- **App name**: Local Weather
- **About**: A small app to check the weather on your location.

- **APIs used**:
  -com.android.support.design:28.0.0
  -com.google.android.gms:play-services-location:16.0.0
  -OpenWeatherMap
  -FusedLocationApi over android legacy Location api. (See the app's build.gradle file).

Summary

- Using OpenWeatherMap Api (https://openweathermap.org/) we create a small but simple weather app where we can check the current location weather in real time, and/or non real time.

- We will get a JSON response (could also be XML, but we stick to JSON this time) that we use for show on the app UI. The system will get a snapshot of such weather with a timestamp, valid for 24 hours.
- For real-time mode: We need the Gps ON, then we retrieve the location and the weather, using Google's FusedLocationClient api and connecting to OpenWeatherMap for a weather reading.

- For non-real-time: It is required to have had to use our app within the latest 24 hours. If the app happens to have a snapshot of the weather consultation that's younger than 24 HOURS, it will be used. If the snapshot is older it will be discarded and not used. If there's no snapshot, NO DATA will be shown.

Engineering decisions:

- We require the usage of google play services, therefore the app will look for it but we can still use the app with probable malfunctioning (very unlikely since most of phones do have google play services);

- The offline system uses shared preferences but a database is implemented to achieve the same goal (to find out how, read the last paragraph of this DOC)
- JSON is chosen over XML when parsing the weather server's answer (personal preference). Thought of using GSON library, but didn't because the space required by the files to download, isn't worth the amount of code we will write.

- Because the location service will only return one answer when we open the app or when we hit the "reload" button, battery shouldn't be drained, besides the battery required by the GPS hardware.

- -Due to the server answer in Kelvin degrees, timestamps instead of dates and wind direction degrees in number instead of Directions (N, S, W, E), some methods have been created for convenience.

- Together with the JSON comes the name of an image file. We can download them or use Picasso or GLide . Didn't do it, downloaded the pictures instead. It's not worth it for a single picture and in case our device doesn't have internet connection and our weather snapshot is within 24 hours (means still usable), we can use the files locally. Obviously, that makes the app's size somewhat bigger.

- Fetching data from weather server, internet connection to achieve it, unit conversions and a "loading" view are done by using an implementation of an AsyncTask as we dont require a service which is a heavy object in terms of workload and memory usage. AsyncTask is lightweight and achieves its goal quickly to stop afterwards.

- The "loading" progress bar is almost impossible to see due to the speed of the AsyncTask so i created a TimerTask with a Runnable that keeps it working for 0.5 seconds after the AsyncTask finishes fetching data.

- We use an HttpsConnection within the AsyncTask to connect with the weather server and download its data into an array of bytes that will be converted into a string, to be (JSON) parsed later.

- “Reports” Folder contain coverage reports.

<u>**Other possible implementations**</u>

- Implementing the location system using the legacy api instead of using google play services for retrocompatibility.

- Make app visually nicer more attractive for the user experience (styling).

- Implementing blinking animation for "Loading..." text and a new horizontal ProgressBar underneath with load percentage.

- Implementing a ToolBar or a BottomNavigationView or MenuItem to add a menu to search for weather in the future (as much as the OpenWeatherMap API allows us). An EditText to type the city name would be also valid.

- Adding new screens via fragment and a ViewPager to pass through the next days weather forecast.

- Using ViewModel and LiveData would be useful as ViewModel retains its data through configuration changes and therefore it’s good practise to use for screen rotations that might call an asynctask to execute multiple times for no reason.
  For this reason we would also use LiveData, as it is lifecycle-aware and only updates viewmodel data if there are any changes. We didn’t allow this happening, only needed to force the app to draw ALWAYS in portrait mode. No turning screen, no redrawing, no reloading asynctask and no viewmodel-livedata are required anymore. We save a lot of code at a price, we can’t turn the screen anymore.

TESTING:

- Tests has been made with JUnit (test file is /app/src/test/java/com/juan/weatherapp/BaseUnitTests.java)
- Instrumented test file is /app/src/androidTest/java/com/juan/weatherapp/InstrumentedTest.java
- Multiple methods accross the app have been marked public instead of private for the sake of testing.
- Added extra base constructors in MainActivity and SingletonClass for testing reasons (they can be erased and has notes next to them indicating to do so).
- Last three instrumented tests won't pass as they require using other elements from the app (views and other classes) to be initialized in the methods they test.
  Doing so is ok but would take a long time and restructuring the app and for the length of the code which I find useless as it can be tested on runtime (the app won't work properly if those methods don't work).
  Those tests are: location_test(), test_enable_gps() and test_deployment(). All of them commented out (located in InstrumentedTest.java file). Uncomment in case we want to try them or simply enhance the code to make them useful.
- For testing reasons, I enabled the write/read methods from class "DataFlowController" named writeToDatabase(...) and readFromDataBase(...) but I leave them disabled within the method MainActivity.enableGps()
  and MainActivity.getLocation() (take special care if performing coverage tests as they arel be disabled).
- As mentioned above, caching weather will be done with sharedpreferences but using a local database instead (or together but pointless), is possible by commenting/uncommenting these methods callings. Turning on handling by database is next to the methods that handles the SharedPrefences. Looking at the methods i named before, makes this task very straightforward.
- Javadoc available in file: /WeatherApp/Javadoc/index.html.
  Contains full description of classes and methods of the entire project.
- REPORTS folder contains reports generated by Android studio about the coding coverage.