ECM2425 Mobile and Ubiquitous Computing

660050748

Continuous Assessment

Report

Introduction

The following report will outline to the reader the low- and high-level functionality of the Android Weather Forecast App and how they map to the given specifications, the system design choices and justifications behind them, as well as A UML diagram describing the organisation of classes as well as screenshots of the app may be found in the appendix. The report will be starting at a high-level view of the app before diving down into the low-level character.

High-Level

Main Activity

The Weather Forecast app is an Android application which provides the user with weather forecast information for the current day as well as the five following days. The weather forecast for the current day is displayed in a thorough way, including the day of the week, the date, an image depicting the current conditions, the city, the temperature, and a description of the weather situation, e.g. Light Rain. This provides the user with a brief overview of the current weather conditions without forcing them to navigate through the application too much. Weather for five future days is displayed in alternately coloured rows, giving a pleasant sketch of what the following days will look like. The data displayed in the rows is kept to an absolute minimum and only includes the abbreviated day of the week, the day of the month, an image showing the conditions, and the highest temperature for that day. In case the screen is too small to fit all of the data, the user has the ability to scroll down the activity to view any information which could not be displayed.

In the top right corner, the user is provided with an option menu with alternatives to change the location, change the unit of temperature, and refresh the data. Changing the location is an important aspect of this app due to the mobility of mobile devices. As users move around to different cities, countries, and continents, they want to view the weather for their specific location to be able to plan their schedule. More on location in the high-level location section. Although the overwhelming majority of countries have adopted the metric system as their standard, having the possibility to switch between different units of temperature is a neat addition, especially when travelling to countries such as the U.S. Last, there is an option to refresh the data which will make another call to the openweather API to get the most recent forecast data. Though data is refreshed every time the activity is created or restarted, users like to exercise control over certain features such as getting the data when they feel like it. Giving them the option to do so would thus lead to a more interactive and immersive experience.

Change Location Activity

The importance of changing location has already been allured to, so the subsequent part will be relatively brief. In the Weather Forecast App, users can choose between the following eleven pre-set locations:

- Amsterdam
- Belfast
- Dublin
- Berlin
- Cardiff
- Copenhagen
- Exeter
- London (default)
- Paris
- Roma
- Stockholm

The decision behind providing more than just one or two places for the user to select was partially fuelled by catering to the curiosity of the user (people like to click around on buttons and whatnot to explore), however also to provide a scalable base from which more cities can easily be added (more on this in the low-level aspect. Once the user has selected their option and successfully navigated back to the main activity, the data for the chosen city will be loaded via another API call and replace the current forecast. However, in the event where no network connection is available, the user will only be able to choose from cities which have had their forecast information loaded prior at a point in time where a network connection was available, and data could be retrieved. By only presenting a sub set of locations, the user will not be confused by which locations are available and which are not.

Detailed Weather Activity

When looking at the forecast for the coming, users will want to get more precise information about the conditions which will exist on that specific day and at a specific time of day. This demand is filled by a further activity providing the most detailed view of weather data in the Weather Forecast application. It can be reached by clicking on either of the rows mentioned earlier or the detailed version for the current weather. The data displayed maintains similarities with the detailed description on the main activity, however with a few tweaks. In addition to the information already mentioned in the main activity section, the detailed view displays data for wind, clouds, humidity, and the time of day for the particular forecast. The reason for only displaying these three specific measurements has for a simple reason. The user should not be overwhelmed by a mountain of information and only be given what they need. Wind, clouds, and humidity are values which actually have the potential of influencing decisions made with regard to day to day activities. For example, in the case of extreme wind speeds, a person might take public transport to work instead of their bike. If it is very cloudy, they may want to bring a jumper with them, and on days with high humidity a stick of deodorant. Another feature of this activity is the ability to switch between different times of the day and show the forecast for each of them. This further enables users to directly see what the weather is going to be like at a given point on this specific day.

Switch Temperature Unit Activity

As already mentioned, most of the world uses the centigrade scale to measure temperature. Nonetheless, giving more options for customisation is seldom wrong and also speaks to a larger pool of user types. E.g. an U.S. American exchange student is most familiar with Fahrenheit and will thus want her weather displayed like shew is used to. Likewise, an EU student will want to use Celsius as it is the system in use here. An impassioned physics student on the other hand may well prefer his temperature to shown in degrees Kelvin. Switching units will change the temperature in all activities to maintain a consistent picture.

Low-Level

The following section will contain low-level descriptions of and justifications for the different design choices made with regards to layouts, data structures, objects, classes and their relations, and low-level implementations of features such as data storage.

Layout

Activities were exclusively implemented using vertical and horizontal linear layouts due to their familiarity and simplicity. The main activity and detailed weather activity both rely on different fragments to function. Using fragments instead of duplicating the same XML code has multiple benefits. First, the absence of code repetition decreases the likelihood of bugs while increasing maintainability, scalability, reusability, and simplicity of the entire layout. This in turn means that changes can easily be implemented by only making them in one place instead of many. Second, although it was not attempted in this application, utilising fragments has the benefit of more easily conforming to different screen sizes and configurations. For example, on a smartphone fragments A and B are in activities C and D, whereas on a tablet with a larger screen size, both fragments A and B can be in activity C. Though this requires more XML files and Java classes specifying these fragments, the payoff is plentiful.

Data Storage

The Weather Forecast Application uses two ways of storing data externally, namely Shared Preferences and SQLite. The reason for using a combination of both and not sticking with either for data being written to non-volatile storage is the range of different situations within the application that need to be addressed. A blanket approach of using SQLite for any kind of data, even minute pieces, would do more good than harm. The same goes for shared preferences. Both approaches have their drawbacks and advantages for specific circumstances. While SQLite is excellent for structured bulky data, Shared Preferences with its key-value pair approach shines in cases where simple primitive values need to be stored. Hence, the Weather Forecast Application uses SQLite to store forecast data and Shared Preferences for minor data such as the currently selected unit of temperature.

The SQLite database consists out of two tables, one for cities and one for the forecast data. The cities table is what makes possible the decent sized set of locations for the user to choose from. Upon starting up the app for the first time, the `assets/cities.json` file containing the information for all the offered cities will be read and have its contents placed in the cities table. This data is then presented to the user in the location activity and is a scalable approach as the only measure that needs to be taken to add more cities is to place their data in the `assets/cities.json` file. The weather relation contains information for each three-hour interval of weather. This table is written to every time a new API call is made and JSON data is retrieved from openweather. Any previous data stored for the same city will be overwritten by the new data to keep the storage space small and the computations as fast as possible. Data is only ever read from the weather table when there is no network connection available. In that case, the cities and weather table will be joined together on the id of the city to allow the user to only choose those cities which have data stored in the database.

Data Structures (See UML Diagram in the Appendix)

The JSON data delivered by the API has many pieces of information, e.g. the wind, rain, cloudiness, time of day, humidity, pressure etc. All of these values could be stored as primitives in a single Java class, but that would not use encapsulation and data abstraction to its full potential. Instead, the structure used for this application is to provide each of these values with their own class. First, this is a separation of concerns as now the primitive variables for clouds and wind are stored in their respective objects. Second, it allows for more detailed manipulation of the values and decouples them from one another.

All these different objects describing an aspect of the forecast are brought together in an HourlyWeather object. This object is an important piece as it groups together all of the data for a single forecast, providing a single access point to it. The encapsulation does not stop there however. When forecast information is received, it can be divided into three different logical parts: the entire forecast, the daily forecast, and the hourly forecast. The hourly forecast already has its data structure, so what about the daily forecast and the entire forecast? For each day there usually six different hourly forecasts made, one every three hours. They are stored in a Map in the DailyWeather object with their time of day as a key and the HourlyWeather object as the value. Now, the different HourlyWeather objects are easily accessible through a DailyWeather object. To complete the hierarchy of encapsulation, all of the different DailyWeather objects are placed in a Map in the Forecast class with the week days as keys and DailyWeather objects as values. This allows us to access any forecast data through a single object, greatly simplifying the program. Enumerations are also heavily utilised for things such as the unit of temperature, day of the week and hourly interval.

Non-UI Thread Computation

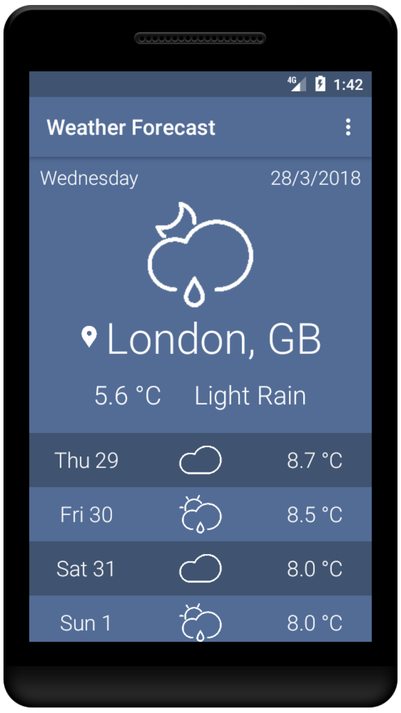
When performing extensive operations which might take an indefinite amount of time to complete, these should be performed in a background thread to avoid blocking the user interface and therefore slowing down the app. Furthermore, it is essential to decouple that background thread from the activity it is running in so that it can still deliver results even though the activity has been destroyed and rebuilt as a result of screen rotation for example. This goal may be achieved using an AsyncTaskLoader, which can asynchronously perform background operations and return the result even though the activity which started it may not exist anymore. The Weather Forecast Application uses an AsyncTaskLoader to make the API call to openweather and return the result to the main activity where the JSON String is passed to the Forecast class to be turned into HourlyWeather, and DailyWeather forecast objects.

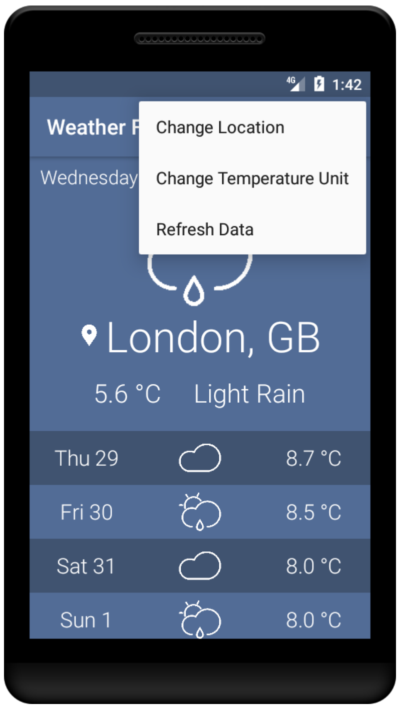
General Class Descriptions

- **MainActivity**
  The main activity of the application.
- **LocationActivity**
  Activity to choose location.
- **TemperatureUnitActivity**
  Activity to choose temperature unit.
- **DetailedForecastActivity**
  Activity to give detailed forecast description.
- **Database**
  Performs database computations.
- **SharedPreferences**
  Performs shared preferences operations.
- **NetworkUtil**
  Performs network related operations.
- **ForecastRowFragment**
  Fragment for each row of the forecast.
- **MainDetailedForecastFragment**
  Fragment for detailed representation of weather.

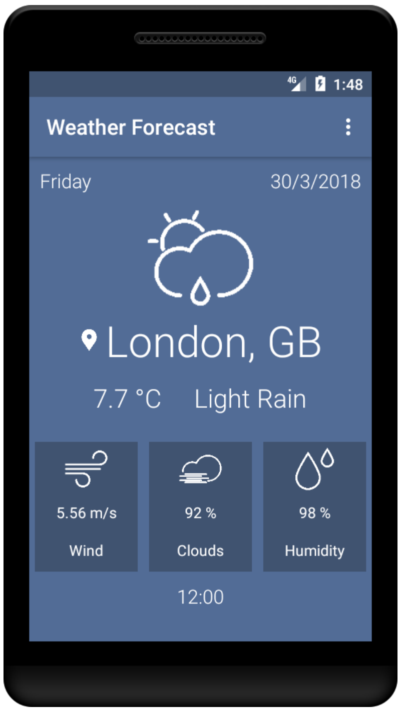
Appendix

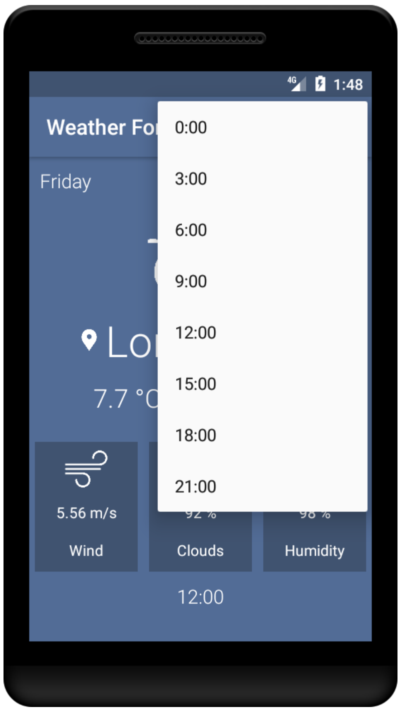
Main Activity



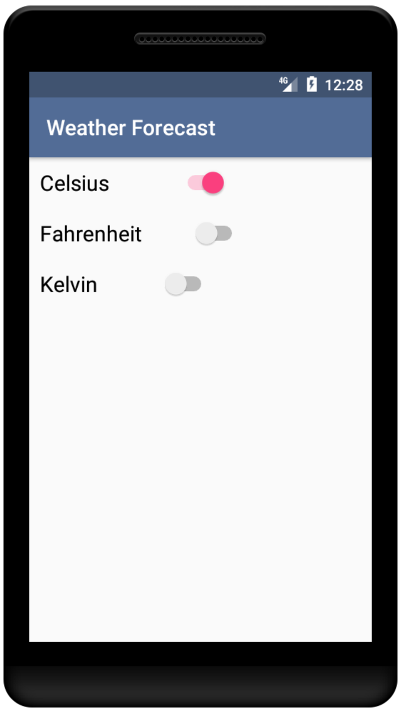


Detailed Forecast Activity





Temperature Unit Activity & Location Activity



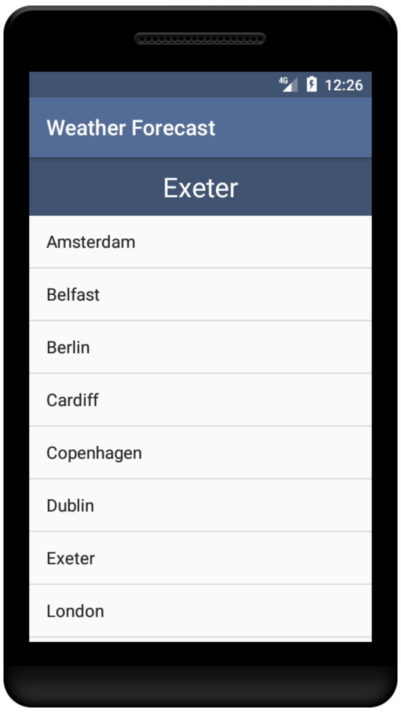
12:28

Weather Forecast

Celsius ●

Fahrenheit

Kelvin



12:26

Weather Forecast

Exeter

Amsterdam

Belfast

Berlin

Cardiff

Copenhagen

Dublin

Exeter

London