CHAPTER 2. LITERATURE REVIEW

2.1 Weather Forecasting

Throughout the past century, one of the most challenging scientific and technological problems has been forecasting the weather. Weather forecasting systems are one of the most complex equation systems that a computer must solve. Weather is a continuous, dynamic, multi-dimensional chaotic process, and data-intensive and these properties make weather forecasting a stimulating challenge [1]. It is one of the most important and difficult operational duties that numerous meteorological services around the world are required to carry out. Weather forecasting can be categorized into six different groups according to the length of the forecasting period:

1. Now-casting, current weather variables and 0-6 hour's description of forecasted weather variables.
2. Very short range weather forecasting, up to 12 hours description of weather variables
3. Short range weather forecasting, for a period of 12 hours to 72 hours.
4. Medium range weather forecasting, for periods of 3 to 10 days.
5. Extended range weather forecasting, for periods of 10 to 30 days.
6. Long range weather forecasting, the long range weather forecasts are issued thrice in year. Validity period of long range weather forecast is 10 to 30 days [2].

Weather forecasters utilize a variety of tools like barometers to measure air pressure, radar to locate and track clouds, thermometers to measure temperature, and computer models to process the information these tools' data generate.

2.2 Android

Android is an open-source operating system that runs on the linux kernel. Java IDEs and android java libraries are both supported, giving developers a flexible platform for creating mobile applications for Android. The android operating system gained popularity among developers for its customizable nature [3]. Building an application in one platform and deploying it in multiple platforms at once and doesn't require any additional changes to be made is very effective. Furthermore, software developers can easily modify and incorporate improved features to meet the latest standards for mobile technology. Android Operating System is mainly divided into four main layers: the kernel, libraries, application framework and applications.

2.2.1 Kernel

Linux Kernel (Linux 2.6) is at the bottom layer of the software stack [4]. The entire Android operating system is constructed on top of this layer, with some modifications made by Google. Similar to the main operating system, it offers the following functionalities: process management, memory management, and device management (for example, camera, keypad, display, etc.). Additionally, it also works on network management and security systems.

2.2.2 Native Libraries Layer

On the top of the Linux Kernel layer is Android's native libraries. The device can handle different types of data thanks to this layer. Data is hardware-specific. These libraries are all written in the languages C or C++. These libraries are called through java interface. Some important native libraries are: Surface Manager, SQLite, WebKit, Media framework, Free Type and libc [4]. A set of core libraries and a Java virtual machine (Dalvik virtual machine) that have been modified and optimized by Google to work with the Android platform are included in the Android Runtime.

2.2.3 Application Framework Layer

This layer is designed to allow developers getting access to the core application services. Developers may modify the system architecture of their applications in this layer so they can use the various services offered by the API libraries. These are the blocks with which developer's applications directly interact. Important blocks of Application framework are: Activity Manager, Content Providers, Telephony Manager, Location Manager and Resource

Manager [4].

2.2.4 Application Layer

The top layer in the Android architecture is called the Applications Layer. Every device contains a few preinstalled applications, including a dialer, web browser, contact manager, and an SMS client app. A developer can create his own application and replace it into an existing one.

2.3 JavaScript Object Notation (JSON)

JSON is intended to be a data exchange language that is both computer- and human-readable. It is a lightweight format based on the data types of the JavaScript programming language [5]. JSON is directly supported inside JavaScript and is best suited for JavaScript applications [6]. JSON documents are dictionaries with key-value pairs in which the value may also be another JSON document, allowing for any number of levels of nesting. Along with atomic types like strings and numbers, JSON also supports arrays. The format is now fully compositional as it is possible for arrays and dictionaries to hold any JSON document. JSON is quickly rising to the top of the list of the most widely used formats for exchanging data on the Web due to its simplicity and easily read by both humans and machines. This is especially clear when Web services use Application Programming Interfaces (APIs) to communicate with their users considering JSON is currently the format of choice for sending API requests and responses over the HTTP protocol.

2.4 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA. On May 16, 2013, Android Studio was introduced as the official IDE for creating Android apps at the Google I/O conference. It began its early access preview with version 0.1 in May 2013. Beginning with version 1.0, the first stable built version was made available in December 2014. Kotlin has been Google's preferred language for creating Android applications since May 7th.

Besides this, Android Studio also supports other programming languages. Android Studio offers even more features that enhance productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where developer can develop for all Android devices
- Apply Changes to push code and resource changes to running app without restarting app
- Code templates and GitHub integration to help developer build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

2.4.1 Thread and Handler

Android automatically creates the "main" thread, the first thread of execution, when an application is launched. The main thread is in charge of communicating with elements from the Android UI toolkit and dispatching events to the appropriate user interface widgets. Common examples of operations that should be avoided in the main thread include network operations, database calls, and loading specific components. They are called synchronously when they are made in the main thread, which means that the UI won't respond at all until the operation is finished. They are typically carried out in separate threads due to preventing UI blocking while they are being carried out. A thread is the direction a programme takes while being executed. The Java virtual machine allows an application to run multiple threads simultaneously. As Android is a single-threaded model, we need to create different threads to perform our task and post the result to the main thread where the UI gets updated. All operations, with the exception of updating UI elements, are allowed inside threads.

A Handler is a component that can be connected to a thread and instructed to take some action via runnable tasks or simple messages [7]. It collaborates with a different component, Looper, which manages message processing for a specific thread. When a handler is created, it can get a Looper object in the constructor, which indicates which thread the handler is attached to.

Although Android offers a variety of thread management and handling options, none of them are perfect. Depending on the use case, picking the appropriate threading strategy can make a big difference in how simple it is to implement and comprehend the overall solution. Although not in every situation, the native components work well. Similar things apply to the third-party solutions.

2.5 Application Programming Interface (API)

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications or systems to communicate with each other [8]. APIs allow developers to use the services provided by an application or system without needing to know the internal details of the application. APIs help in integrating applications easily, increase flexibility, and speed up application development. API architecture is typically described in terms of client and server [9]. Applications that transmit requests and responses are referred to as clients and servers, respectively. In the weather example, the mobile app is the client and the bureau's weather database are the server.

APIs come in different types, such as REST APIs, SOAP APIs, and RPC APIs. REST APIs are the most common type of API, which stands for Representational State Transfer. REST APIs use HTTP to transfer data between client and server. SOAP APIs, on the other hand, use XML to transfer data and are less popular than REST APIs. RPC APIs invoke executable actions or processes. RPC can employ two different languages, JSON and XML, for coding; these APIs are dubbed JSON-RPC and XML-RPC, respectively [10].