utilizes storage mechanisms that best fit the data stored to store data locally on the device. Example: SharedPreferences for user settings or an internal database for data persistence for application data. Libraries such as Room may be utilized to achieve this functionality.	The data is stored in the database to persist the data. Converters were used to store and get the data properly
	Data stored is accessible across user sessions.	
	Data storage operations are performed on the appropriate threads as to not stall the UI thread.	
	Data is structured with appropriate data types and scope as required by application functionality.	
Android system and hardware integration		
Architect application functionality using *MVVM*.	Application separates responsibilities amongst classes and structures using the MVVM Pattern:	The application uses the MVVM pattern and uses best practices
	Application adheres to architecture best practices, such as the observer pattern, to prevent leaking components, such as Activity Contexts, and efficiently utilize system resources.	
Implement logic to handle and respond to hardware and system events that impact the Android Lifecycle.	Beyond MVVM, the application handles system events, such as orientation changes, application switching, notifications, and similar events gracefully including, but not limited to: -Storing and restoring state and information -Properly handling lifecycle events in regards to behavior and functionality	The application does not change orientation (it's made that way) because the design would not look good. It's implemented a bundled to store the latitude and longitude. The

	-Implement bundles to restore and save data -Handling Android Permissions	permissions event is handled as well
Utilize system hardware to provide the user with advanced functionality and features.	Application utilizes at least 1 hardware component to provide meaningful functionality to the application as a whole.	The application uses the location component to obtain the latitude and longitude of the current device location. This information is needed to call the API's.
	Permissions to access hardware features are requested at the time of use for the feature.	
	Behaviors are accessed only after permissions are granted.	