Hung Minh Doan

Developing a Map Application Utilizing Collected Environmental Data

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

6 March, 2019

Author Title Number of Pages Date	Hung Minh Doan Developing a Map Application Utilizing Collected Environmental Data 40 6 March 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart System
Instructors	Kimmo Sauren, Instructor

This thesis aimed to develop a mapping application and website which utilized the environmental data (temperature, humidity, light level). The Android Application was developed by Java in Android Studio and the website was written in HTML. Just the same as most of map applications, the map application shown information about the cities, roads, buildings and the cur-rent location of user using GPS data. Moreover, the application could connect with a sensor device called SensorTag by Bluetooth low energy protocol. SensorTag would collect information of surrounding temperature, humidity and light level, the application then gathered the data from the sensors and shown the environmental information together with the user's current location on the map.

Furthermore, the environmental data and the exact location of the user collected by the application were sent to a server API, which was written by JavaScript and hosted on Heroku cloud service. The web server gathered all the weather and locations data from every using application from the API and shown the information on a big map on the website. Therefore, the more users of the application, the more weather information of different locations is available on the web server. This means that a user who is using the mapping application helps users in other locations to be aware of weather condition of his/her location, and by accessing the web server, user is able to see the weather information of different places.

After a considerable studying about different cloud services, Java and JavaScript programming language, the thesis has achieved the initial aim by successfully developing the Android application, the weather map website and the server API. In addition, plenty of value knowledge about cloud services was also obtained. User could observe weather condition at current location by using the Android application and received latest updates about weather information in another locations by accessing the website.

Keywords	IoT, Application, Map, Environmental Data, Weather Information

Contents

List of Abbreviations

BLE	Bluetooth Low Energy
API	Application Programming Interface. A set of routines, communication protocols and tool for developing software applications.
SDK	Software Development Kit. Providing tools, libraries, documentation, code samples and guides for software applications on a specific platform.

1 Introduction

From the earliest time, maps were used to show geographical information of surrounding area. Maps are not limited by any cultures nor language barriers so that people can use maps to study about any area on the Earth. Nowadays, with the creation of choropleth maps and many other types of maps, maps are not only used in geography but also take an important part in many social topics such as human, politics, environment, employment, etc. For example, resource maps indicate natural resources of a location which is helpful in economic geography, political maps showing boundaries and borders between countries are useful to politician and international relations study. Another explanation for the value of maps is that seeing maps of a location in different periods of time, geographers can study about changes in topography, infrastructure, population and economy of that location more simply [1].

Moreover, with the rapid development in Internet of Things (IoT) technology, besides the Internet, after Bluetooth Low Energy (BLE) was introduction in Bluetooth 4.0 as a part of its specification, it has been easier for physical devices like sensors now to connect and communicate to a host (smart phone, tablet, laptop, etc.), the power consumption is also improved and not as high as that in standard Bluetooth [2, part 1]. Sensors, which are directly or indirectly connected to IoT networks, collect data from the sur-rounding environment and convert into signals. The controller of the system then uses these signals to process and takes proper actions.

This project aims to develop an Android map application and a weather map of the Earth on a website. Another part of the project is an Android application, which shows weather information of its user's current location by connecting with a sensor device. The information about temperature, humidity and light level from many users in different location will be collected and shown on a world map. By accessing the website, user can observe currently updated weather information of different areas in the world. Furthermore, the database can keep track of weather information sent from a specific host which would bring plenty of usefulness, for example, people who have issues about respiratory or heart, the owner of the building can use the weather data from these people to control the warming system more properly. That is one of in numerous potential utilizations the project might offer.

2 Theoretical Background

2.1 Definition and types of maps

Maps have been used by human since ancient time. Maps use various symbols to represent characteristic of a place, they also show sizes and shapes of countries and continents. Besides, every map has a scale to express ratio between the distances on the map and the real distances in the real world, based on the scale, people can calculate distances between locations on the Earth. Cartographers (map makers) use symbols to show geographic factors on the map for instance capital city of a country, various kinds of lines to indicate boundaries, roads or highways and various colors to represent forest, deserts or water.

There are two general types of maps: general reference maps and thematic maps. General reference maps are the maps that we mostly use, they show information about continents, cities, boundaries, roads, mountains, rivers and coastlines, another kind of reference map is topographic map, which shows changes in elevation. On the other hand, thematic maps display the distributions of different factors over Earth's surface. The theme of the distributions is usually about people, other organisms or land, for instance population, average income, spoken languages, agricultural production or average rainfall.

Reference maps are usually made by government agencies such as United State Geological Survey. They are helpful to everyone from people finding a way to a location, hikers trying to choose a route to walk to engineers trying to figure out where to build highway or dams because all the hills and valleys in an area are showed in the reference maps.

Thematic maps are conducted with the support of geographic information system (GIS) technology. GIS is a computer system which stores all the data about Earth's surface, people, climate, houses, businesses, etc. The system combines all these data and show them on the thematic map. Governments use GIS technology and thematic maps to analyze and make important decision for instance which area in the country has the risk of being polluted, where to build a new shopping mall. [3]

2.2 Overview of sensor

Nowadays, sensors are pervasive. They play an important role in countless devices such as automobiles, airplanes, phones, radios, chemical and industrial plants, etc. An oxygen sensor controls the gasoline emission ratio in a car, motion sensors controls light system and automatic doors in building, etc. There are plenty of applications for sensors that human will never be aware. The sensors collect the data and send to the computer, which is the brain of the system, then the computer controls the systems properly thanked to the data from the sensors by comparing that data with the pre-designed values.

In the early 1800s, scientists had taken the advantage of temperature sensitivity of electrical resistance to make temperature measuring device, the story of sensor began from a long time ago. But it was not until 1975, the American National Standard Institute (ANSI) indicated that “sensor” is a device that outputs a valid value from a specific physical measurement. At that time, based on the standard of ANSI, the terms “sensor” and “transducer” have the same meaning. However, the scientists do not agree with ANSI about the synonyms, therefore, the term “sensor” is the more common one. Sensor acquires a physical quantity and exchanges it into a signal which is then used for processing, most of the sensors today convert measurement into an electrical signal. [4 p.10]

Table 1. Sensor Physical and Chemical Measured Quantities. Copied from [4 p.11-13]

Energy Forms	Measurands Examples
Mechanical	Length, area, volume, mass, acceleration, force, pressure, wave
Thermal	Temperature, specific heat,
Electrical	Voltage, current, charge, resistance, electric field, inductance

Magnetic	Magnetic field, permeability, flex
Radiant	Wavelength, intensity, polarization
Chemical	pH, reaction rate, oxidation/reduction potential

As can be seen from the table 1, sensor can measure various of quantities in physics and chemistry. Next, Table 2 would show specifications of a sensor that users should consider when choosing a sensor device for his/her project.

Table 2. Factors Affecting Choosing a Sensor

Factors	Characteristics
Environmental Factor	Temperature range, humidity effects, power consumption, size
Economic Factors	Cost, availability, lifetime
Sensor Characteristics	Sensitivity, range, stability, response time, error

For this project, the device of Texas Instruments SensorTag has been chosen. SensorTag is a small-size (5 x 6.7 x 1.4 cm) multi-sensor device that includes temperature sensor, humidity sensor, accelerometer, pressure sensor, light sensor, digital microphone and buzzer. BLE is also support in SensorTag. The device is sort of price (about £25) and it has a cover to protect the sensors from extreme weather or damage when being dropped. [5]

2.3 Project Overall

Before going deeper in others parts, it is better to take a look of the overall of this project.

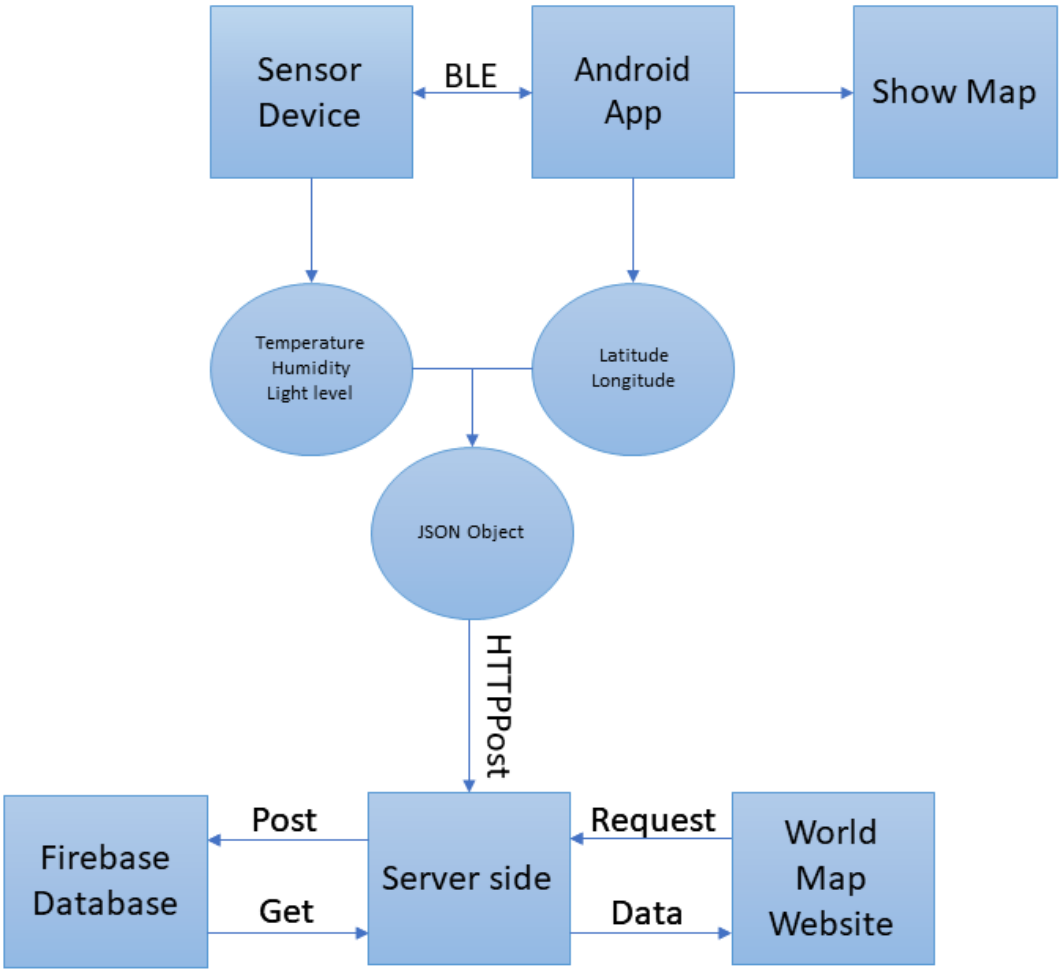


Figure 1. Project operation flow chart

As can be seen from figure 1, the Android Application and SensorTag device connect to each other via Bluetooth Low Energy. Then the weather data from SensorTag and the GPS data (latitude and longitude) from host device are merged into a JSON object, which then is pushed to the server side using HTTPPost library by the Android application. Precisely, HTTPost sends a post request to server side and the server would post the JSON object to the database. At the same time, the application shows user's current location on a map together with weather information.

The API of the server side is written in JavaScript, which contains functions to post data to database and get data from database. The API is hosted by Heroku cloud platform, one of the first cloud platforms. The database is provided by Firebase. Next sections would go deeper about Heroku and Firebase.

The website is written in HTTP, its main functions is showing all weather information from different users in different locations together in one map. The website sends data request to server side API and it is the task of the API to get data from the database.

2.4 Heroku Cloud Platform

Heroku is a cloud platform supporting application development and deployment. Heroku platform will handle hardware and servers, therefore, the developers can focus on developing their application. Literally, when an application is built on Heroku, it will be deployed to Amazon Web Services together with the entire Heroku platform. [6]

Before going deeper about Heroku, it is better to have an understanding about cloud computing and the role of Heroku platform in cloud computing.

2.4.1 Cloud computing and Heroku

Cloud computing is an information technology model that delivering services hosted on the Internet, such as computer networks, servers, storage, applications, etc. As servers and supporting services are managed by the cloud service, developers can utilize their time and effort on developing their applications. Thus, cloud computing is becoming in interesting choice for an IT businesses and projects.

As can be seen from the figure 2, cloud computing services are divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS):

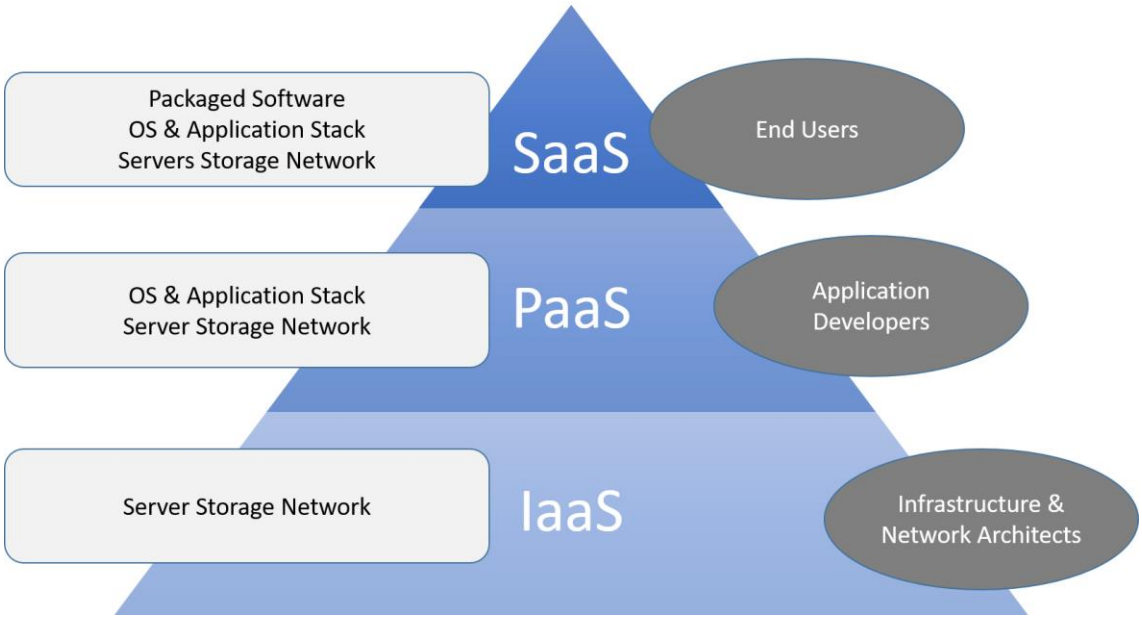


Figure 2. Cloud Computing Categories. Copied from [7]

IaaS group provides storage, networks and server resources where users can freely deploy and execute their applications. Developers can manage the operating systems, storage and their deployed applications but they cannot control the underlying architecture of the cloud service.

PaaS architectures consist of libraries, programming-language execution environment, databases, servers for websites and connectivity to other platforms. Users of PaaS can deploy applications onto the cloud service. Users have a restricted ability to configure settings of hosting environment and resources provided by PaaS service. However, users cannot manage deeper in servers, operating systems and storage.

SaaS services provides the usage of applications that developers are running on a cloud service. The applications can be accessed by many client devices through many kinds of interface, for example a website browser or a mobile application. The ability to access services for SaaS users is much more limited than that in PaaS and IaaS group. Therefore, users can only configure a few parts in application settings and they cannot control servers, operating systems, storage and resources.

Heroku is an easy-to-use Platform-as-a-Service architecture in cloud computing. Heroku offers free service for small projects but for big IT businesses, Heroku will charge a monthly fee based on how complex the project is. [8]

2.4.2 Heroku and Amazon Web Services

As mentioned above, the entire Heroku system and all applications deployed on Heroku run on Amazon Web Services (AWS), and many people might be wondering why developers do not deploy their applications directly to AWS and get rid of Heroku. The first reason is that AWS is an IaaS provider, in IT businesses, IaaS providers concentrate more on being a data center and keep the applications running than working with the developers.

On the other, Heroku is a PaaS provider, which is on the top of IaaS architecture in figure 3. IaaS services bring developers an environment to design and developers their application easier, to keep their application running on server, users only make a few simple commands on PaaS provider (Heroku), and IaaS provider (AWS) will do the job. [6]

2.4.3 Benefits of Heroku

This section will indicate reasons why Heroku is one of the most cloud platforms for deploying and running applications.

Heroku handles many programming languages and solutions

Heroku is one of the first cloud platforms, and when it was firstly introduced in June 2007, Heroku only supported Ruby programming language. Until now, Heroku supports plenty of programming languages that are mostly used for website application deployment model, such as Node.js, PHP, Go, Java, Scala, Clojure, Python, etc. However, for different programming languages, Heroku provides the same manner to build, run and deploy applications. For this reason, Heroku is called ”polyglot platforms” which makes it a preferred cloud platform amongst developers.

As a result, various developers from different programming languages can choose Heroku for a cheap and convenient cloud platform to develop and deploy application. [6]

Dynos brings easier development and integration with GitHub

Heroku manages its applications by container model, and the containers used in Heroku are called ”dynos”. Dynos are designed based on Linux containers, which execute code based on user command. Developers can scale their application to a suitable number of dynos based on application resources, and Heroku also offers the container management feature to support developers to scale and manage the number, size and type of dynos their application are using. Easy scaling is another reason why working with Heroku is so conducive. [9]

GitHub is a famous and reliable platform to save source code, thus, Heroku works together with GitHub to optimize developers' efforts and money on development of their projects. Projects deployed in Heroku are integrated with GitHub so that they can be built and deployed with the latest version of code. [8]

2.4.4 Heroku in comparing with other PaaS cloud platforms

This section will compare about the price and supported operating systems of Heroku and other famous PaaS platforms.

Table 3. Supported operating System of Heroku and other PaaS platforms. Copied from [8]

OS	PaaS Services								
	Heroku	Co-caine	Digital-Ocean	Cloud Foundry	Blue-mix	Elastic Bean-stalk	App Engine	OpenShift	Mi-crosoft Azure
Linux	✓	✓	✓	✓	✓	✓	✓	✓	✓
Win-dows	✗	✗	✗	✓	✗	✓	✓	✗	✓

Ma-cOS	✗	✗	✗	✓	✗	✗	✗	✗	✗

As can be seen from table 3, Linux OS is supported by all PaaS platforms it is a popular, cheap and easy-to-build operating system. On the second place, Windows OS is supported by four PaaS platforms and MacOS is at the bottom with only Cloud Foundry architecture supports it. As being shown on the table, Heroku only supports Linux and its managers are trying to develop and evolve Heroku based on Linux OS. [8]

Table 4. Price of Heroku and other PaaS platforms. Copied from [8]

Free _	PaaS Service								
	Heroku	Co-caine	Digital-Ocean	Cloud Foundry	Blue-mix	Elastic Bean-stalk	App Engine	OpenShift	Mi-crosoft Azure
Every-thing	✗	✓	✗	✗	✗	✗	✗	✗	✗
Small project	✓	✗	✗	✗	✗	✗	✓	✓	✗
Trail	✗	✗	✓	✓	✓	✓	✗	✗	✓

As can be seen from table 4, only Cocaine platform is totally free and the interesting thing is Cocaine is created by Heroku's found Andrey Sibiryov. Heroku offers free price for small projects, these projects are usually test projects. Users can test their ideas, build a general concept for their projects before getting ready to pay more for full development.

2.5 Firebase

2.5.1 Overall and history

Firebase is a platform for mobile and web application development. The platform brings plenty of tools and services for developers to develop superior applications and earn more advanced benefit. The Firebase Realtime Database, which is one of Firebase's services is cloud-hosted database, stores data in type of JSON file and connects to client applications in real time enabling client applications to receive updates with latest data automatically.

The precursor of Firebase was a startup company called Envolve, which was found in 2011. Envolve provided an API for developers to integrate online chat functionality into their applications and websites. However, developers used Envolve's API to exchange application data rather than just chatting with messages, which led the founders of Envolve, James Tamplin and Andrew Lee, to develop chat system and real-time architecture separately. As a result, in April 2012, Firebase was established as a separate company that develop backend services and real-time functionality. Later in 2014, Firebase was bought by Google and developed into a mobile and web application platform as it is today. [10]

2.5.2 Firebase Services

Firebase offers various of functionality supporting backend and server-side level stuffs, such as Realtime Database (this feature is used in this project), Storage and Hosting, Authentications, Analytics, Notifications, Crash Reporting, etc. [11]

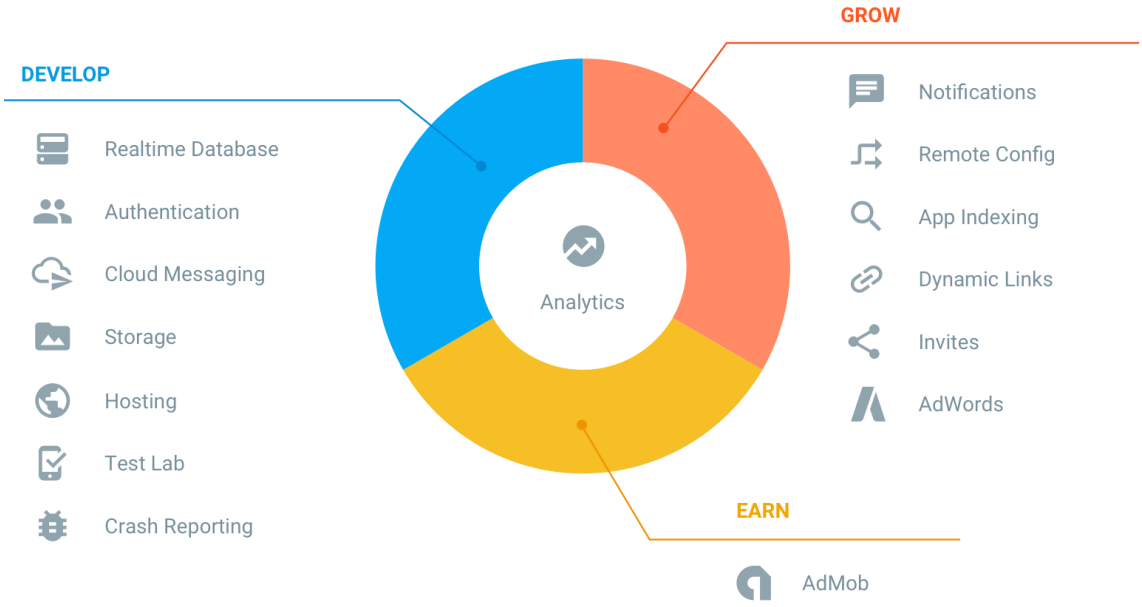


Figure 3. Firebase Services. Copied from [10]

As can be seen from figure 3, Firebase services are divided into three groups: services for developing an application, services for growing application's audience (users) and services for developers to earn money from their applications. This section would go details in almost all of the features of Firebase.

Realtime Database

Firebase Realtime Database is a cloud-hosted NoSQL database, which stores data in JSON format. The database can connect to many clients at the same time. With just a simply API, users can access their data get updates easily from any device. Realtime Database also enable other users to access the data if they have the permission, which improve collaboration between users.

Moreover, Realtime Database can be integrated in mobile and web SDKs, allowing users to build their application or website without a server. In offline mode, Realtime Database SDK stores data and changes on device's cache and when the device comes online, local data and changes are automatically synchronized with the database. Last but not least, Firebase Realtime Database can work together with Firebase Authentication to perform an authentication process. [10]

Authentication

Commonly, developers spend months to build their own authentication system, and the system needs to be maintained frequently after that. Firebase Authentication implements backend service, SDKs and user interface for users to develop authentication feature on their applications. The user interface and the SDK are instant and easy to use. Thanks to Firebase, with just about 10 lines of code, user can set up an entire authentication system, which might handle even complicated operations such as account merging.

Firebase brings plenty of authenticating methods such as email-password, phone numbers, Google account, Facebook account, Twitter account, etc. About the quality of Firebase Authentication, users do not need to worry about it because Firebase Authentication is built by the developers who created Google Sign-in, Smart Lock, and Chrome Password Manager. [10]

Cloud Messaging

Firebase Cloud Messaging (FCM) provides communication between server and devices (iOS, Android and website) by sending and receiving messages or notifications. The battery consumption for FCM feature on host devices is not much and there is a limit of 2KB for notification messages and 4KB for data messages.

Users can compose messages using predefined segments or create their own messages. Messages can be sent to several devices or single device instantly or at any future time as the user wants. The content of messages is usually about applications data or setting priorities. In addition, there is not any coding required in FCM, the feature is already fused with Firebase Analytics. [10]

Crash Reporting

Crash Reporting helps developers to detect and debug errors in their application. Firebase Crash Reporting gathers all information about crash and sends to Firebase developer console. The content of the report can be customized by developers. [11]

Remote Config

Remote Config feature supports developers to publish updates to their end users immediately. Not only updates, developers can make changes or set segments behavior in their application or website and the server would implement the changes on client's device. Therefore, no new version publishing is needed. [10]

Dynamic Links

Most website contains deep links, which take users directly to a content of the website. Firebase can convert deep links into Dynamic Links – the links that take users directly to an appropriate part in the application.

Dynamic Links feature allows developers to convert mobile websites into native applications. Developers can also make advertisement for their applications by creating links to application installation in social networks, email or SMS (users can see explicit content when they install the application). [10]

AdMob

This feature is an advertise platform of Google which let developers earn money through their application. [11]

3 Implementation of Weather Map Android Application

Weather Map is an Android Application written by Java programming language using Android Studio. The application scans for all nearby SensorTag device and connect to one of them based on user's choice. The data advertised by SensorTag and GPS data from the host device will be collected and show on a map supported by Google Map library in Android Studio. Not only showing information about current location weather, the application also converts weather and GPS data into a JSON object, that object would be pushed to the server side.

Firstly, to make the application operate properly, mandatory permissions need to be added in the Manifest file of the application.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Figure 4. Required permissions in Manifest file

As can be seen from figure 4, *android.permission.BLUETOOTH* is for general usage of Bluetooth and *android.permission.BLUETOOTH_ADMIN* is for other services of Bluetooth such as discovery mode [2, part 3]. Others permissions' function can be easily guessed by their name, they offer permission for using Internet, which is needed for sending data to the server and location permission to enable the application to detect user's current location.

The application only has one Service and one Activity, the Activity controls user interface by two fragments.

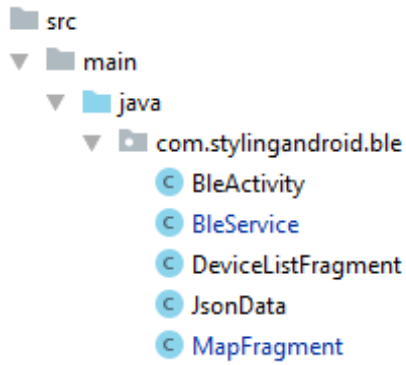


Figure 5. Activities, Services, Fragments and Classes in application

BleService does almost the jobs: scanning Sensor devices, reading the data advertised by SensorTag and doing calculation with the raw data. On another the hand, BleActivity manages the user interface by two fragments based on message received from BleService. As can be seen from figure 5, DeviceListFragment is for showing list of scanned SensorTag devices, and MapFragment indicates user's current location on a map with the weather information.

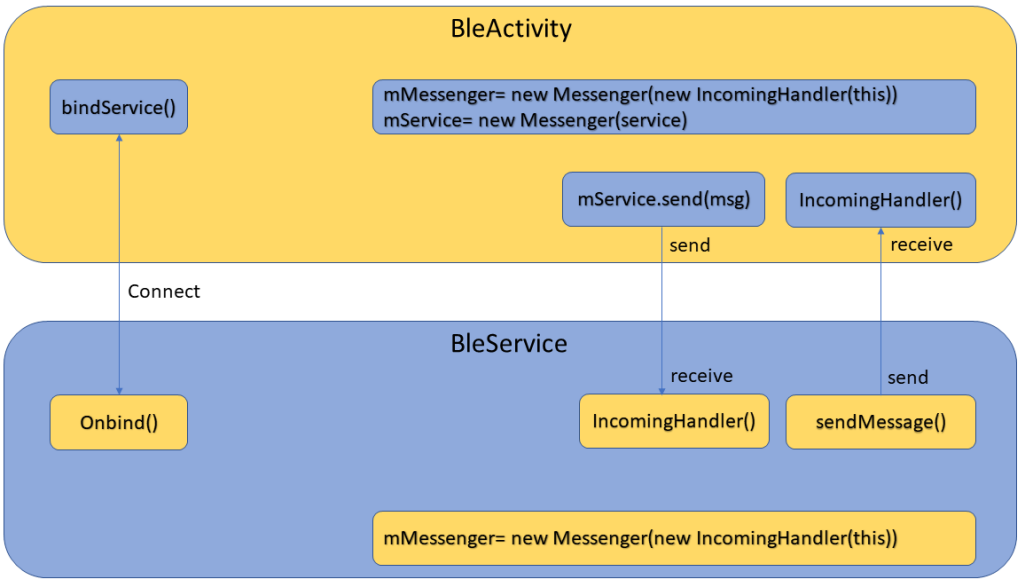


Figure 6. Messenger interfaces in BleActivity and BleService

As can be seen from figure 6, BleActivity and BleService work together by Messenger library, each of them has one Messenger to send message and another Messenger with a handler to handle received message to make correct action. An example scenario of exchaging messages betweeen BleService and BleActivity is that when any BLE device is detected, BleService will send a message containing MAC address of that device to BleActivity. After receiving the message, BleActivity shows the MAC address on user interface so users can see and choose the BLE device they want to connect. Furthermore, when user has chosen a BLE device to connect, a message will be sent to BleService. The service then makes the connection to that BLE device, reads the data and lastly, sends a message including sensor data to BleActivity. BleActivity then receives the data and shows it on user interface and hence users can see the information of temperature, humidity and light level on the map.

This section only focusses on working with BLE device: scanning and connecting to BLE device, reading and getting data from BLE device. Implementation of user interface is not covered.

3.1 Scanning SensorTag and pairing with host device

3.1.1 Scanning SensorTag

First thing to mention is that BleService executes like a state machine, thus, performs different task based on each state. BleService will start scanning BLE devices when it enters SCANNING state after receiving a MSG_START_SCAN Message from BleActivity.

```
private static class IncomingHandler extends Handler {
    private final WeakReference<BleService> mService;

    public IncomingHandler(BleService service) { mService = new WeakReference<BleService>(service); }

    @Override
    public void handleMessage(Message msg) {
        BleService service = mService.get();
        if (service != null) {
            switch (msg.what) {
                case MSG_REGISTER:
                    service.mClients.add(msg.replyTo);
                    Log.d(TAG,  msg: "Registered");
                    break;
                case MSG_UNREGISTER:
                    service.mClients.remove(msg.replyTo);
                    if (service.mState == State.CONNECTED && service.mGatt != null) {
                        service.mGatt.disconnect();
                    }
                    Log.d(TAG,  msg: "Unegistered");
                    break;
                case MSG_START_SCAN:
                    service.startScan();
                    Log.d(TAG,  msg: "Start Scan");
                    break;
```

Figure 7. BleService Message Handler

As can be seen from figure 7, the Message Handler executes function based on received messages, in this case, when MSG_START_SCAN is received, the service will call startScan() method to begin the scan.

```
private void startScan() {
    mDevices.clear();
    setState(State.SCANNING);
    if (mBluetooth == null) {
        BluetoothManager bluetoothMgr = (BluetoothManager) getSystemService(BLUETOOTH_SERVICE);
        mBluetooth = bluetoothMgr.getAdapter();
    }
    if (mBluetooth == null || !mBluetooth.isEnabled()) {
        setState(State.BLUETOOTH_OFF);
    } else {
        mHandler.postDelayed(() → {
                if (mState == State.SCANNING) {
                    mBluetooth.stopLeScan( callback: BleService.this);
                    setState(State.IDLE);
                }
        }, SCAN_PERIOD);
        mBluetooth.startLeScan( callback: this);
    }
}
```

Figure 8. startScan() method

As can be seen from figure 8, the method check weather Bluetooth is available on the host device, if it is not, the method will get an instance of the service BLUETOOTH_SERVICE, from which an instance of Bluetooth Adapter (mBluetooth) can be obtained. Bluetooth Adapter indicates the Bluetooth radio of the host device. Then another null check and isEnabled() method is called to specify that Bluetooth is switched on and accessible. If Bluetooth is ready, the service starts scanning for BLE devices, if it is not, the service changes state to BLUETOOTH_OFF. Everytime the state is changed, BleService sends a message to BleActivity. For state BLUETOOTH_OFF, the BleActivity will ask users to turn on the Bluetooth on the device. The more important thing to mention is scanning for BLE devices. BluetoothAdapter class has the method startLeScan() which literally does the scanning. However, startLeScan() method only starts the scan so that stopLeScan() method together with postDelayed() will stop the scan after a fixed period.

```
public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {
    if (device != null && !mDevices.containsValue(device) && device.getName() != null && device.getName().equals(DEVICE_NAME)) {
        mDevices.put(device.getAddress(), device);
        Message msg = Message.obtain( h: null, MSG_DEVICE_FOUND);
        if (msg != null) {
            Bundle bundle = new Bundle();
            String[] addresses = mDevices.keySet().toArray(new String[mDevices.size()]);
            bundle.putStringArray(KEY_MAC_ADDRESSES, addresses);
            msg.setData(bundle);
            sendMessage(msg);
        }
        Log.d(TAG, msg: "Added " + device.getName() + ": " + device.getAddress());
    }
}
```

Figure 9. onLeScan() method

Figure 9 indicates another important method, onLeScan(), which is called every time the Bluetooth adapter receives a advertisement from a BLE device. BLE device normally sends out plenty of advertising messages in a period of time, so there must be a factor

keeping the host device only responding to new devices during the scan. Therefore, MAC addresses of found devices will be kept to check whether it is known. A name-filter is also added so that the application only looks for devices matching the name ”CC1350 SensorTag” (check Figure 6 for DEVICE_NAME), because the CC1350 SensorTag device is used in this project. When a BLE is detected, it will be added to a map and the MAC address will be put in the MSG_DEVICE_FOUND Message, which later is sent to BleActivity together with MAC addresses of all discovered devices.

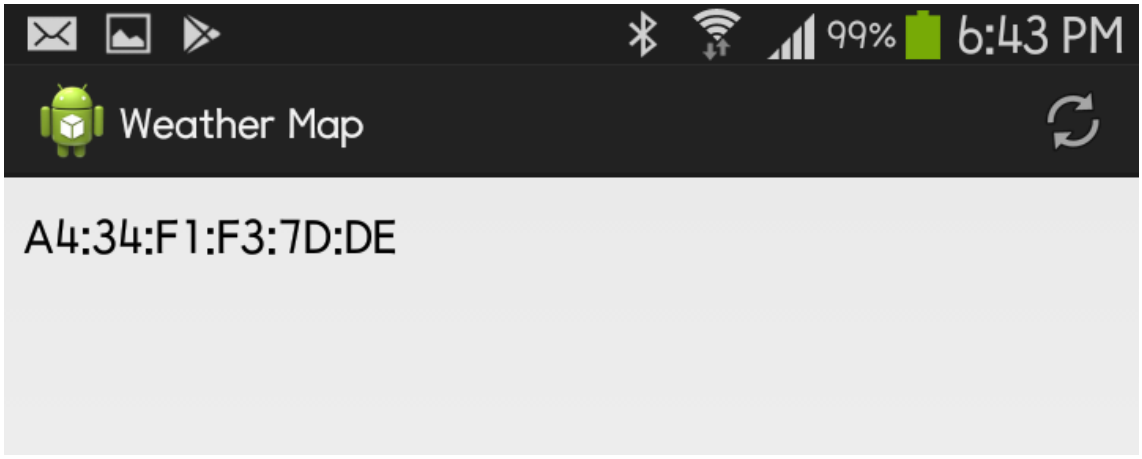


Figure 10. Discovered devices are displayed on user interface

The BleActivity also operates like a state machine, it modifies the user interface based on the state changed by BleService. When BleActivity receives the MSG_DEVICE_FOUND Message from BleService, it will display all found devices on the user interface with their MAC addresses as displayed in figure 10. [2, part 3]

3.1.2 Connecting to SensorTag

Before going to connecting part, it is better to have a basic understanding about BLE standard Profile. Profile generally is a standardized mechanism for Bluetooth devices to exchange data. In the case of BLE, Generic ATTribute (GATT) is the Profile, which allow BLE devices to share small, atomic pieces of data. Any sensor using BLE will include a GATT server that host devices should connect to in order to read and transfer data with the sensor.

There are two steps to connect to GATT server on the sensor. First a local proxy instance serving as the GATT server on host device must be created, then connecting this local proxy to the GATT server on the sensor. To create the local proxy, connectGatt() method

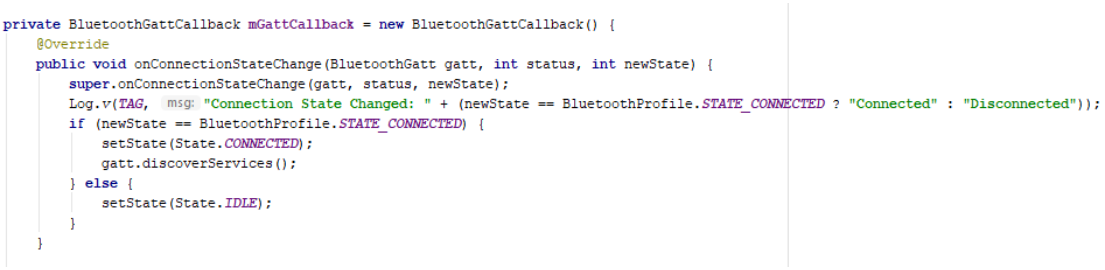
will be called on the BluetoothDevice instance received for each discovered device during scanning phase, which is shown in figure 9.

```
public void connect(String macAddress) {
    BluetoothDevice device = mDevices.get(macAddress);
    if (device != null) {
        mGatt = device.connectGatt( context: this,  autoConnect: true, mGattCallback);
    }
}
```

Figure 11. connectGatt() method in connect() function in BleService

As can be seen from figure 11, connectGatt() method takes three arguments: a Context, a Boolean called autoConnect and BluetoothGattCallback instance (mGattCallback). The method will return a BluetoothGatt object (mGatt) which is the needed local proxy to communicate with GATT server on the sensor.

In detail, the last argument indicates that connectGatt() executes on an asynchronous thread and a callback will be called when it finishes executing. connectGatt() method creates a local proxy and it is included in connect() method, which connects that local proxy to the GATT server on the sensor, and only when the connect is established, the application will receive the callback. If the autoConnect boolean is set to FALSE, local proxy is also created but there is no callback to inform that it has been successfully connected with GATT server of the sensor. Another good thing of autoConnect is that if it is set to TRUE, the created local proxy will automatically connect to GATT server on the sensor, even when connection is lost, the proxy will attempt to reconnect with the sensor without any prompting. Therefore, all the application has to do is to manage the connection state and interact with the connected sensor, which will be done in the callback function. [2, part 4]



```
private BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        super.onConnectionStateChange(gatt, status, newState);
        Log.v(TAG,  msg: "Connection State Changed: " + (newState == BluetoothProfile.STATE_CONNECTED ? "Connected" : "Disconnected"));
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            setState(State.CONNECTED);
            gatt.discoverServices();
        } else {
            setState(State.IDLE);
        }
    }
```

Figure 12. Callback method for manage connection state

The first method, which is indicated in figure 12, allows the application to manage connection state change between host device and sensor, the change of connection state will lead to the state change in the state machine of BleService. As the state changes in the state machine, a Message will be sent to BleActivity so that the Activity can update the user interface properly [2, part 4].

The only left is getting data from the SensorTag, and data exchange between GATT server is enabled by characteristics. In GATT server, there are plenty of GATT services representing different sensor components in the device (temperature sensor, humidity sensor, etc.). Each GATT service involves one or more characteristics, which is a single piece of data of random type. In addition to characteristics, GATT services might contain descriptors of GATT server. The descriptors indicate information about unit of characteristic value or whether users want to receive notification for each change in the value of characteristic. As a rule, GATT services, characteristics and descriptors are identified using a Universally Unique IDentifier (UUID). UUID is an exclusive value of each GATT server, characteristic and descriptor which is allows the application to retrieve them.

```
private static final UUID UUID_HUMIDITY_SERVICE = UUID.fromString("f000aa20-0451-4000-b000-000000000000");
private static final UUID UUID_HUMIDITY_DATA = UUID.fromString("f000aa21-0451-4000-b000-000000000000");
private static final UUID UUID_HUMIDITY_CONF = UUID.fromString("f000aa22-0451-4000-b000-000000000000");

private static final UUID UUID_LUXOMETER_SERVICE = UUID.fromString("f000aa70-0451-4000-b000-000000000000");
private static final UUID UUID_LUXOMETER_DATA = UUID.fromString("f000aa71-0451-4000-b000-000000000000");
private static final UUID UUID_LUXOMETER_CONF = UUID.fromString("f000aa72-0451-4000-b000-000000000000");

private static final UUID UUID_CCC = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");
```

Figure 13. SensorTag UUID

UUIDs of SensorTag can be found in SensorTag attribute table [12]. All UUIDs have the base ”F0000000-0451-4000-B000-00000000”. Last four digits of the first part can be replaced for different GATT services, characteristics and descriptors. As can be seen

from figure 13, the UUID for humidity service is ”F000AA20-0451-4000-B000-00000000”. The humidity service contains two characteristics: the humidity data has the UUID AA21 and a flag, of which UUID is AA22, to turn the sensor on or off. [2, part 5]

On the other hand, after connecting with GATT server, the local proxy of the host device will discover the services in the GATT server. As can be seen from figure 12, this is done by discoverService() method.

```
@Override
public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    Log.v(TAG, msg: "onServicesDiscovered: " + status);
    if (status == BluetoothGatt.GATT_SUCCESS) {
        subscribe(gatt);
    }
}
```

Figure 14. onServicesDiscovered callback function

When services in GATT server are discover, onServicesDiscovered will be called. In the callback function shown in figure 14, necessary services and characteristics will be subscribed to the local proxy (BluetoothGatt instance) by subcribe() method. The content of subcribe() method will be explained in next sections.

3.2 Displaying map with weather information

In previous section, local proxy has been created in host device and connected to the GATT server in the SensorTag. Local proxy also discovered and knew about the GATT services provided by the server. The next thing is to obtain the services, then the characteristics and descriptors to use them. In this project, characteristics of GATT service will be used to show weather information of user’s current location on a map. Achieving services, characteristics and descriptors will be done by subscrible() function in BleService.

```
private void subscribe(BluetoothGatt gatt) {
    BluetoothGattService humidityService = gatt.getService(UUID_HUMIDITY_SERVICE);
    if (humidityService != null) {
        BluetoothGattCharacteristic humidityCharacteristic = humidityService.getCharacteristic(UUID_HUMIDITY_DATA);
        BluetoothGattCharacteristic humidityConf = humidityService.getCharacteristic(UUID_HUMIDITY_CONF);
        if (humidityCharacteristic != null && humidityConf != null) {
            BluetoothGattDescriptor config = humidityCharacteristic.getDescriptor(UUID_CCC);
            if (config != null) {
                gatt.setCharacteristicNotification(humidityCharacteristic, enable: true);
                humidityConf.setValue(ENABLE_SENSOR);
                write(humidityConf);
                config.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                write(config);
            }
        }
    }
```

Figure 15. subscribe() function in BleService

In Android Studio there is a class called BluetoothGattService, which can represent a GATT service in the server. Figure 15 shows that a BluetoothGattService object can be obtained by BluetoothGatt instance using getService() method with the correct UUID of the service. Moreover, GATT characteristics and descriptors are represented by BluetoothGattCharacteristic and BluetoothGattDescriptor objects respectively. These objects can be obtained by BluetoothGattService object using getCharacteristic() or getDescriptor() method with the proper UUID of the characteristics and descriptors.

To get notifications for changes in data, first the sensor must be turned on. This is done by writing the value 0x01 (ENABLE_SENSOR) to UUID_HUMIDITY_CONF characteristic. Then notifications on local proxy also must be enabled by calling method setCharacteristicNotification(). Finally, writing value {0x00,0x01} (ENABLE_NOTIFICATION_VALUE) to the UUID_CCC descriptor to switch on notifications on the humidity GATT service.

After receiving the characteristic, the application has to calculate and convert the data to human measuring units for temperature and humidity. This task is executed by another callback function called onCharacteristicChanged, which is called whenever a change in characteristic occurs. [2, part 6]

```
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
    Log.v(TAG, msg: "onCharacteristicChanged: " + characteristic.getUuid());
    if (characteristic.getUuid().equals(UUID_HUMIDITY_DATA)) {
        int t = shortUnsignedAtOffset(characteristic, offset: 0);
        int h = shortUnsignedAtOffset(characteristic, offset: 2);
        float humidity = ((float)h / 65536f)*100f;
        float temperature = ((float)t / 65536f)*165f - 40f;
        Log.d(TAG, msg: "Value: " + humidity + ":" + temperature);
        Message msg = Message.obtain( h: null, MSG_DEVICE_DATA);
        msg.arg1 = (int) (temperature * 100);
        msg.arg2 = (int) (humidity * 100);
        sendMessage(msg);
    }
```

Figure 16. onCharacteristicChanged callback function

Data from humidity characteristic includes both temperature value in bytes 0 and 1, and humidity value in bytes 2 and 3. As can be seen from figure 16, the formula for calculate the temperature and humidity from raw value is:

$$temperature = (rawTemp \div 65536) \times 165 - 40$$
$$humidity = (rawHum \div 65536) \times 100 \quad [13]$$

Then the data is attached to arg1 and arg2 fields of the Message and a MSG_DEVICE_DATA Message containing weather data will be sent to BleActivity. When receiving a MSG_DEVICE_DATA Message, Message Handler of BleActivity will display the data on user interface with the method setData().

```
public void handleMessage(Message msg) {
    BleActivity activity = mActivity.get();
    if (activity != null) {
        switch (msg.what) {
            case BleService.MSG_STATE_CHANGED:
                activity.stateChanged(BleService.State.values(
                break;
            case BleService.MSG_DEVICE_FOUND:
                Bundle data = msg.getData();
                if (data != null && data.containsKey(BleServic
                    activity.mDeviceList.setDevices(activity,
                }
                break;
            case BleService.MSG_DEVICE_DATA:
                float temperature = msg.arg1 / 100f;
                float humidity = msg.arg2 / 100f;
                activity.mMap.setData(temperature, humidity);
                break;
```

Figure 17. Message Handler in BleActivity

As can be seen from figure 17, the temperature and humidity value will multiply by 100 before sending to BleActivity and then are divided back by 100 in BleActivity in order to get a float value. [2, part 6]

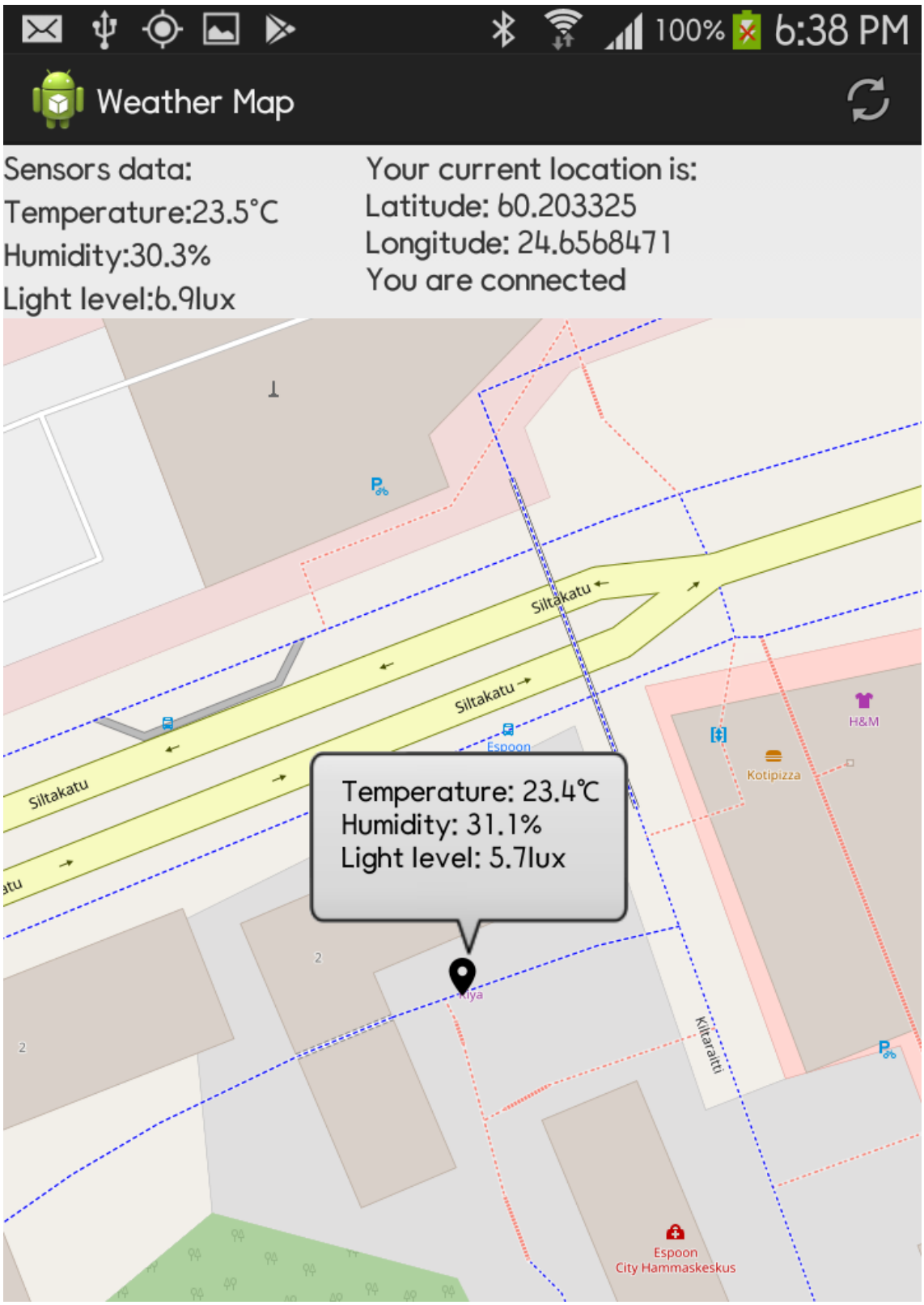


Figure 18. Display data on map and user interface

As can be seen from figure 18, marker indicates user's current location on the map, and by tapping on the marker, user also is able to see the weather information of the location.

If there are changes in user's location, the position of marker in the map as well as the weather information shown on tapping it will be updated. All of these actions will be managed by BleActivity after it receives the data Message.

3.3 Sending received data to database

As mentioned above, the application will attach weather data from SensorTag and GPS data from host device to a JSON object. Then a post request will be sent to server side API by the application, and the API will send the JSON object to the database.

The JSON object is created in JsonData class of the application. As can be seen in figure 19, the class takes temperature, humidity, light level data from SensorTag and a Location object as member variables. These variables then are put into a JSONObject in the constructor.

```
public class JsonData {
    public double m_temp;
    public double m_humi;
    public double m_light;
    public Location m_location;

    private JSONObject data;

    public JsonData(double temp, double humi, double light, Location location){
        m_temp = temp;
        m_humi = humi;
        m_light = light;
        m_location = location;

        data = new JSONObject();
        try{
            data.put( name: "temperature", m_temp);
            data.put( name: "humidity", m_humi);
            data.put( name: "light", m_light);
            data.put( name: "latitude", m_location.getLatitude());
            data.put( name: "longitude", m_location.getLongitude());
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
```

Figure 19. JsonData class

To send post request to the API, JsonData class uses a method of Android Studio called ”HTTP Post”. HTTP Post in Java is used to request server side to receive and store data. The submitted data is stored in name-value pairs, such as name-your name, email-your email, id-your id, etc. HTTP Post is implemented in post function of JsonData class.

```
public String post(String url){
    String result = "";
    InputStream inputStream = null;
    try {
        HttpClient httpClient = new DefaultHttpClient();
        HttpPost httpPost = new HttpPost(url);
        String json = this.data.toString();
        StringEntity se = new StringEntity(json);
        httpPost.setEntity(se);
        httpPost.setHeader( name: "Accept",  value: "application/json");
        httpPost.setHeader( name: "Content-type",  value: "application/json");
        HttpResponse httpResponse = httpClient.execute(httpPost);
        inputStream = httpResponse.getEntity().getContent();
        if(inputStream != null)
            result = convertInputStreamToString(inputStream);
        else
            result = "Did not work!";
    }catch (Exception e){
        Log.d( tag: "InputStream", e.getLocalizedMessage());
    }
    return result;
}
```

Figure 20. post() function in JsonData class

As can be seen from figure 20, first, an object of HttpClient and HttpPost have to be created, the ”url” parameter is the url of the server receiving the data. Next, the data need to be encoded into valid URL format, this is done by setEntity() method. Finally, headers are set before making an HTTP Post request by calling execute() method with HttpClient object. As a result, when the application receives data from SensorTag, a JsonData object will be created with all received data in its variables to send a post request to the server side API.

4 Implementation of Database, API and Weather Map Website

This project aims to study the operation of different cloud services, therefore, both Firebase and Heroku cloud service trials will be carried out. The Firebase will be used

for database implementation because it is simple to create a database with Realtime Database feature of Firebase. Besides, the server side API is written in JavaScript and executes in Node.js runtime environment. Moreover, the API will be hosted on Heroku cloud service, the API only has two functions for getting and posting data hence Heroku will not charge money because it is free for small projects.

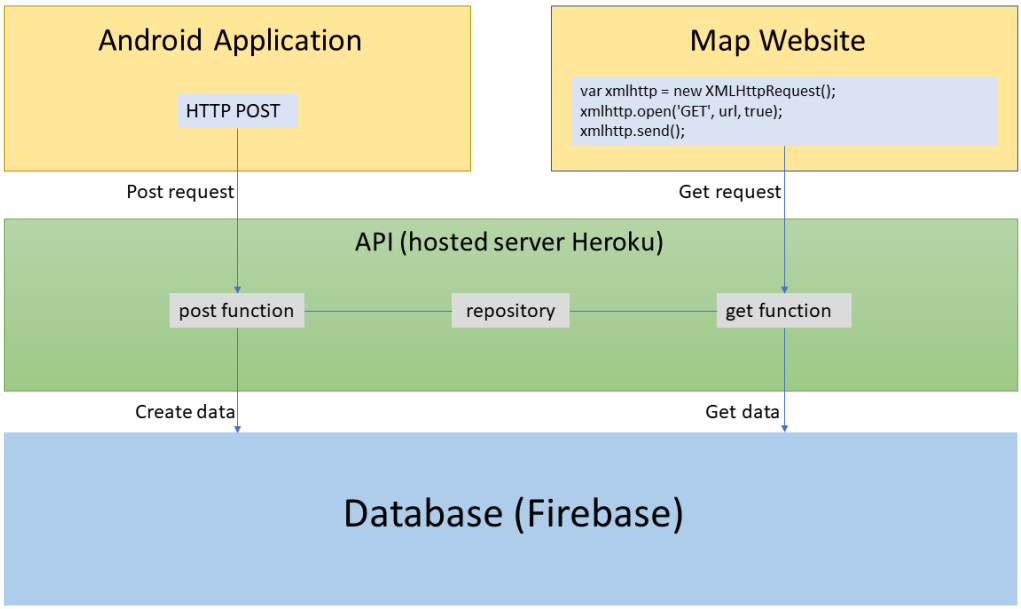


Figure 21. API usage flowchart

As can be seen from figure 21, the API includes a post function to post data to the database and a get function to get data from database. Besides, the API contains a repository which directly linked with the database. Two functions of the API use the repository to communicate and exchange data with the database. The above section has indicated that when user runs the Android application, the application will send a data post request to the API by HTTP Post method. After receiving the post request, the API will execute the post function where repository creates data based on the received data and push it to the database. Equally important, when user accesses the website to see the weather map, the website will send a get data request to the API. Then get function will be called by the API and as the function executes, the repository will get the data from database so that weather map website can present the weather data on the world map.

This section only provides important steps of setting up and explanations to main functions of the API and the website.

4.1 Creating Realtime Database on Firebase

The database for this project is implemented by utilizing the Realtime Database feature of Firebase. It is quite straightforward to create a realtime database in Firebase, and steps of setting up a Firebase database in this section follow an reliable article about how to set up Firebase [14]. The first step, which is indicated in figure 22, is accessing Firebase homepage and create a project.

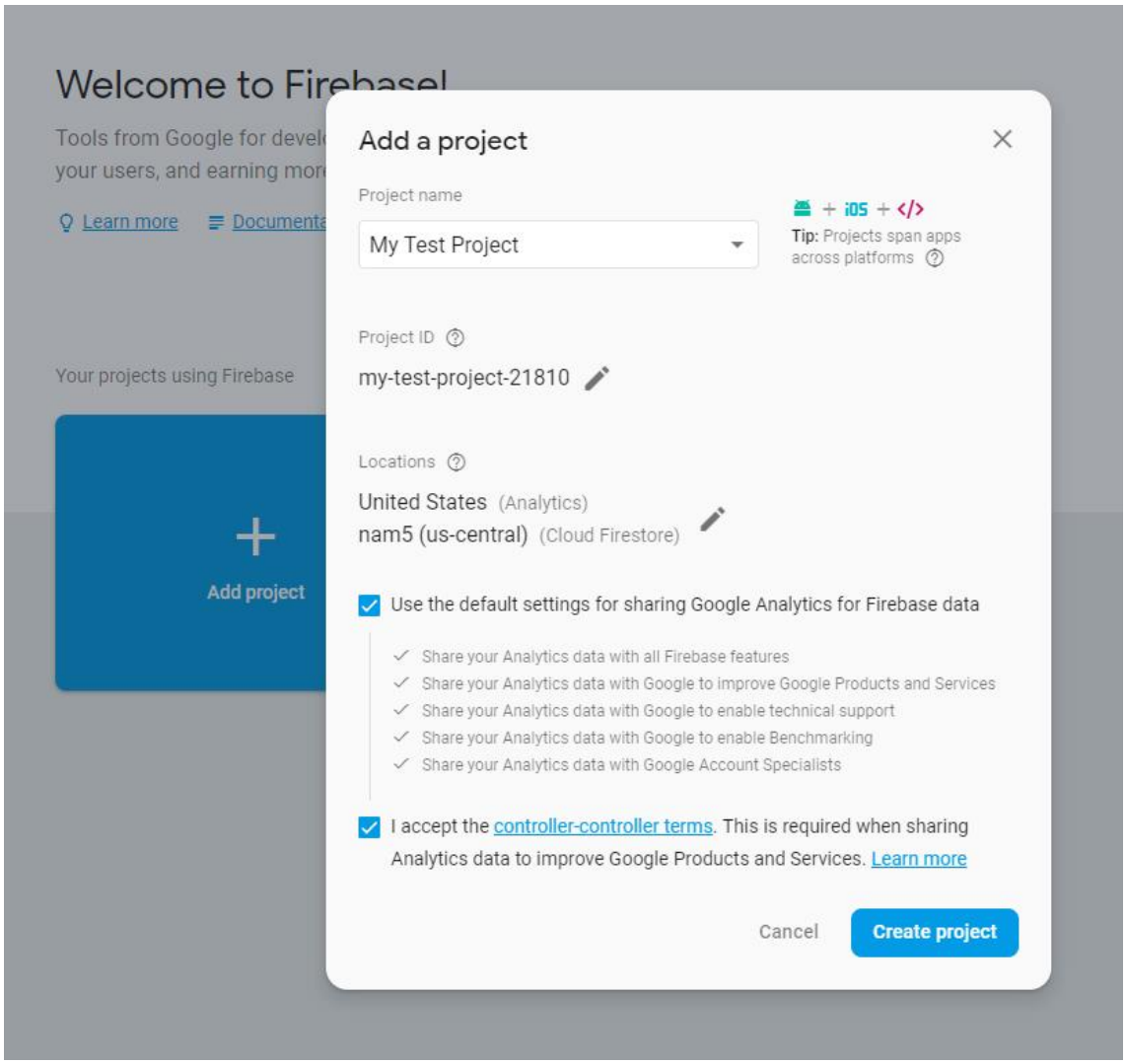


Figure 22. Create a project in Firebase

Then in the project page shown in figure 23, selecting the Database menu and choose Create database.

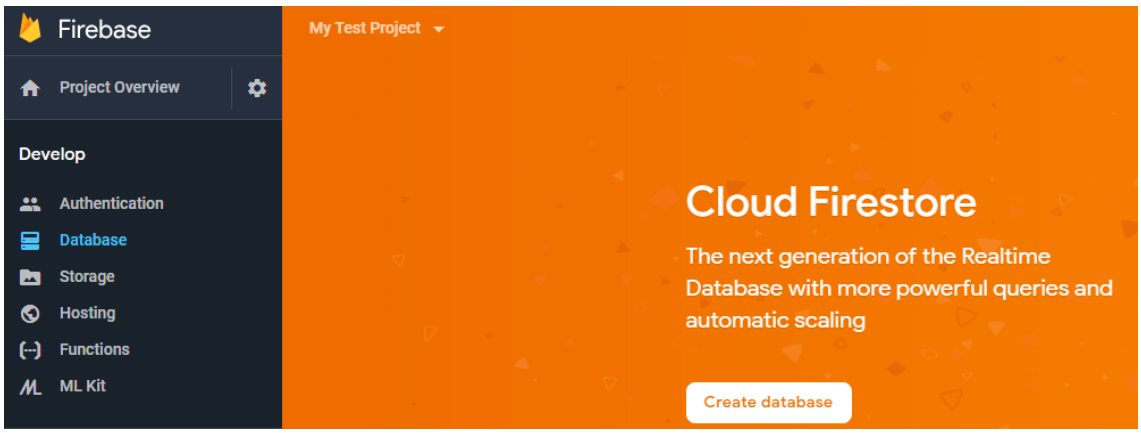


Figure 23. Create database in Firebase project

There will be a prompt asking for data security mode: locked mode and test mode. In locked mode, the database is private and all reading as well as writing action will be denied. On the other hand, if the database is in test mode, anyone who has the database reference can read data and write on the database.

Then user need to get a service account key, this key allows users to add the Firebase Admin SDK to their server such as reading and writing on Realtime Database with full admin privileges, sending Firebase Cloud Messaging messages, generating and verifying Firebase authentication tokens, etc. [15]

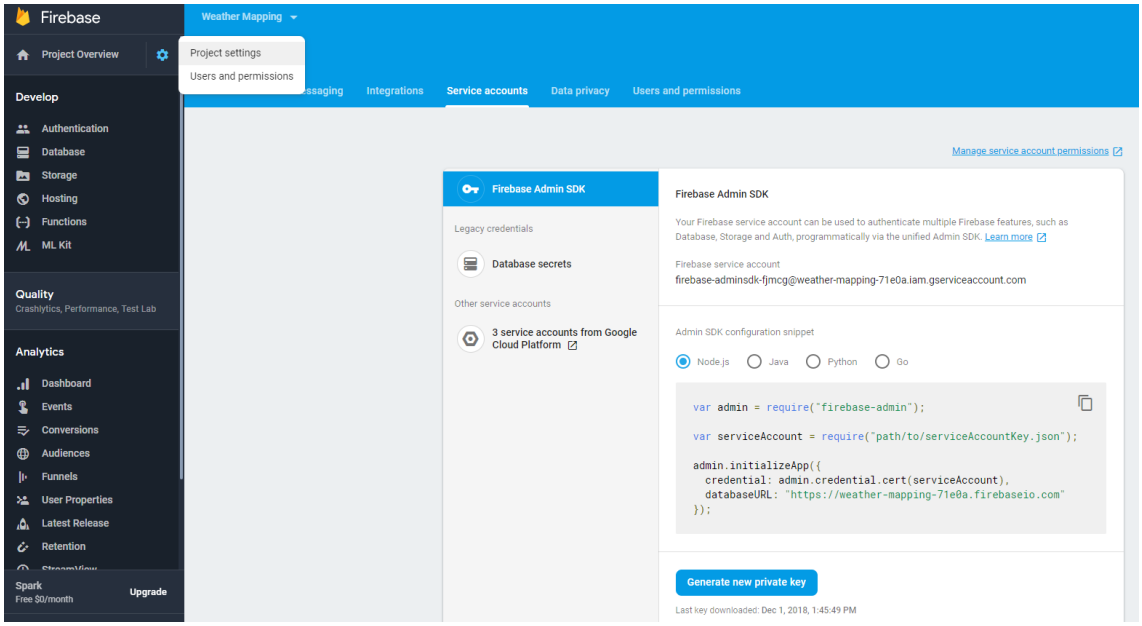


Figure 24. Get service account key

As can be seen in figure 24, to get the service account key, user has to access the ”Service accounts” menu in the Project settings. After clicking the ”Generate new private key” button, a JSON file consisting of all project credentials will be downloaded, this file will be used to initialize the SDK in the server side API.

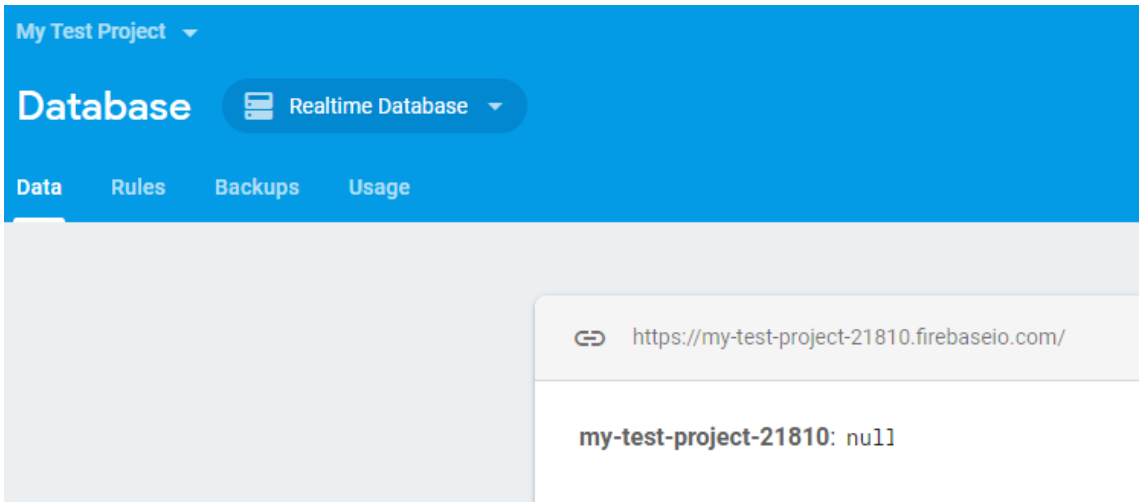


Figure 25. Link to Firebase database

As can be seen from figure 25, after creating the database, user will receive a link to the database in the Database menu of the project. This link will be given to the server so that the server can access the database to post and get data.

4.2 API implementation

4.2.1 API repository implementation

The repository of the API connects directly to Firebase database to allow the API to exchange data with the database. First, connection between API repository and database must be established.

```
firebase.initializeApp({
  serviceAccount: './Weather Mapping-09c00ead2a4b.json',
  databaseURL: 'https://weather-mapping-71e0a.firebaseio.com/'
})

var ref = firebase.database().ref('weatherData');
```

Figure 26. Commands to connect API repository and database

As can be seen from figure 26, the ”serviceAccout” parameter is the directory is the JASON file of service account key and ”databaseURL” parameter is the link to the database in Figure 23. After connecting the repository to the database, a data reference of exchaged data must be created, in this repository, the data reference is called ”weatherData”. Last but not least, the API repository must contain functions for getting and posting data.

```
// FIREBASE
module.exports.getData = function(callbackSucess, callbackError) {
  ref.once('value', callbackSucess, callbackError);
}

module.exports.createData = function(weather, callback) {
  const currentDateString = (new Date()).toString();
  const weatherWithCreatedAt = Object.assign({}, weather, { createdAt: currentDateString });
  ref.push(weatherWithCreatedAt, callback);
}
```

Figure 27. Functions for getting and posting data

As can be seen from figure 27, getting and posting functions are quite simple to implement. However, in createData function, before pushing the data to the database, the current time is added to the data so that the API can detect when it is posted to database for future processing and calculating.

4.2.2 API controller implementation

The API controller receives requests from the application and website. Based on the type of the requests, the controller will execute getting or posting function in which the repository communicates and exchanges data with the database.

```
Weather = require('./model/weather');

app.get('/', function(req, res) {
  res.send('Please use /api/...');
});

app.get('/v2/weather', function(req, res) {
  res.set('Access-Control-Allow-Origin', '*');
  Weather.getData(function(snap) {
    const weather = convertObjectToArray(snap.val());
    const filteredWeather = weather.filter(weather => {
      const createdAt = new Date(weather.createdAt);
      console.log(weather);
      return createdAt.setHours(0,0,0,0) >= (new Date(new Date() - 1)).setHours(0,0,0,0);
    });

    res.json(filteredWeather ? filteredWeather : []);
  }, function(error) {
    console.log(error);
  });
});

app.post('/v2/weather', function(req, res) {
  res.set('Access-Control-Allow-Origin', '*');
  Weather.createData(req.body, function(error) {
    if (!error) {
      res.json(req.body);
    }
  });
});
```

Figure 28. API controller get and post functions

As can be seen from figure 28, to be able to use the repository in get and post functions, a reference of the repository called Weather has to be created in the beginning. The first get function is for the default API and the second get function is for the API of this project. As shown in figure 26, there is a filter for getting data in the get function. The reason of the filter is that the weather map website only shows weather information within one day, therefore, the data posted on previous days of the current day is not collected to show on the map. Finally, in post function, the repository reference posts data to the database if there is no error.

4.2.3 Deploying API to Heroku

After implementation, the API must be deployed to Heroku to be hosted and run persistently. Heroku is integrated in GitHub therefore it manages application deployment with Git. The steps for code deployment to Heroku are simple and user does not need to be a Git expert to do it.

This section will not go deeper about Heroku code deployment because all the instructions have been stated clearly in Deploying with Git article [16] on Heroku.

4.3 Weather map website implementation

The implementation of weather map website is based on the guidance of Google Maps Platform for creating a Google map with makers for website [17]. Moreover, the map also gets temperature, humidity, and light level data from the database to show together with the locations on the markers.

```
function getWeatherCoordinates() {
    var xmlhttp = new XMLHttpRequest();
    var url = 'https://weathermap-api.herokuapp.com/v2/weather'
    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            weatherData = JSON.parse(this.responseText);
            const weatherDataArray = convertReceivedJsonToArray(weatherData);
            console.log(weatherDataArray);
            initMap(weatherDataArray);
        }
    }
    xmlhttp.open('GET', url, true);
    xmlhttp.send();
}

function initMap(weatherData) {
    if (weatherData == null || weatherData.errorCode || weatherData.length === 0) {
        document.getElementById('message').innerHTML = 'No data to show';
        return;
    }

    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: 15,
        center: getMapCenter(weatherData)
    });

    weatherData.forEach(data => {
        const title = "Temperature: " + data.temperature +
            "<br>Humidity: " + data.humidity +
            "<br>Light: " + data.light;

        var infowindow = new google.maps.InfoWindow({
            content: title
        });

        const point = { lat: data.latitude, lng: data.longitude };

        var marker = new google.maps.Marker({
            position: point,
            map: map,
            title: title
        });

        google.maps.event.addListener(marker, 'click', function() {
          infowindow.open(map,marker);
        });
    });
}
```

Figure 29. Initializing and getting data for weather map website

As can be seen from figure 29, getting data for weather map website is executed in getWeatherCoordinates() function. This function uses XMLHttpRequest which can be used to request data from a web server, XMLHttpRequest object is integrated in almost all modern browsers. The onreadystatechange property defines a function executing every time the status of the XMLHttpRequest object changes. The map is initialized only when the response is ready, which means when readyState property is 4 and the status

property is 200. Then the open() function asks for the type of the request which is ”GET” and url to send the request to. Finally, XMLHttpRequest object sends the request to web server by calling send() method.

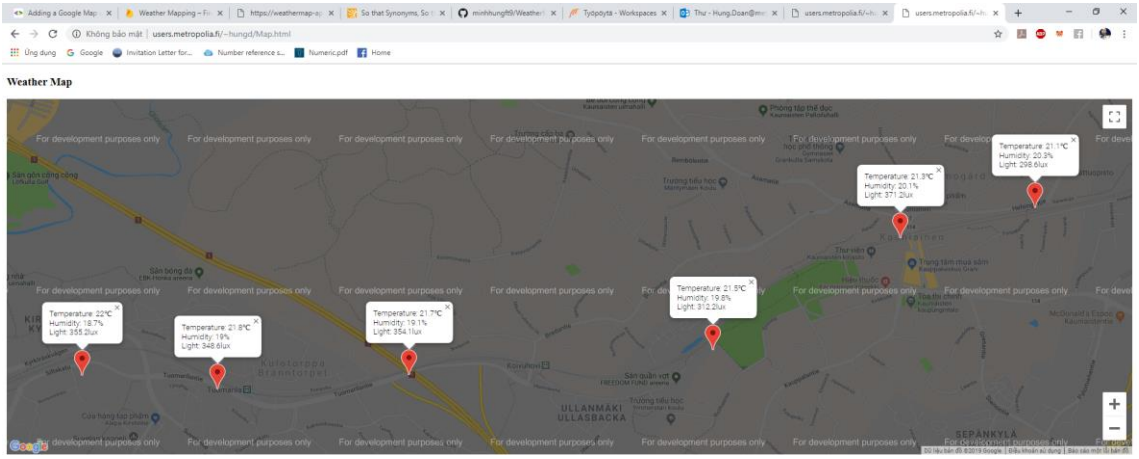


Figure 30. Weather map website with weather information

The initMap() function initializes and shows the Google map on the website. The Android application posts JSON objects containing temperature, humidity, light level and location data, and as can be seen from figure 30, for each JSON object, the map will create a marker for the location and the weather information will be indicated on the marker.

5 Conclusion

The main goal of the project was to develop a weather map website which shows information about temperature, humidity and light level in different locations, the weather data is taken from a database hosted by Firebase. In order to collect and post weather data to the database, an Android application and a server API were also developed. The Android application connects with a sensor device called SensorTag to collect environmental data and try to post it to the database by making a post request. Then server side API handles post requests from the application and post the data to the database. The API also allow the website to get data from the database.

After a plentiful amount of studying about Firebase, Heroku, Node.js and JavaScript, all main components of the project, which is the Android application, the server side API and the map website, has been successfully developed. If the project is widely used in the future, user can access the weather map website to get update of current weather condition in all over the world locations, which might be more accurate comparing to the weather forecast. The project can also be used to keep track of weather condition during traveling. For example, parents can access weather map website to check weather condition at traveled locations if their kids are going on a camping trip at school, this would make the parents assured about their kids. Moreover, doctors dealing with respiratory or heart diseases can use the project to observe patients' daily environment condition to give advices and take action if something bad happens.

However, the project needs more improvement and further development to be popular in the future. At the moment, the application only works with SensorTag BLE device, it would be better if the application can connect to more types of BLE device. Besides, route planning feature is also needed so that the application can indicate the way and the weather condition of location that user wants to visit. Last but not least, an account managing system should be developed to allow user to login and share their data or keep an eye on data of another specific user.

References

[1] C. Riggs. Explain the Importance of Maps in Human Geography. 2016. [Online] Available from: https://www.1dea.me/2016/10/02/explain-importance-maps-human-geography/. [Accessed 7 March 2019]

[2] M. Allison. Bluetooth LE. 2014. [Online] Available from: https://blog.stylingandroid.com/bluetooth-le-part-1/. [Accessed 7 March 2019]

[3] National Geographic. Map. [Online] Available from: https://www.nationalgeographic.org/encyclopedia/map/. [Accessed 7 March 2019]

[4] National Research Council. Expanding the Vision of Sensor Materials. 1995. [Online] Available from: https://www.nap.edu/read/4782/chapter/1. [Accessed 7 March 2019]

[5] Texas Instruments. Simplelink CC1350 SensorTag Bluetooth and Sub-1GHz Long Range Wireless Development Kit. [Online] Available from: http://www.ti.com/tool/cc1350stk#technicaldocuments. [Accessed 7 March 2019]

[6] S. Rostad. What is Heroku? A Simple Explanation for Non-Techies. February 28 2018. [Online] Available from: https://trifinlabs.com/what-is-heroku/. [Accessed 14 March 2019]

[7] A. Fu. 7 Different Types of Cloud Computing Structures. March 3 2017. [Online] Available from: https://www.uniprint.net/en/7-types-cloud-computing-structures/. [Accessed 15 March 2019]

[8] K. Rusev, D. Gadzhev. What is Heroku Used For? May 15 2018. [Online] Available from: https://mentormate.com/blog/what-is-heroku-used-for-cloud-development/. [Accessed 14 March 2019]

[9] Heroku. Dynos: the heart of the Heroku platform. [Online] Available from: https://www.heroku.com/dynos. [Accessed 16 March 2019]

[10] Hackernoon. Introducetion to Firebase. December 28 2017. [Online] Available from: https://hackernoon.com/introduction-to-firebase-218a23186cd7. [Accessed 11 March 2019]

[11] R. Rupareliya. Get Started With Firebase Android. January 25 2017. [Online] Available from: http://www.androidgig.com/getting-started-with-firebase-android/. [Accessed 13 March 2019]

[12] Texas Instruments. SensorTag attribute table. [Online] Available from: http://processors.wiki.ti.com/images/a/a8/BLE_SensorTag_GATT_Server.pdf. [Accessed 22 March 2019]

[13] Texas Instruments. CC2650 SensorTag User’s Guide. [Online] Available from: http://processors.wiki.ti.com/index.php/CC2650_SensorTag_User's_Guide#Humidity_Sensor. [Accessed 22 March 2019]

[14] P. Chang. [Nodejs] Setup Firebase in 4 step. April 6 2018. [Online] Available from: https://hackernoon.com/nodejs-setup-firebase-in-4-step-tutorial-example-easy-beginner-service-account-key-json-node-server-d61e803d6cc8. [Accessed 1 April 2019]

[15] Firebase. Add the Firebase Admin SDK to Your Server. Last updated on 28 March 2019. [Online] Available from: https://firebase.google.com/docs/admin/setup. [Accessed 1 April 2019]

[16] Heroku Dev Center. Deploying with Git. [Online] Avaible from: https://devcenter.heroku.com/articles/git#deploying-code. [Accessed 2 April 2019]

[17] Google Maps Platform. Adding a Google Map with a Marker to Your Website. Last updated on 28 February 2019. [Online] Available from: https://developers.google.com/maps/documentation/javascript/adding-a-google-map. [Accessed 2 April 2019]