Internet of Things: Services for a Connected World

Laboratory work 3: IoT, sensors, and MQTT

Last updated: November 2022

Version: 1.2

Table of Contents

1 Introduction

This lab is the continuation of lab 2 of this IoT course. The first lab focused on sensing and actuating things via an IoT gateway. The second lab focused on accessing things via an Android app in Android studios. For this lab, we need to connect the sensor to the Raspberry Pi and configure the scripts for publishing data through MQTT to a broker. Following this, we will create an Android application to read the messages that are published on a topic in the MQTT broker, and we subscribed to the topic.

1.1 Overview

The purpose of this lab is to create an application to access things' status and affect actuation. We have demonstrated during lab one that the gateway collects data from things and forwards it to the application layer. Instead of things, this lab includes sensors connected to a Raspberry pi. The app is used to display data retrieved from the sensors and published through the Raspberry pi.
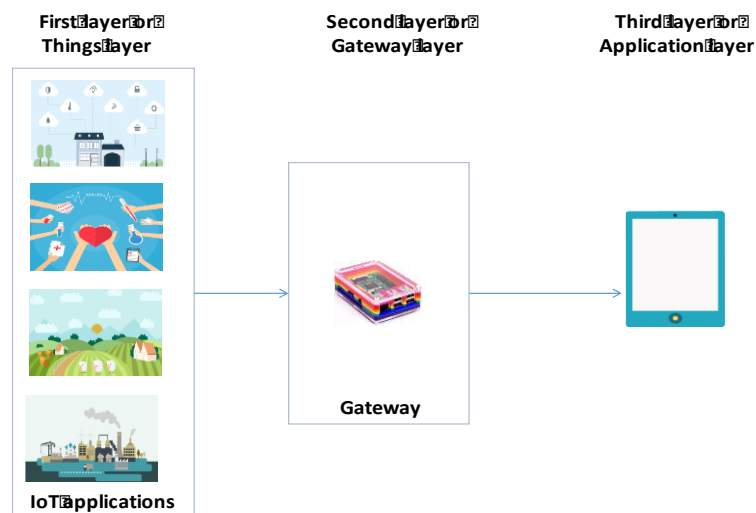
Figure 1: Typical three-layer IoT

1.2 Lab purpose

After the lab, you will:
- ✓ be able to know how to read pinouts for Raspberry Pi and connect a sensor
- ✓ be able to import and use CircuitPython libraries to use with sensors
- ✓ know how to use MQTT to publish messages from Raspberry Pi
- ✓ know how to subscribe to an MQTT topic from Android studios

1.3 Before lab

You should have:
- ▪ read through this entire lab manual
- ▪ glanced through the lecture notes

▪ read about MQTT; see references

2 Background

2.1 Android

Android is a mobile OS developed by Google for touchscreen smart devices such as smartphones and tablets [1]. An android application (app) is a software app running on android-enabled devices. Apps are used to extend the smart device functionalities which are written in the Android software development kit (SDK)- Java and XML are the dominant languages. Once written, users can download and install using the app's APK (Android application packages). For this lab, we will use Android Studio as our SDK and create an APK that can be installed on android-enabled devices.

2.2 Android Studio

Android studio is the fastest and the official integrated development environment (IDE) for android application development. It is freely available to the public, and the first stable version was released in December 2014. It is available for Windows, Mac OS X, and Linux, and it replaced Eclipse ADT for native android app development [2].

2.3 Running an android app

Once we have written an app, we can run the app either on an emulator or on a real device. Please refer to reference [3, 4, 5] to learn more about it (if interested).
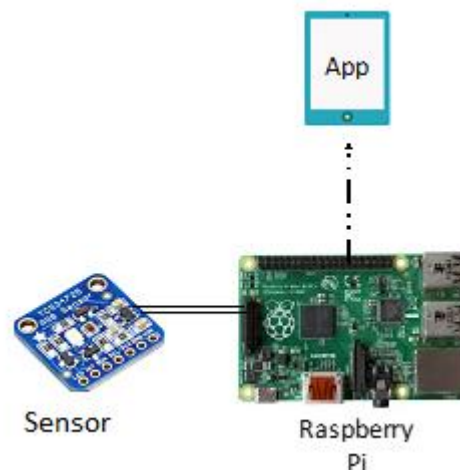

Figure 2: Lab Scenario

2.4 Tools for the lab
For this lab, you are provided with the following tools and items:
- A Raspberry Pi 4, with static IP, network access, and where SSH has been enabled
- Sensors, with soldered headers: to sense light, color and/or proximity
- Jumper wires, female to female, to connect the sensors to Raspberry pi

- CircuitPython libraries for the sensors
- MQTT library: Paho MQTT
- Template code
- PuTTY
- Android Studios

In preparation for the lab, the following has been done:

Raspberry Pi
Rasbian has been installed, and SSH has been enabled. To work in DSV, each Raspberry Pi has been given a static IP.

Sensors
In preparation for this lab, each sensor has had header pins added through soldering. The three types of sensors used for this lab are manufactured and designed by the Adafruit industries [6].

2.5 MQTT

MQTT (MQ Telemetry Transport) was invented in 1999, designed as a lightweight protocol to minimize the network bandwidth required for messaging. It uses a publish/ subscribe structure that publishers publish messages to a broker on a chosen topic. Subscribers can subscribe to topics and retrieve messages. Since MQTT is a lightweight protocol, it has been used in M2M (machine-to-machine) communication, as well as for IoT. See [7], [8] & [9] for more information on MQTT.

In this lab, the android app will work as the subscriber, test.mosquitto.org [10] as the broker, and the Raspberry Pi will be the publisher.

2.6 The sensors

For this lab, three types of sensors are provided. They are all made by Adafruit industries and have easy-to-use CircuitPython libraries available, which can be used for Raspberry Pi and Arduino. Read more about CircuitPython in [11].

TSL2591: high dynamic range light sensor
This sensor can detect light ranges from 0.000118 lux up to 88 000 lux. It can also measure different values for infrared, full-spectrum, or human-visible light. This type of sensor is suitable for when you need an accurate reading of light or luminosity. Read more about this sensor at [12].

VCNL4010: Proximity and light sensor
This sensor can sense small-distance proximity, preferable within the range of 10-150mm. Additionally, this sensor can read light intensity, although not as accurate or as wide range as TSL2591. This kind of proximity sensor is good for detecting very close range, such as when something is moving nearby or detecting collisions. Read more about this sensor at [13].

TCS34725: Color sensor

This sensor can read color in the RGB format (0 for low intensity of a color, 255 for high), color temperature in Kelvin, and light intensity (although, according to Adafruit, not that accurate). It also has a LED connected to add light to the thing you are trying to read colors from them. Read more about this sensor at [14].

3 Laboratory

As depicted in figure 2, the lab scenario is to familiarize you with MQTT and use the values in an application. Compared to previous labs in this course, the data from the sensors will be directly read through the Raspberry Pi. This time, the Raspberry Pi is used to both collect and interpret the data from the sensors and send them to an MQTT broker, which we can subscribe to in the Android app.

3.1 Raspberry Pi & connecting the sensor
For this lab, you need to prepare the Raspberry for input through I2C and SPI, connect the sensor, and edit the script files. Read more about I2C in [15] and SPI in [16].

3.1.1 Access Raspberry Pi through PuTTY
Much like the previous lab, we are using SSH through PuTTY to access the Raspberry Pi.

- To connect to a device through SSH (Secure shell), open PuTTY.
- In the field for Host Name, enter the IP address of the Raspberry Pi you want to connect to it. If you are using one of the provided Raspberry pis for the lab, you can find the IP address on the device.
- The port number can be left as 22, and make sure "SSH" is selected.
- When ready, click Open to initialize the connection.
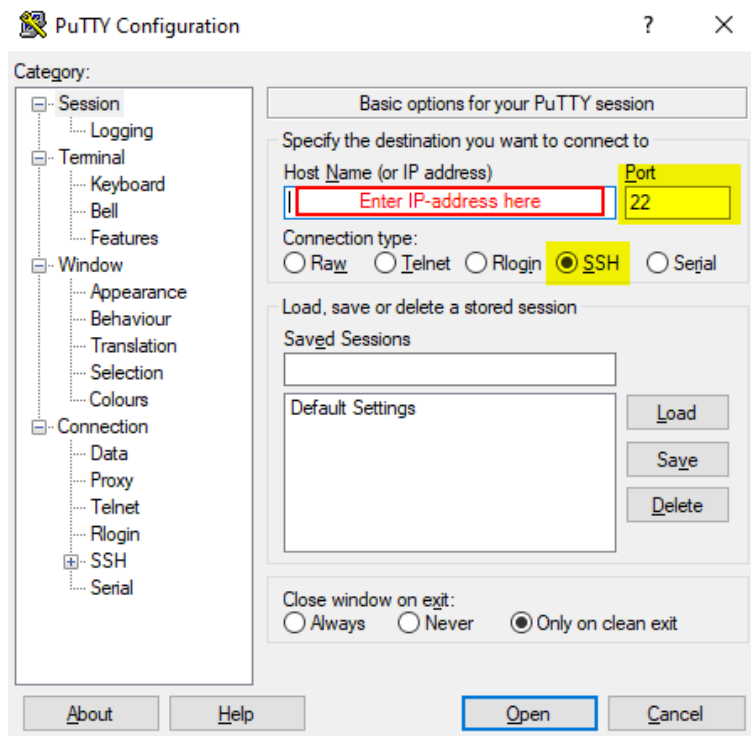
- A new window will open, with a prompt for username and password.
- Use the following credentials:

Username: *pi*
Password: *IoT@2021*

3.1.2 Enable I2C and SPI
The sensors for this lab make use of I2C and SPI. To connect these, we need to configure the Raspberry pi to allow this.

Enable I2C
Run the following two commands:

```
sudo apt-get install -y python-smbus
sudo apt-get install -y i2c-tools
```

In the terminal type:

```
sudo raspi-config
```

To open the software configuration tool for the pi.

Within the Raspberry software configuration tool, enter *5 Interfacing options* and select *P5 I2C*. In the prompt, select *<Yes>* to enable I2C.
Again, go to *5 Interfacing options* and tap enter to select.
Select *P4 SPI* and press enter. In the new prompt, select *<Yes>* to enable SPI.

Now, reboot the Raspberry pi (by doing this, you might need to reconnect to the raspberry pi with putty) by typing

```
sudo reboot
```

In the terminal/PuTTY, when typing

```
sudo i2cdetect -y 1
```

you should see

Now it is time to connect the sensors. Close PuTTy and disconnect the Raspberry pi from its power source before connecting the sensors.

3.1.3 Connecting the sensors
Please disconnect the Raspberry pi from any power source before connecting the sensors!

The sensors used for this lab are described in section *2.6 (The sensors)*. They all connect to the following pinouts on the Raspberry pi: 3.3v, ground, SDA, and SCL.
To see an interactive pinout diagram, visit https://pinout.xyz/.

A simplified version of the pinout can be seen using the `pinout` command in the terminal:

Raspberry Pi GPIO BCM numbering

Pinout diagram from pinout.xyz. Ground pins are colored black.

The above diagram shows that SDA is the same pin as GPIO2, and SCL is the same pin as GPIO3.

Connecting a single sensor
For connecting a single sensor to the Raspberry, simply connect the following:

Raspberry Pi pin	Sensor pin
3.3v	Vin
Ground (GND)	GND
SDA (GPIO2)	SDA
SCL (GPIO3)	SCL
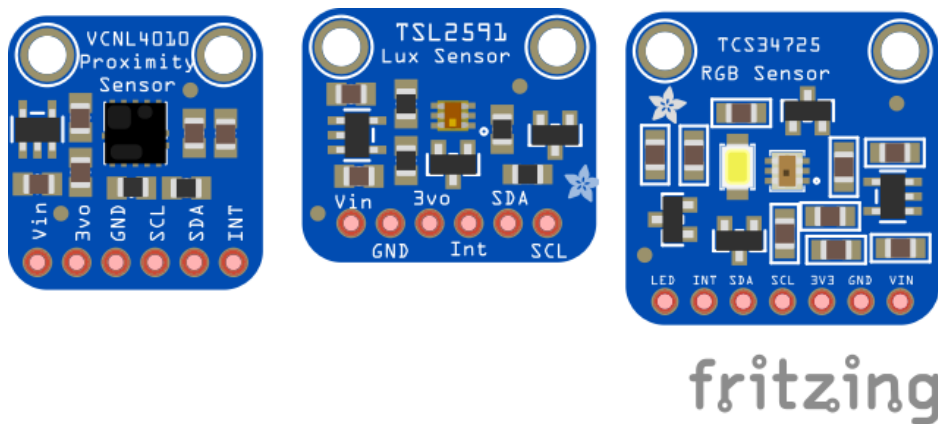
Connecting two sensors
It is possible to connect more than one sensor; however, that requires a breadboard as there are not enough pins of the kinds we use in the Raspberry Pi's 40 pins.
By using a breadboard, we can "chain" the sensors together.

On the following page, the diagram illustrates how two sensors can be connected to the Raspberry pi.

- Red: *Raspberry pi 3.3v* to the row following the red line on the breadboard. This allows all sources connected to the row following the red line to connect to the Raspberry Pis 3.3v.
- Black: *Raspberry Pi ground pin* to the row along the blue line. This allows all sources connected to the row below the blue line to connect to the ground.

- Blue: *Raspberry Pi SDA (GPIO2)* to the same column on the breadboard.
- Yellow: *Raspberry pi SCL (GPIO3)* to the column on the breadboard.
- Red: *On the board*, a cable from the row following the red line to the same column as the sensor's Vin pin. One for each sensor.
- Black: *On the board*, a cable from the row below the blue line connects to the same column as the sensor's ground pin. One for each sensor.
- Blue: *On the board*, a cable from the same column as the left sensors SDA pin to the right sensors SDA pin.
- Yellow: *on the board*, a cable connecting the column at the left sensors SCL to the column at the right sensors SCL pin.
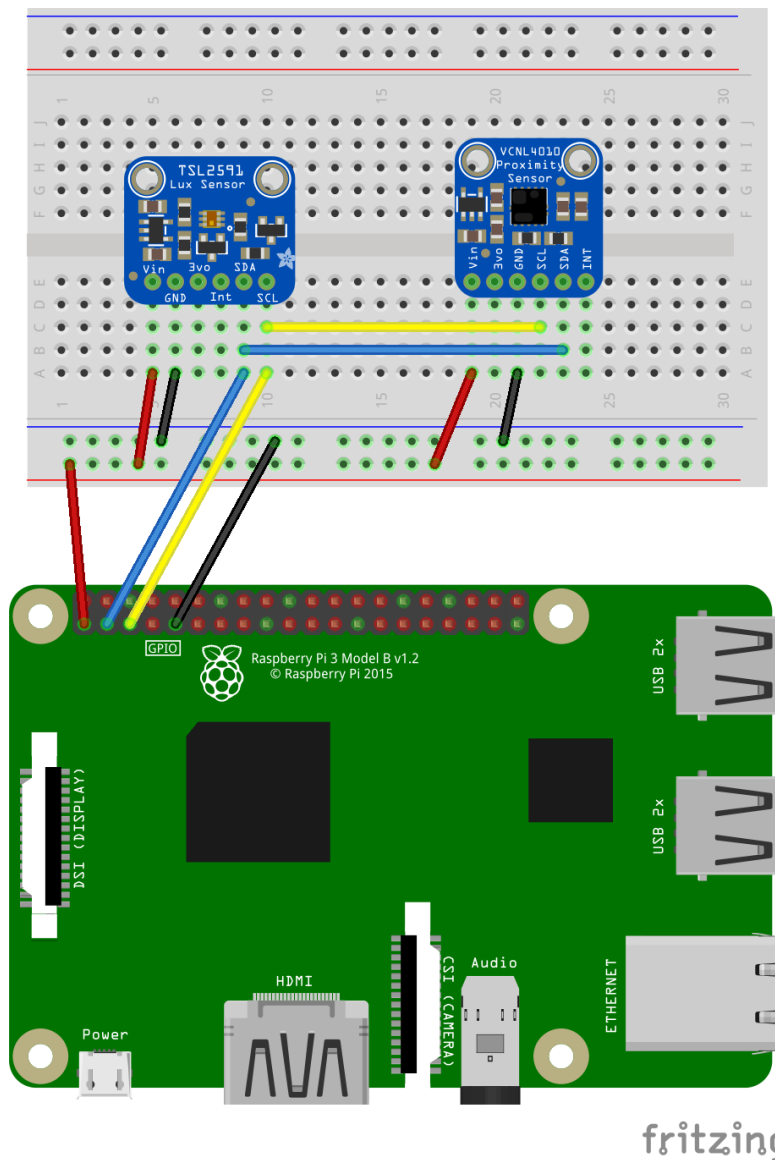

Diagram of TSL2541 and VCNL4010 connected to a Raspberry Pi. Diagram made in fritzing[17].

Please note that the sensors might not have the same order of the pins as shown in the diagram above.

When you have done with the wiring, connect the Pi to the power source again, and use PuTTy to connect to the Pi through SSH.

To check if the Raspberry Pi can find the sensors, use the command

```
sudo i2cdetect -y 1
```

If the sensor(s) has been successfully connected, they will now show in the table.

The TSL2591 was successfully found by the Raspberry Pi.

3.1.4 Install CircuitPython

The sensors used in this lab are all manufactured by Adafruit, and all have easy-to-use CircuitPython libraries available.

However, to use the libraries, we first need to install Circuit python. The official guide on how to install CircuitPython on Raspberry Pi can be found on Adafruit.com [18].

Check and install the Raspberry Pi GPIO library:

```
pip3 install RPI.GPIO
```

Now, install adafruit_blinka:

```
pip3 install adafruit-blinka
```

All done, you can now move forward to download the specific libraries for each sensor in the next section.

3.1.5 Download code libraries and modify the scripts

For this lab, the script will be slightly different from the previous as we use sensors directly connected to the Pi and use the MQTT protocol to connect to the Android studios app.

Two templates code are provided in nextiLearn for this lab: *mqtt-template-lab3.py* and *activity_main_template.xml*. Additional code can be found within this document.

To make use of the template script, you need to install the CircuitPython libraries for each sensor and Paho MQTT [19].

Install Paho MQTT with the command:

```
pip3 install paho-mqtt
```

To download and install the library for **TSL2591** [20]:

```
pip3 install adafruit-circuitpython-tsl2591
```

To download and install the library for **VCNL4010** [21]:

```
sudo pip3 install adafruit-circuitpython-vcnl4010
```

To download and install the library for **TCS34725** [22]:

```
pip3 install adafruit-circuitpython-tcs34725
```

Please note that mqtt-template-lab3.py is designed to send data from one sensor. You need to modify the script to publish data from several sensors at once. You can either make use of the script as it is (following the below instructions, and send data from one sensor at a time) or, with modifications, send data from two sensors within the same call, and then split the data in the android app. Additionally, you can publish to several topics from the Pi and subscribe to several topics in the android app. See Extended instructions in the end of this manual for more information on how to send data from several sensors.

The provided script *mqtt-template-lab3.py* have placeholders for the broker and MQTT topic.

You will need to modify the values for broker, pub_topic, and uncomment the rows relating to the sensor(s) you are using.

Modifying the scripts might be easier in Notepad++ or Notepad rather than through nano in the terminal window.

On the Raspberry pi, create a file and open it in nano:

```
nano filename.py
```

- Copy&paste the code from mqtt-template-lab3.py.
- Change the value for the broker to test.mosquitto.org.
- Change the value of pub_topic to a topic name. For example, *iotlab/your-initials/sensors*. Or the name of your favorite pet.
- Uncomment the rows for the sensor(s) you connected to the pi.
- Save the file in nano by ctrl+o and exit nano with ctrl+x.

The CircuitPython libraries are all based on python3, so to use the script, we use

```
python3 filename.py.
```

Call the mqtt-script you just edited to make sure you do not get any errors.

If there is any error please run the following commands and then try again to call mqtt-script

```
sudo apt-get update
sudo apt-get upgrade
sudo pip3 install --upgrade setuptools
```

3.2 Android Studios app

For this lab, we are creating a simple app in Android studio. This app will feature a couple of text views to show the sensors' data and a button to visualize values.

Template code is provided for the lab, but feel free to modify the layout, colors or add new elements as you wish.

3.2.1 Create a project in Android studios

Do the following steps:

- Open Android Studio
- Start a new Android Studio project or Go to: File menu –> New -> New Project
- In Choose your project, select **Empty Views Activity**
- You will come to "Configure your new project" window o Application Name: A name, e.g., IoTLab3App
  - o Choose Java as the language
  - o Leave rest as it is and press **Next**
- Now choose the configuration, you can choose which minimum SDK you want. Leave as it is for this lab and press Next
- Wait until the project is created for you.

3.2.2 Changing in the project XML files (Layout and values)

As we have sensors that sense different data, we need to have something that can visualize the values. We begin with creating a User interface that can show the values. For this, we will edit a layout XML file (in app -> res -> layout) which has at least these settings.

To save time, a layout XML file for UI has already been created for this lab which can be downloaded from nextiLearn, the file *activity_main_template.xml*.

Change the content of the layout file, activity_main.xml, by selecting the text view. There, replace the existing code with the code provided in *activity_main_template.xml*.

We further will need to change the colors.xml file in app -> res -> values to make it prettier. Add the followings to the **colors.xml** file:

```
<color name="colorPrimary">#FFC107</color> <color
name="colorPrimaryDark">#DAA407</color> <color
name="colorAccent">#D81B60</color> <color
name="black_alpha">#000000</color> <color
name="text_box_background">#E7E7E7</color> <color
name="blue_text">#0166ff</color> <color
name="title_bar_color">#EB164E</color> <color
name="activitybackground">#FFF5B8</color> <color
name="activitybutton">#FFE06E</color> <color
name="activityfont">#3f3f3f</color>
```

Go back to src -> main -> res -> layout -> actitvity_main.xml. The view should now show the text views for RGB. Lux value and proximity as well as a button.

Feel free to edit the values within activity_main.xml or colors.xml to your liking. However, note that the following instructions will refer to the names given to each element.

3.2.3 Access UI elements and add listeners

Now we have to edit the Activity class to initiate activity for the fields we created in our layout file. Therefore, we have to use things' status in MainActivity's onCreate method (you can read more about onCreate in [23]). In app -> java -> com.example.<projectname> MainActivity.java add the following code:

Outside onCreate method

```
private TextView txv_rgb;
private TextView txv_light;
private TextView txv_proximity;
private Button btn_color;
```

Inside onCreate method

Make sure that setContentView is set to the same layout name as the layout.xml-file you're using. Below super.onCreate(savedInstanceState) and setcontentView(R.layout.activity_man); add the following to initiate the elements:

```
txv_rgb = (TextView) findViewById(R.id.txv_rgbValue);
txv_light = (TextView) findViewById(R.id.txv_lightValue);
txv_proximity = (TextView) findViewById(R.id.txv_proximityValue);
btn_color = (Button) findViewById(R.id.btnColor);
```

3.2.4 Import libraries, add permissions, and set values for MQTT

Import libraries:

In lab2, we imported a .jar-file to enable SSH. In this lab, we instead use MQTT to fetch data from the Raspberry Pi. There are several libraries for MQTT in Java. For this lab, we're using Paho Android Service [24].

In Gradle scripts -> settings.gradle.kts (Project Settings) add the below code to dependencyResolutionManagement, within repositories:

```
maven {
url = uri(https://repo.eclipse.org/content/repositories/paho-snapshots/)
}
```

In gradle scripts > build.gradle.kts (Module: app) change the targetsdk to 30 and add the comment above it.

Also add the below code within *dependencies* in the same file:

```
Implementation("org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0")
implementation("org.eclipse.paho:org.eclipse.paho.android.service:1.1.1")
implementation("androidx.legacy:legacy-support-v4:1.0.0")
```

In gradle scripts -> gradle.properties (Project Properties) add the following line.

```
android.enableJetifier=true
```

Sync the project gradle. Click on the small elephant icon with an arrow in the upper right corner to Sync Project with Gradle Files or on the alert that appears in connection to the added dependencies.

Add permissions:

Much like in lab 2 we need to add permissions to access the internet in the Android manifest. Read more about android manifest and permissions in [25].

To do so, add the following lines to the AndroidManifest.xml file (outside the <application> tags), in app -> manifests -> AndroidManifest.xml.

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Inside the <application> tags add the following line

```
<service android:name="org.eclipse.paho.android.service.MqttService" />
```

After adding the permissions, you might need to sync gradle again.

Values for MQTT:

Go back to MainActivity.java. In the class MainActivity, outside of onCreate add:

```java
private MqttAndroidClient client;
private static final String SERVER_URI = "tcp://test.mosquitto.org:1883";
private static final String TAG = "MainActivity";
```

This will cause a prompt from the IDE asking if you want to import Paho MQTT, click yes.
In the above lines, for SERVER_URI we are setting the value for the broker.
In our case, we are using test.moquitto.org and port 1883.

TAG allows us to log messages, in case of errors or events.

Outside of onCreate, add the following methods: connect and subscribe.

Connect:
Here, we initialize a client and create a connection to the set broker declared in SERVER_URI.

```java
private void connect(){

        String clientId = MqttClient.generateClientId();

        client = new MqttAndroidClient(this.getApplicationContext(), SERVER_URI, clientId);

        try {

                IMqttToken token = client.connect();

                token.setActionCallback(new IMqttActionListener() {

                @Override

                        public void onSuccess(IMqttToken asyncActionToken) {

                        // We are connected

                        Log.d(TAG, "onSuccess");

                        System.out.println(TAG + " Success. Connected to " + SERVER_URI);

                        }

                @Override

                        public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

                        // Something went wrong e.g. connection timeout or firewall problems

                        Log.d(TAG, "onFailure");

                        System.out.println(TAG + " Oh no! Failed to connect to " + SERVER_URI);

                        }

                });

        } catch (MqttException e) {

                e.printStackTrace();

        }

}
```

To make use of the method, add connect() within onCreate(), below the initializations of the layout elements (text views and buttons).

Subscribe:

In subscribe, you need to set the topic to the same one you set in the script for Raspberry Pi. The subscribe method sets a subscription to the set topic, using the client from connect().

```java
private void subscribe(String topicToSubscribe) {
        final String topic = topicToSubscribe;
        int qos = 1;
        try {
                IMqttToken subToken = client.subscribe(topic, qos);
                subToken.setActionCallback(new IMqttActionListener() {
                        @Override
                        public void onSuccess(IMqttToken asyncActionToken) {
                        System.out.println("Subscription successful to topic: " + topic);
                        }

                        @Override
                        public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                        System.out.println("Failed to subscribe to topic: " + topic);
                        // The subscription could not be performed, maybe the user was not
                        // authorized to subscribe on the specified topic e.g. using wildcards
                        }
                });
        } catch (MqttException e) {
        e.printStackTrace();
        }
}
```

Finally, we set a callback to the client. In the code, we use the subscribe() method to set up a subscription, and again if we happen to reconnect to the connection. It also allows us to decide what happens when a new message is fetched on the set topic.

In onCreate, add a call to the connect method to initiate a connection at the start of the app. Then, add de following to use the callback (and don't forget to set the name of your topic in subscribe()):

```java
client.setCallback(new MqttCallbackExtended() {
    @Override
    public void connectComplete(boolean reconnect, String serverURI) {

        if (reconnect) {
            System.out.println("Reconnected to : " + serverURI);
            // Re-subscribe as we lost it due to new session
            subscribe(" YOUR-TOPIC");
        } else {
            System.out.println("Connected to: " + serverURI);
            subscribe(" YOUR-TOPIC");
        }
    }

    @Override
    public void connectionLost(Throwable cause) {
        System.out.println("The Connection was lost.");
    }

    @Override
    public void messageArrived(String topic, MqttMessage message) throws
Exception {
        String newMessage = new String(message.getPayload());
        System.out.println("Incoming message: " + newMessage);

        /* add code here to interact with elements
        (text views, buttons)
        using data from newMessage
         */

    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {

    }
});
```

In messageArrived, add code to make use of the information. For example, change the value of the corresponding text view or the button's color depending on data from the MQTT message.

Much like previous labs, you might need to use string.replace, string.split or create substrings to make a pretty output.
If you opted for the color sensor, you might get better values if you amplify them in the code (e.g., by multiplying the values).

With some modifications, you can send data from several sensors at once. Either by adding data from two sensors to the same published message, splitting the string in the Android app, or publishing to different topics.

Please do try on your own before heading to the extended instructions.

3.3. Completing the lab

You have completed the lab once you have shown that you successfully can fetch data from your sensors and display those in your app.

When done, do not forget to have a look at the questions for the lab report.

Before leaving the lab, please remove the files from the raspberry pi.
To remove a file in Raspberry pi through SSH, type the following command:

```
rm filename.filetype
```

To remove a folder with all its content, use

```
rm -rf directoryname
```

4 Lab report

After the end of the lab work, you should hand over a lab report and submit it on the iLearn2. In the report, you should write the following (Numerical values in parenthesis indicate achievable points for the answer to each question):

1. Briefly describe MQTT. Include concepts such as a broker, publisher, and subscriber. (2)
2. In a few sentences, describe the benefits of MQTT when working with IoT. Discuss its benefits both for large-scale operations and for smaller projects, such as this lab. (2)
3. Which values did you display on your app? (attach a screenshot) (1)

To pass the lab, you **must** obtain *3.5 points* out of possible **5 points**.
All questions are **mandatory** to answer.
You should submit the lab report via iLearn2 **within two weeks** from the scheduled lab day.
<u>No submission will be entertained unless you perform the actual lab during scheduled lab time.</u>

5 References

[1] Android, https://en.wikipedia.org/wiki/Android_(operating_system)

[2] Android Studio, https://en.wikipedia.org/wiki/Android_Studio

[3] Running your app, https://developer.android.com/training/basics/firstapp/running-app.html

[4] Android emulator, https://developer.android.com/studio/run/emulator.html

[5] Building Your First App,
https://developer.android.com/training/basics/firstapp/index.html

[6] Adafruit: about us, https://www.adafruit.com/about

[7] Wikipedia: MQTT, https://en.wikipedia.org/wiki/MQTT

[8] MQTT.org, http://mqtt.org/

[9] Mosquitto.org: MQTT man page, http://mosquitto.org/man/mqtt-7.html

[10] Mosquitto.org: test, http://test.mosquitto.org/

[11] Circuit Python, https://circuitpython.org/

[12] Adafruit learn: TSL2591, https://learn.adafruit.com/adafruit-tsl2591

[13] Adafruit learn: VCNL4010, https://learn.adafruit.com/using-vcnl4010-proximity-sensor/

[14] Adafruit learn: TCS34725, https://learn.adafruit.com/adafruit-color-sensors

[15] Sparkfun: I2C, https://learn.sparkfun.com/tutorials/i2c/all

[16] Sparkfun: Serial periphral interface (SPI), https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[17] Fritzing, http://fritzing.org/

[18] Adafruit: installing CircuitPython libraries on Raspberry Pi,
https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi

[19] PyPi: paho-mqtt, https://pypi.org/project/paho-mqtt/

[20]Github: Adafruit CircuitPython TSL2591,
https://github.com/adafruit/Adafruit_CircuitPython_TSL2591

[21] Github: Adafruit CircuitPython VCNL4010,
https://github.com/adafruit/Adafruit_CircuitPython_VCNL4010

[22] Github: Adafruit CircuitPython TCS34725,
https://github.com/adafruit/Adafruit_CircuitPython_TCS34725

[23] onCreate,
https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)

[24] Eclipse: Eclipse Paho Android service,
https://www.eclipse.org/paho/clients/android/

[25] Android: Manifest.permission,
https://developer.android.com/reference/android/Manifest.permission

6 Extended instructions

6.1. Publishing data from several sensors in one message
In the MQTT script in the Pi, modify the code:

- Uncomment both sensors used
- Rename one of the sensors from simply sensor to another name. E.g., sensor2.
- In the corresponding function for fetching data for the second sensor, change the sensor's name to the one you added. If you uncommented the proximity sensor and renamed it sensor2, in get_proximity, change it to
  ```
  proximity = sensor2.proximity
  ```
- In the while-loop that publishes the data, call the second sensor function and add it to data_to_send. Example with light and proximity sensor:
  ```
  data_to_send = get_lux() + "." + get_proximity()
  ```

  Then in Android studios, make sure to split the newMessage and then set the corresponding values to the right filed/button.

6.2 Publishing data to several topics
In the MQTT script in the Pi, modify the code:

- Uncomment both sensors used
- Rename one of the sensors from simply sensor to another name e.g., sensor2.
- In the corresponding function for fetching data for the second sensor, change the sensor's name to the one you added. If you uncommented the proximity sensor and renamed it sensor2, in get_proximity, change it to
  ```
  proximity = sensor2.proximity
  ``` to
- In the while-loop, call another client.publish but change pub_topic to be the name of the second topic, and data_to_send to the data you want to send to the topic. Continuing with the example above:
  ```
  client.publish("iotlab3/proximityexample", str(get_proximity()))
  ```

In android studios:

- In connectComplete, within client.setCallback call subscribe() with your second topic:
  ```
  subscribe("iotlab3/proximityexample");
  ```
- In messageArrived, within clientsetCallback add an if-statement to check from which topic the message was sent, and what to to with the message, depending on topic:

```
if (topic.equals("YOUR-FIRST-TOPIC")) {
    String newMessage = new String(message.getPayload());
    System.out.println("Incoming message: " + newMessage);
            /* Do something */
```

```java
    } else if (topic.equals("YOUR-SECOND-TOPIC")) {
        String newMessage = new String(message.getPayload());
        System.out.println("Incoming message: " + newMessage);
            /* Do something */


}
```