

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



PHÁT TRIỂN ỨNG DỤNG INTERNET OF THINGS (CO3038)

**LAB 5: IMPLEMENT THE ANDROID PROGRAM
AN MQTT APPLICATION ON MOBILE**

SV thực hiện:

Nguyễn Trọng Tín

2012215

Contents

1	INTRODUCTION	2
1.1	Android	2
1.2	Android Studio	3
1.3	JAVA and KOTLIN	4
2	IMPLEMENTATION	6
2.1	Overview	6
2.2	The Sign-in View	6
2.3	The Monitor View	8
2.4	MQTT Handler	10
3	REFERENCES	11

List of Figures

1	Overview the first android app	6
2	Button Clicked Listener	8
3	Different parts in CardView	8
4	MDC Android	8
5	Direction to add Image asset	10
6	MQTTHelper's Implementation and Abstract	10
7	Singleton Pattern	11
8	Using Singleton Pattern	11

1 INTRODUCTION

1.1 Android

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. It has become one of the most popular operating systems in the world, powering billions of devices worldwide.

Key features and aspects of Android include:

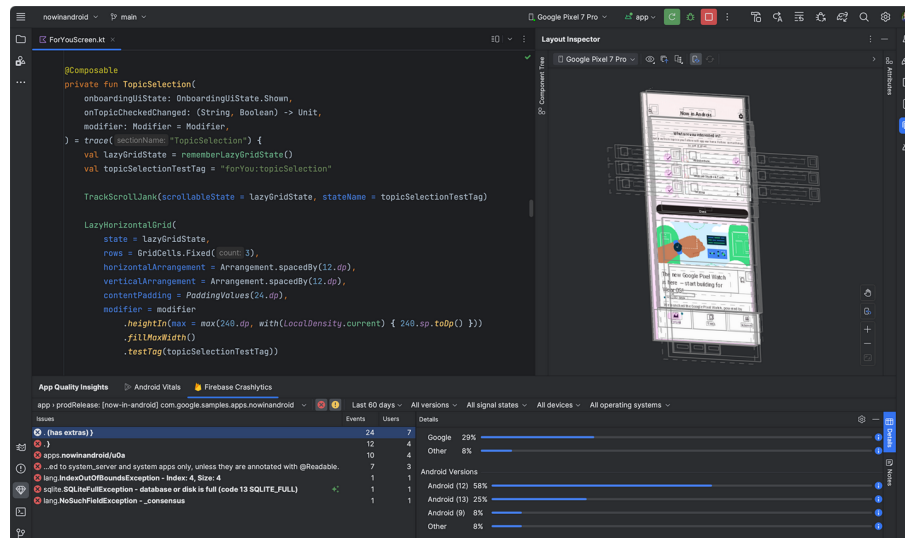
- **Open-Source Platform:** Android is an open-source platform, allowing developers to customize and modify the operating system according to their needs. This openness has led to a thriving ecosystem of apps, custom ROMs, and community contributions.
- **Google Play Store:** Android devices come pre-installed with the Google Play Store, which provides access to a vast library of apps, games, movies, music, and other digital content. Users can download and install apps from the Play Store onto their devices.
- **Customization:** Android offers extensive customization options for users, including the ability to change wallpapers, themes, widgets, and launchers. Users can personalize their devices to suit their preferences and style.
- **Security:** Google regularly updates Android with security patches and features to protect users from malware, viruses, and other security threats. Android devices also come with built-in security features such as Google Play Protect, which scans apps for potential risks.
- **Integration with Google Services:** Android seamlessly integrates with various Google services such as Gmail, Google Maps, Google Drive, and Google Assistant. This integration provides users with access to a wide range of productivity tools and services.



android

1.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Android app development, provided by Google. It offers a comprehensive set of tools and features to streamline the app development process, from design to deployment. Android Studio is built on the IntelliJ IDEA platform and is tailored specifically for Android development.



Key features and aspects of Android Studio include:

- **Intelligent Code Editor:** Android Studio provides an intelligent code editor with features such as code completion, syntax highlighting, and code refactoring to enhance productivity. It supports multiple programming languages for Android development, including **Java**, **Kotlin**, and **C++**.
- **Layout Editor:** The Layout Editor allows developers to visually design user interfaces for their apps, with drag-and-drop functionality and real-time previews. Developers can create complex layouts and customize UI components with ease.
- **Built-in Emulator:** Android Studio includes a built-in emulator for testing apps on different Android device configurations. Developers can simulate various screen sizes, resolutions, and hardware configurations to ensure their apps work seamlessly across different devices.
- **Performance Profiling Tools:** Android Studio offers performance profiling tools to help developers analyze and optimize their app's performance. Developers can monitor CPU, memory, and network usage, identify performance bottlenecks, and optimize their code for better efficiency.
- **Version Control Integration:** Android Studio integrates with version control systems such as Git, enabling developers to manage and collaborate on their codebase efficiently. It provides features for code versioning, branching, merging, and conflict resolution.

- **Gradle-based Build System:** Android Studio uses the Gradle build system to automate the process of building, testing, and deploying Android apps. Gradle provides flexibility and customization options for managing dependencies, building app variants, and generating APKs.
- **Google Play Services Integration:** Android Studio seamlessly integrates with Google Play services, allowing developers to easily integrate features such as maps, location services, authentication, and cloud messaging into their apps. Developers can leverage Google Play services APIs to add powerful functionality to their apps with minimal effort.
- **Extensible:** Android Studio is highly extensible, with support for plugins and extensions that enhance its functionality and cater to specific use cases or development workflows. Developers can install plugins to add new features, tools, or integrations, customizing Android Studio to suit their needs.

1.3 JAVA and KOTLIN

When we experienced Android Studio, we had to get involved and develop our application by using two robust and powerful languages: Java and Kotlin.

Java is a widely-used, object-oriented programming language developed by Sun Microsystems (now owned by Oracle Corporation) in the mid-1990s. It is designed to be platform-independent and can run on any device with a Java Virtual Machine (JVM), making it highly portable. Java is known for its simplicity, readability, and vast ecosystem of libraries and frameworks.



- **Platform Independence:** Java programs can run on any device with a JVM, regardless of the underlying operating system. This makes Java highly portable and suitable for developing cross-platform applications.
- **Object-Oriented:** Java is a pure object-oriented programming language, emphasizing the concept of objects and classes. It supports encapsulation, inheritance, and polymorphism, allowing developers to write modular and reusable code.
- **Robust and Secure:** Java provides built-in features for error handling, memory management, and security. It includes automatic garbage collection, exception handling, and strong type checking to ensure robust and reliable code.
- **Large Ecosystem:** Java has a vast ecosystem of libraries, frameworks, and tools that facilitate software development. Popular frameworks like Spring, Hibernate, and Apache Struts are widely

used for building enterprise-level applications.

- **Community Support:** Java has a large and active community of developers, contributing to its growth and evolution. The Java Community Process (JCP) oversees the development of the Java language and platform, ensuring ongoing updates and improvements.



Kotlin is a modern, statically-typed programming language developed by JetBrains, the company behind IntelliJ IDEA, in 2011. Kotlin is designed to be fully interoperable with Java, allowing developers to seamlessly integrate Kotlin code with existing Java projects. Kotlin aims to address some of the limitations of Java while providing a more concise and expressive syntax.

- **Concise Syntax:** Kotlin offers a concise and expressive syntax, reducing boilerplate code and improving readability. Features such as type inference, data classes, and extension functions allow developers to write more compact and expressive code.
- **Null Safety:** Kotlin provides built-in null safety features to prevent null pointer exceptions, a common source of errors in Java. Nullable and non-nullable types, along with safe call operators and the Elvis operator, help developers write safer and more robust code.
- **Coroutines:** Kotlin introduces coroutines, a lightweight concurrency framework for asynchronous programming. Coroutines simplify the handling of asynchronous tasks, such as network requests or long-running computations, by providing a sequential and structured approach to concurrency.
- **Interoperability with Java:** Kotlin is fully interoperable with Java, allowing developers to use Kotlin and Java code together in the same project. Kotlin code can call Java code and vice versa, making it easy to migrate existing Java projects to Kotlin or adopt Kotlin gradually.
- **Officially Supported by Google:** In 2017, Google announced Kotlin as an official programming language for Android app development, alongside Java. Kotlin's modern features and seamless integration with existing Android projects have made it increasingly popular among Android developers.

2 IMPLEMENTATION

2.1 Overview

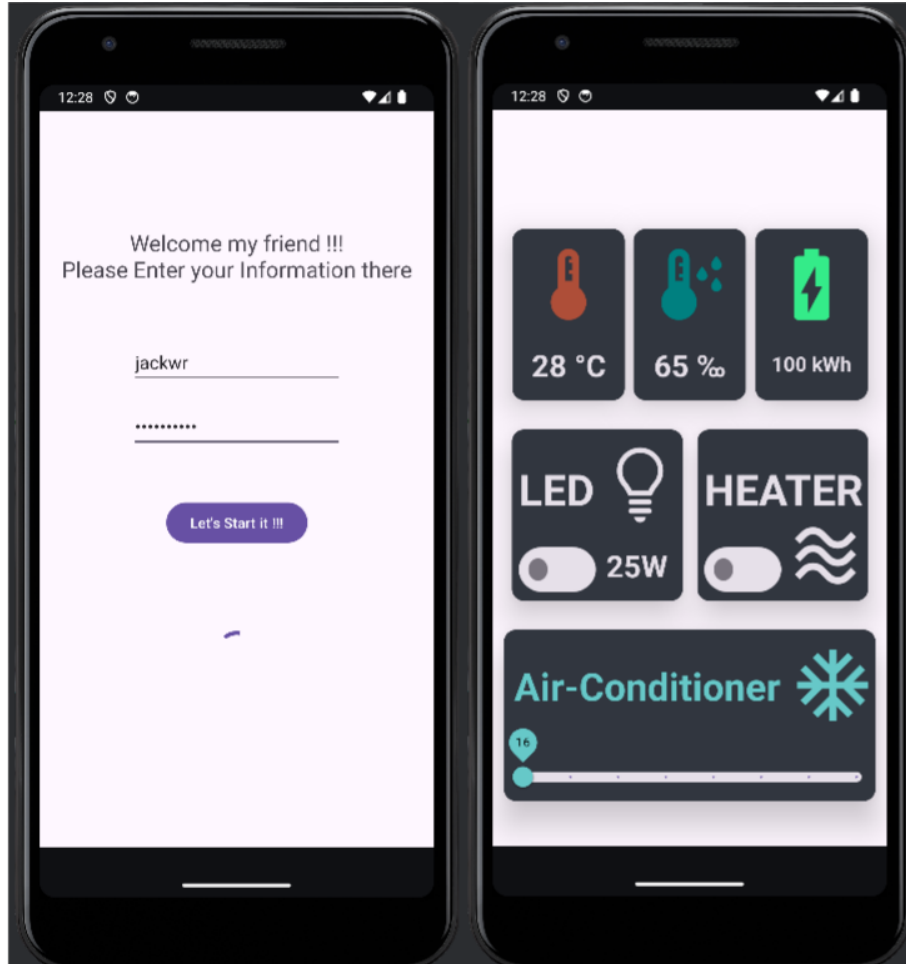


Figure 1: Overview the first android app

My android app includes two **Views** which is called as **Activity** in Android Studio.

* The first view is **Sign-in view**, which takes your **Username** and **Password** in order to get access a Monitor view.

* From the **Monitor View**, at this moment, there are 3 controller cards: LED, Heater, and Air conditioner, and accompanied by 3 status cards. This view allows us to monitor devices according to specific functions such as **Switching** on-off and **Slider** for adjusting temperature. Moreover, we can visually observe the figures of **Temperature**, **Humidity**, and **Solar Battery**

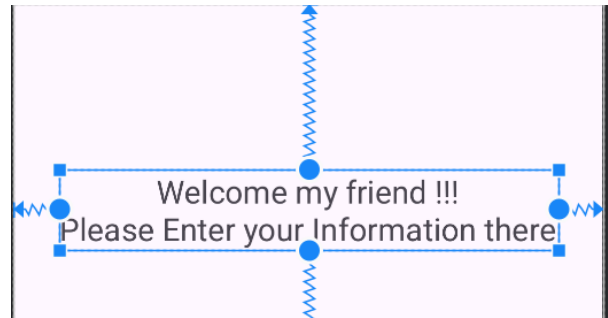
2.2 The Sign-in View

In general, there are 3 types of components: TextView, EditText, and Button

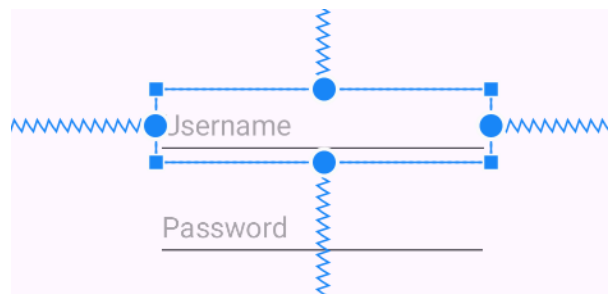
- **TextView**: user interface (UI) element that is used to display text on the screen. It is one of the

most commonly used UI elements in Android app development for showing static text content to users. That means we are only able to edit it when programming; therefore, users cannot edit it in runtime. (Some backend functions in runtime can change it.)

I use it to show my greeting text.



• **EditText**: essential user interface (UI) element in Android Studio, used for capturing user input in the form of text. It allows users to enter and edit text data, such as usernames, passwords, email addresses, or any other type of textual information.



• **Button**: the element used to trigger an action or perform an operation when tapped or clicked by the user. Buttons are commonly used to initiate actions such as submitting a form, navigating to another screen, or executing a function within an app.

In this case, I use it to verify the sign-in information and forward to the next MonitorView.



The special feature of buttons is calling the trigger function when it is clicked. They call it **Button-Clicked-Listener**. I used this to trigger a function that allows to go to the Monitor View.


```
btnHello.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this, MonitorActivity.class);
        startActivity(intent);
    }
});
```

Figure 2: Button Clicked Listener

2.3 The Monitor View

This monitor view is a little bit advanced. However, it generally includes a **CardView**, which contains 3 different types of components such as **TextView**, **ImageView**, and **Switch**. All of these components are designed by **Material Component Android**.

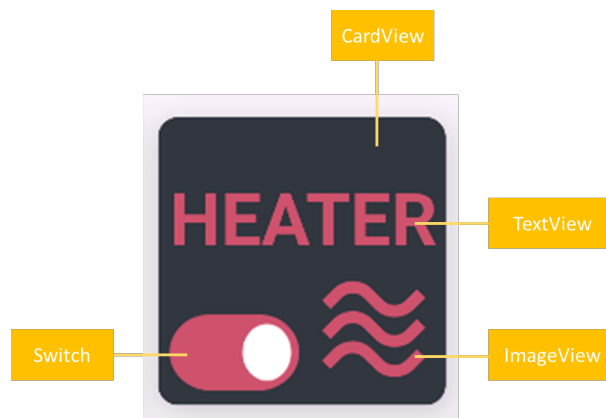


Figure 3: Different parts in CardView

I'm gonna briefly describe: What is Material Component Android?.

Material Components for Android (MDC-Android) is a library of customizable and ready-to-use user interface components based on Google's Material Design guidelines. It provides developers with a set of pre-built UI components and styles that follow the Material Design principles, enabling them to create modern and visually appealing Android applications with consistent design and behavior.

We can access this library via: <https://github.com/material-components/material-components-android>. Follow github instruction: <https://github.com/material-components/material-components-android/blob/master/docs/components/Card.md>

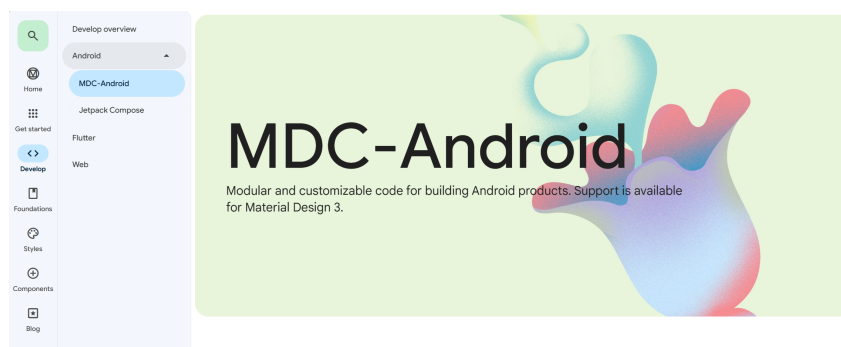


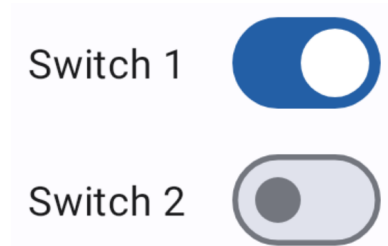
Figure 4: MDC Android

- **CardView:** CardView is a UI component in Android that allows developers to display content in a card-like layout. It provides a flexible container with rounded corners and elevation, creating a visually appealing card appearance. CardView is part of the Material Design library and is commonly used to present information or content in a structured and visually pleasing manner.

To use it, implement tag `<com.google.android.material.card.MaterialCardView>` in the **Activity.xml** file. Follow github instruction: <https://github.com/material-components/material-components-android/blob/master/docs/components/Card.md>

- **Switch** A Switch represents a button with two states, on and off. Switches are most often used on mobile devices to enable and disable options in an options menu. A switch consists of a track and thumb; the thumb moves along the track to indicate its current state.

Follow github instruction: <https://github.com/material-components/material-components-android/blob/master/docs/components/Switch.md>



- **SliderView** allows users to select a value from a continuous range by sliding a thumb along a track. It provides a visually intuitive way for users to adjust settings or input numerical values within a specified range.

In this case, I explicitly implement its range from 16 to 30 with a discrete step 2 unit.

Follow github instruction: <https://github.com/material-components/material-components-android/blob/master/docs/components/Slider.md>



- **ImageView:** that is used to display images or drawables within an app's user interface. It provides a way to visually represent images in various formats, such as bitmap images, vector drawables, or animated graphics.

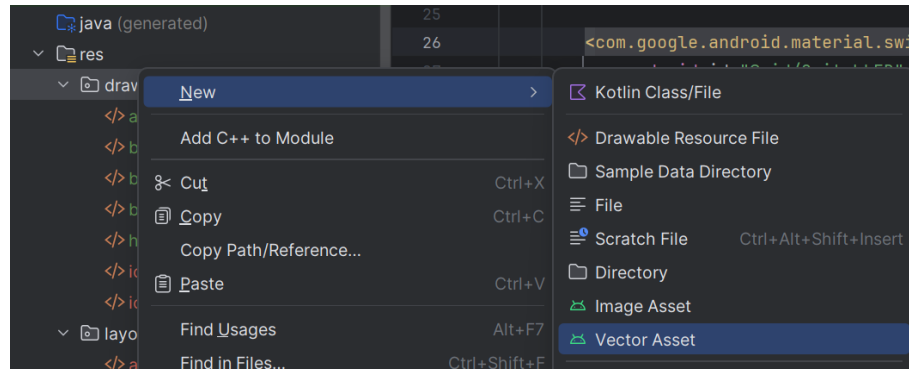
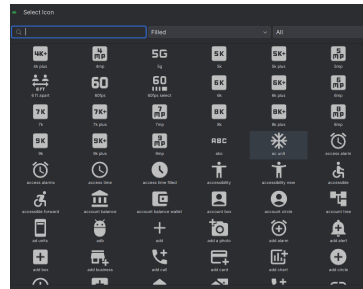


Figure 5: Direction to add Image asset

In this app, I use some popular built-in icons from Android Studio, which are very convenient for newbies.



2.4 MQTT Handler

In order to communicate with the Adafruit MQTT Server, we need to make our Android app operate as an MQTT Client.

First of all, we create a class called MQTTHelper, which implements our MQTT operating Object. There are some vital Properties and Methods.

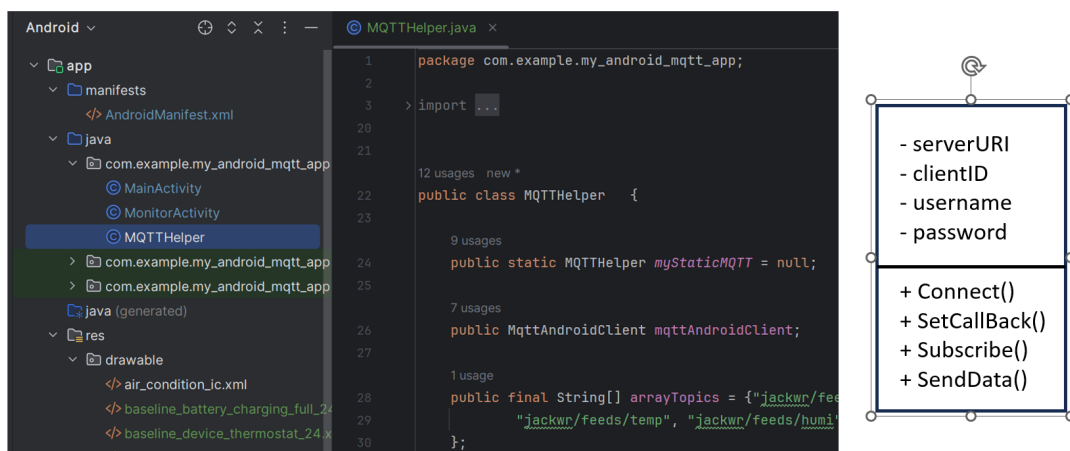


Figure 6: MQTTHelper's Implementation and Abstract

However, we need the MQTT client alive through the session and exist in different **Activity View**.

That is the reason I use **Singleton Pattern** to store the MQTT client, which is a Static Object and will exist throughout the lifecycle of the MQTTHelper Class. Therefore, we can call, override, and implement it in all activities. Notice that the construction method of Singleton Object is slightly different, we must use the method **MQTTHelper.GetMQTTClient()** instead of using **new MQTTHelper()**. The result that whether many times we call Construction method or declare many variable's names, it always return our unique static MQTT Client Object.

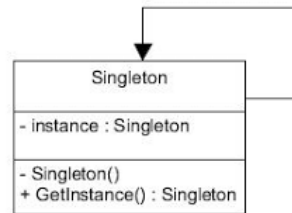
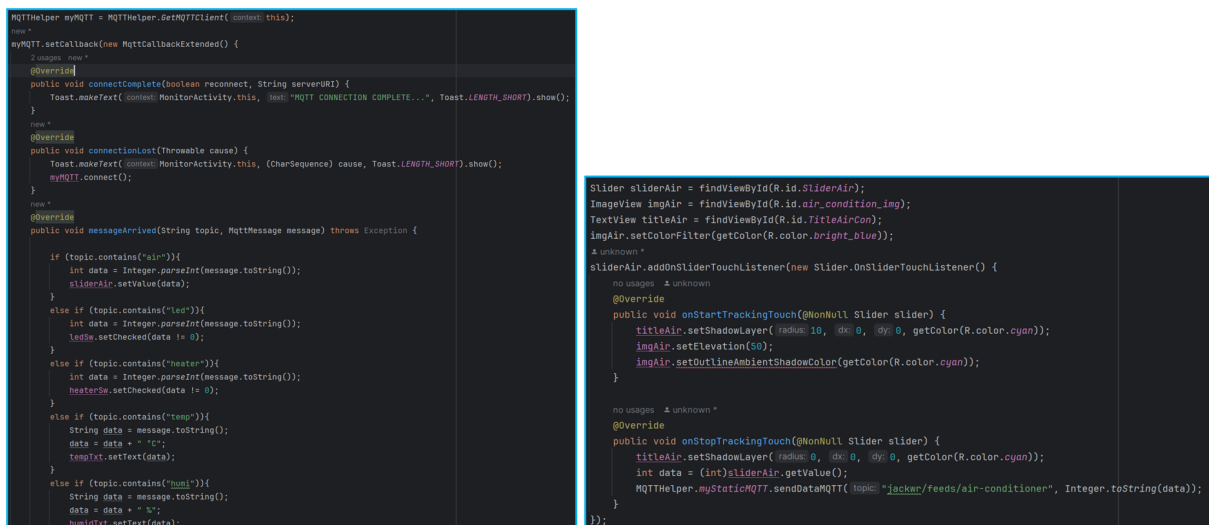


Figure 7: Singleton Pattern

Finally, we can use it and override the **SetCallBack** method to require it to handle the received messages. Moreover, we can call method "SendDataMQTT" to send message to the server.



```

MQTTHelper myMQTT = MQTTHelper.GetMQTTClient(context, this);
new MQTTHelper().setCallBack(new MqttCallbackExtended() {
    @Override
    public void connectComplete(boolean reconnect, String serverURI) {
        Toast.makeText(context, "MQTT CONNECTION COMPLETE...", Toast.LENGTH_SHORT).show();
    }
    @Override
    public void connectionLost(Throwable cause) {
        Toast.makeText(context, "MQTT CONNECTION LOST: " + cause.getMessage(), Toast.LENGTH_SHORT).show();
        myMQTT.connect();
    }
    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception {
        if (topic.contains("air")){
            int data = Integer.parseInt(message.toString());
            sliderAir.setValue(data);
        }
        else if (topic.contains("led")){
            int data = Integer.parseInt(message.toString());
            ledSw.setChecked(data != 0);
        }
        else if (topic.contains("heater")){
            int data = Integer.parseInt(message.toString());
            heaterSw.setChecked(data != 0);
        }
        else if (topic.contains("temp")){
            String data = message.toString();
            data = data + " °C";
            tempTxt.setText(data);
        }
        else if (topic.contains("hum")){
            String data = message.toString();
            data = data + "%";
            humidTxt.setText(data);
        }
    }
});

Slider sliderAir = findViewById(R.id.sliderAir);
ImageView imgAir = findViewById(R.id.air_condition_img);
TextView titleAir = findViewById(R.id.titleAirCon);
imgAir.setColorFilter(getColor(R.color.bright_blue));
sliderAir.addOnSliderTouchListener(new Slider.OnSliderTouchListener() {
    @Override
    public void onStartTrackingTouch(@NonNull Slider slider) {
        titleAir.setShadowLayer(10, 0, 0, 0, getColor(R.color.cyan));
        imgAir.setElevation(50);
        imgAir.setOutlineAmbientShadowColor(getColor(R.color.cyan));
    }
    @Override
    public void onStopTrackingTouch(@NonNull Slider slider) {
        titleAir.setShadowLayer(0, 0, 0, 0, getColor(R.color.cyan));
        int data = (int)sliderAir.getValue();
        MQTTHelper.myStaticMQTT.sendDataMQTT(topic: "jackwr/feeds/air-conditioner", Integer.toString(data));
    }
});
  
```

Construction and Override Callback Method

Using SendData method

Figure 8: Using Singleton Pattern

3 REFERENCES

Video Demo upload here: <https://youtu.be/VxRpGpr3TYc>

Please check my github for more information about source code and building problems

https://github.com/JackWrion/IoT_LAB_2024/tree/main/Lab_04