1 Technical overview

Our project made use of the material discussed in the lectures, and our group chose to use the coding language of Java and Kotlin.

For the networking portion of the project, we made use of Retrofit, which was taught during the lectures. We also referred to the starter project that was shared by one of the speakers to reference how to complete the networking. We decided to utilize Retrofit and the GsonConverter factory as these tools are most suitable for getting data from the government API in the JSON format.

For the UI portion of the project, we made use of different types of views such as TextView, RecyclerView, CardView and ImageView, as well as different types of layouts such as LinearLayout and ConstraintLayouts, some of which were taught during the lectures. We also referred to online documentation on the Android syntax and looked at examples of how to implement them, as well as what attributes or properties we could use within them.

For libraries, it was imported automatically by Android Studio as we typed our code, so it was really convenient. One great library that we used was HashMap, imported from the Java API, for our icons that were linked to the output of the government API that was retrieved by us. For example, if the output was “cloudy”, our application will show a cloud icon. We chose HashMap for this implementation as the retrieval of icons using key-value pairs would be much faster as compared to using switch or if-else statements.

A brief summary of how our code works:

ApiInterface.java: used in conjunction with the ApiData class

ApiData.java: the API provided was divided into two parts, area_metadata and items as ForecastItem. Together, they make up the attributes of our class. There are various other subclasses in this class which are necessary for Retrofit to convert JSON to Java class.

AreaData.java: A class that takes in all the necessary information from ApiData.java like the name, latitude, longitude and forecast which will be used in the MainActivity.java.

MainActivity.java: There are two main features in this activity, location tracking using getLocation() and fetching data from Web API using sync().

> **getLocation()**: We use GMS (Google Maps Services) to provide us an entry point for interacting with the fused location provider. We then used Android’s Geocoder class to provide us with the system’s current latitude and longitude. During runtime, the app will prompt the user to give location permission, and both cases of permission granted and rejected are then handled.

sync(): gets API data from api.data.gov.sg/v1/environment/2-hour-weather-forecast and converts the data from JSON format to ApiData.class. ApiData is then converted to AreaData objects which are then added to areaDataList to ease the retrieval of data. From there, go through the list of AreaData objects one by one to find which Area has the smallest distance to our current location.

```
// index of nearest area in List<AreaData>
int nearestAreaIndex = 0;

// function to calculate squared distance
class Helper {
    public double getSquaredDistanceFromIndex(int index) {
        // scale to prevent rounding-off
        double scale = 1000;
        return Math.pow((areaDataList.get(index).getLatitude() * scale - currLat * scale), 2) +
                Math.pow((areaDataList.get(index).getLongitude() * scale - currLong * scale), 2);
    }
}

// find the nearest area by looping through the list
Helper helper = new Helper();
double minSquaredDistance = helper.getSquaredDistanceFromIndex(0);
for (int i = 1; i < numOfAreas; i++) {
    double currSquaredDistance = helper.getSquaredDistanceFromIndex(i);
    if (currSquaredDistance < minSquaredDistance) {
        nearestAreaIndex = i;
    }
}
```

Snippets of code to find the nearest area to system's current coordinates

MainActivityTest.java: comprises two methods used for unit testing. The first method uses the Mockito library and is used to test whether the getLocation() method in MainActivity successfully updates a non-null value for currLat and currLong. The second method is used to determine the validity of these values.

IconHashMap.java: icons (emoji and .png images) are mapped to their respective weather and are presented on the app itself as an icon. The class has 2 attributes: iconHashMap and imageHashMap.

iconHashMap: used to link forecast (String) to weather emojis (String)

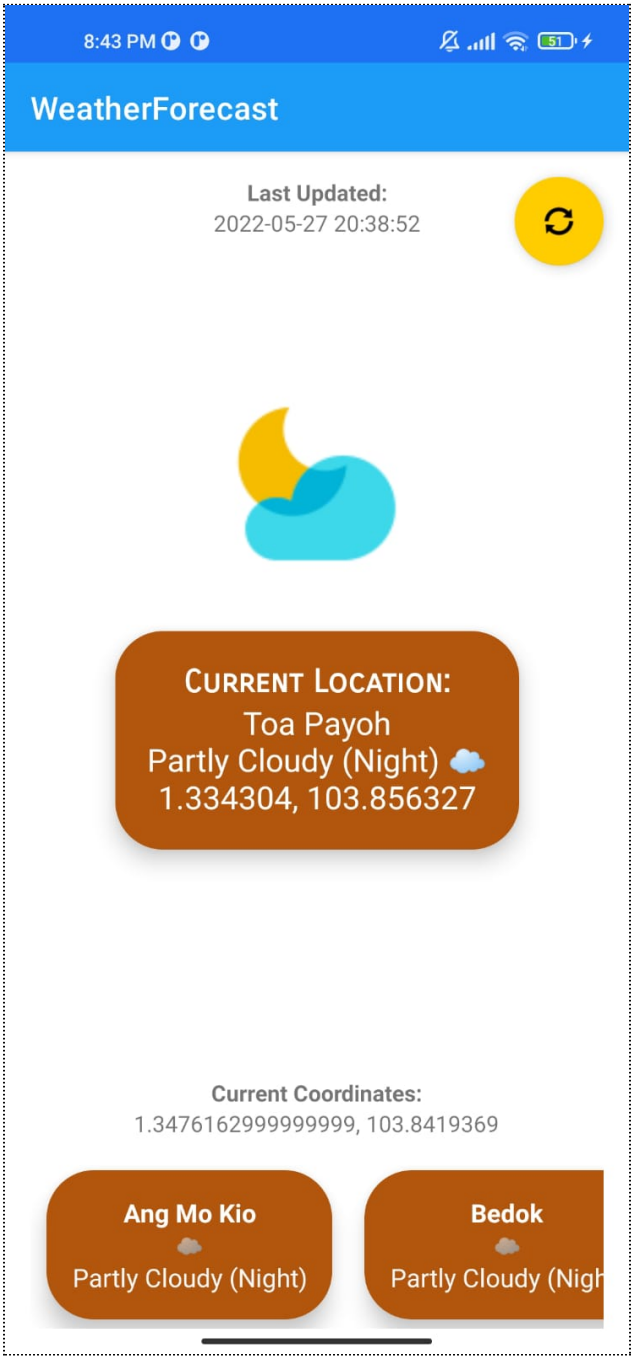
imageHashMap: used to link forecast to .png images in the res/mipmap folder

IconHashMapTest.java: used to test if the emojis match the forecast correctly.

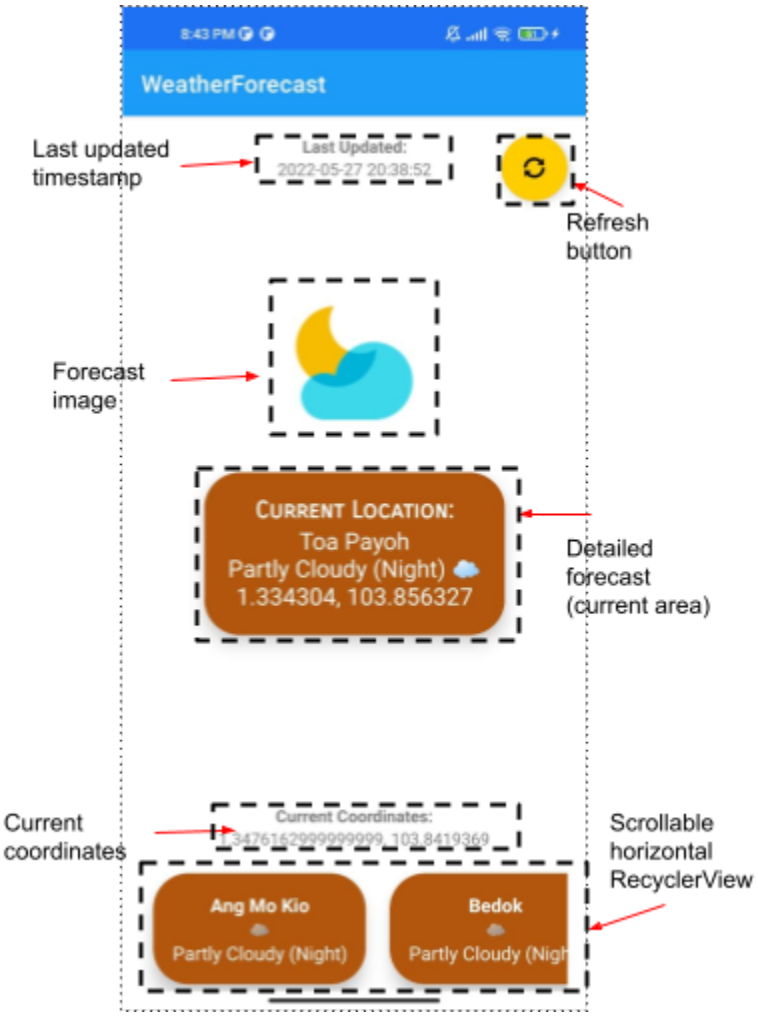
AreasForecastRecyclerAdapter.kt: an instance of this RecyclerView adapter class is initialized when sync() is called in MainActivity for the horizontal scrollable list at the bottom.

AreasForecastViewHolder.kt: binds RecyclerView to the CardView template provided in the folder layout/item_basic_forecast.xml

2 Product overview



Our mobile application retrieves data from an open government API that provides 2-hourly updates for weather forecasts in various locations of Singapore, such as Bedok, Hougang and Pioneer, and displays the appropriate weather forecast data for the region nearest to the user's current geolocation.



One of the features of the application includes weather icons that indicate visually whether it is sunny, rainy, or cloudy, using images and emojis. These features help users to easily and conveniently get an idea of the upcoming weather without even needing to read through the weather forecast text.

Another feature of the application includes the detailed weather forecast, which lists the weather conditions and temperature in text. This helps users to get a more fine-grained understanding of the upcoming weather conditions in their area.

One other feature of our application includes displaying the latitude and longitude of the location that the user is currently at. This gives a more accurate depiction of the user's location, as compared to the region he or she is in.

Another feature is that our application displays a horizontal scrollable list of the basic weather forecast conditions of all areas in Singapore at the bottom of the screen, so that users can easily view the weather forecasts for other regions in Singapore should they be traveling there.

Lastly, our app also has a refresh button that users can click on to retrieve the latest updated data. Along with that, the refreshed view will indicate the latest retrieval of the weather data by displaying a timestamp of the last update, which shows beside the refresh button.

3 Highlights

The most interesting part of the application was designing the UI. This was because our code was converted into actual visible components, and it was very rewarding to be able to see what the output of the code as design was. It was also interesting to work on different views to enable different behaviors within one screen, such as a scrolling behavior through a RecyclerView. It was also fascinating to see how adding text in terms of code could convert visual elements to look nicer and more aesthetically pleasing. Furthermore, Android Studios includes a drag and drop feature to customize the UI, We found this extremely helpful in learning the format of coding the UI and also made the experience fun.

The most challenging part of the application was retrieving the user's current location. This was because we were unsure of how to access data that was related to the user's mobile device. To solve this problem, we looked through tutorials online as well as searched for answers to any bugs that we faced. In the end, we successfully made use of the LocationServices, FusedLocationProviderClient, Geocoder and Location packages to retrieve the user's current location.

The most time-consuming part of developing this application was creating the ApiData object class. The Web API in JSON format is converted into an instance of this class using Retrofit2 and GsonConverter. However, the nature of JSON is that oftentimes there are nested objects deep in the main curly braces. In order to properly convert these nested objects into Java classes, we had to code these nested objects in Java. Since these nested objects are inconsistent and do not have a set format, it is particularly time consuming to convert this format to the nested classes format in ApiData.java.

4 Project management

At first, our group was unsure of how to split the different parts of the group project as we felt that all of the components of the mobile application were very connected to each other and could not be developed separately. However, after a group discussion, we came to the conclusion that we could split tasks between logic and design. This allowed each group to proceed with development without waiting on the other. Hence, the tasks were split under the main categories of logic and design and then assigned to different group members.

Tasks were split between the data retrieval, test cases and the designing of the UI. The group was split into half, with each half being assigned to different tasks. Specifically, 3 group members were assigned to data retrieval and test cases. Three other group members were assigned to design the UI. We assigned the tasks based on interest and as well as each of our strengths.

Along the way, the task delegation and work done changed from our simple split of logic for 3 members and UI for 3 members. As our group was not familiar with Android development with the exception of one group member, that person who was originally assigned to the logic group helped to draft a skeleton of the project and created the Github repository for us to work on. This member also worked on retrieving the API data and converting it into usable data within classes.

Another person that was originally assigned to the logic group designed the unit tests for our application. The last member that was assigned to the logic group ended up working on the RecyclerView and some part of the UI design, as the members assigned to the UI group required extra manpower.

Before we started the project, we agreed on what data is to be collected (such as forecast and location) and how it will be formatted. We also agreed on the app layout and the different UI components to be worked on. Deadlines were set for the project deliverables with buffer time allocated for us to do a close review of the application. All group members were assigned to write the group report, as our group felt that different members could use their experience to best contribute their insights.

All in all, this group project made our group realize that it was not that easy to delegate work and in projects, change is less unusual that status quo. We also understood the importance of helping each other out, so that members with less experience would not feel like they were left out, and that everyone could learn from each other.