Internet of Things: Services for a Connected World

Laboratory work 2: Accessing IoT things via an Android application
Last updated: November 2021
Version: 1.4

Table of Contents

1 Introduction

This lab is the continuation of lab 1 of this IoT course. The first lab focused on sensing and actuating things via an IoT gateway; we will build upon this lab. At the beginning, we need to add the scripts to the Raspberry Pi to access values from Telldus Live. Following this, we will create an Android application to access the things' status and affect actuation.

1.1 Overview

The purpose of this lab is to create an application to access things' status and affect actuators. As demonstrated during lab one, the gateway collects data from things and forwards it to the application layer. The collected data from the things via the gateway will be forwarded to an android app (see figure 1). The app will let the user check the current status of things and alter the situation by actuating in an environment.

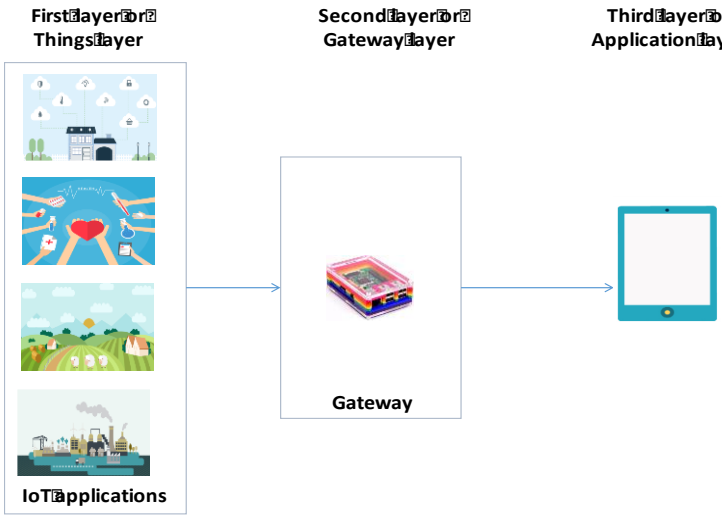


Figure 1: Typical three-layer IoT

1.2 Lab purpose

After the lab, you will:

- ✓ be able to know how to create an android app using Android Studio
- ✓ be able to know how to access raspberry pi, an IoT gateway, from the android app
- ✓ be able to know how to showcase things' status on the app
- ✓ be able to know how to affect actuators remotely.

1.3 Before lab

You should have:

- read through this entire lab manual
- glanced through the lecture notes
- read about android development (highly recommended), 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 for 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).

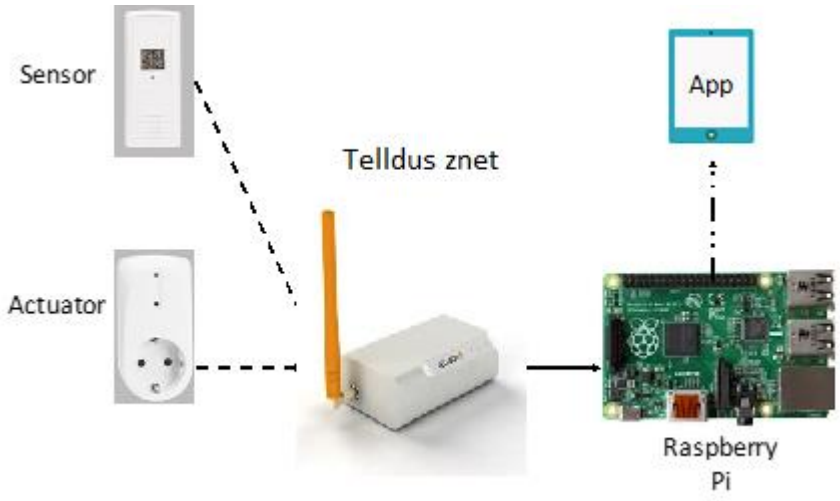


Figure 2: Lab Scenario

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
- Access to temperature and humidity sensors through Telldus Live API
- An actuator, connected to Telldus Live
- A lamp

- 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.

Telldus Live, sensors and actuators
We use the same sensors and actuators as in lab 1.

3 Laboratory

As depicted in figure 2, the lab scenario is to familiarize with how to access things' status using an application. We will further create an android application to achieve our goal. We will take advantage of things such as temperature sensor(s) and actuator(s). The sensor values and actuator status can be obtained as shown in lab 1. Our objective in this lab will be to fetch these values using an android app.

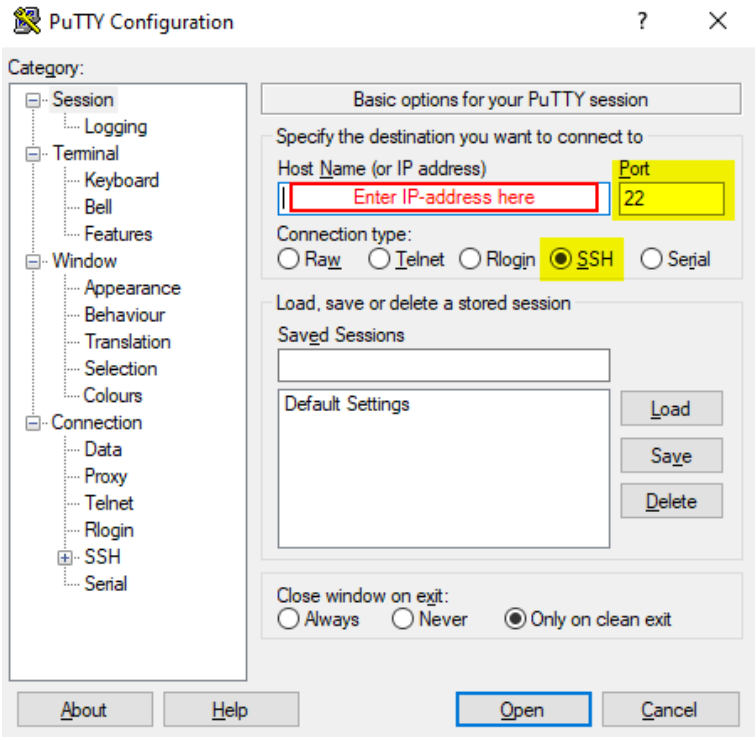
3.1 Raspberry Pi & scripts

Prepare the scripts for the Raspberry Pi to access data through Telldus Live.

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 with. If you're 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 Create and modify the scripts

For this lab, you will use similar scripts as for the previous lab.
Provided for this lab are *listdevices.py*, *listsensorsandvalues,* and *turnoffdevice.py*.
Download the scripts from iLearn.

The provided scripts have placeholders for public and private keys. In each script, you will need to modify the values to those provided for the lab. Ask the workshop leader for the values.
Modifying the scripts might be easier in Notepad++ or Notepad rather than through nano in the terminal window.

In PuTTY, in the terminal for the Raspberry pi, create four new files in the home directory. One for listing all the devices, one for listing the sensors and their values, one for turning off an actuator and lastly one to turn on an actuator.
Each file can be created by *touch filename.filetype*,
for example, by `touch listdevices.py`

Open the files through *nano filename*
(using the example above: `nano listdevices.py`)

There add the provided code and modify the values for *pubkey, privkey, token, secret*. This needs to be done for each file.
Save in nano by ctrl+o and ctrl+x to exit nano.

For the script to turn on the actuator, you may use the code from *turnoffdevice.py* and modify the URL in the GET-request.

3.1.3 Get the ID for your actuator

To turn on or turn off an actuator, you need to provide the ID of that particular actuator. Each actuator got a name on it; however, you need to map this to the ID in the list of devices.

Run the script to list all the devices using *python filename.py.*
Example: `python listdevices.py`

Make a note of the ID of your actuator.

In both scripts for turning off and on the device, using nano, initialize the parameter "id" with the one you wrote down. Save the files.

Now, make sure the scripts for turning on and off the actuator and listing all the sensors and their values work.

3.1.4 Modify the output from list sensors

Run the script provided through listsensorsandvalues.py. For this lab, you only need data from one of the sensors. Modify the script to only output the temperature value from the JSON response to make it easier to process.

The variable responseData holds the values, and specific values can be accessed as values from a dictionary [8]. In responseData, the list of sensors is within an array, and each sensor holds the value for the measured temperature.
Try to modify the script, so the only thing that will be printed is the temp value.

Are you having trouble with extracting the temp value? Check *extended instructions: get specific values* for instructions.

3.2 Android Studios app

For this lab, we are creating a simple app in Android studios.
This app will feature a text view to show data from the temperature sensor, a switch toggle to turn on and off an actuator, and a button to fetch data from the temperature sensor.

3.2.1 Create a project in Android studios

Do the following steps:

▪ Open Android Studio

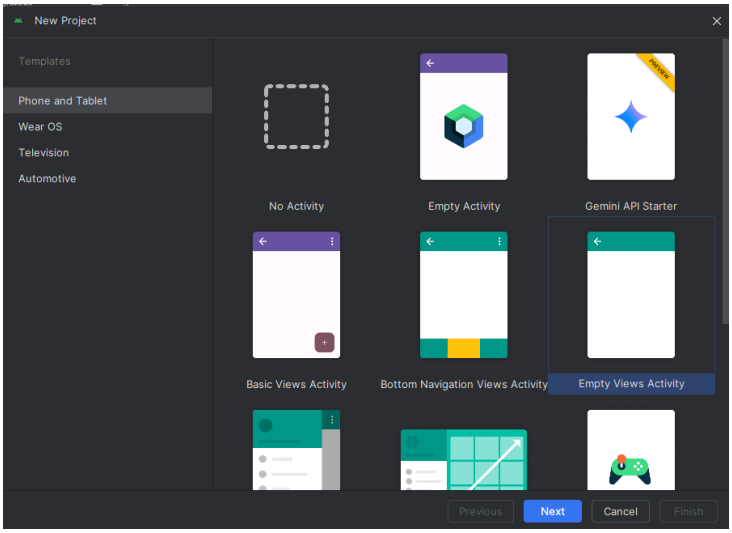
▪ Start a new Android Studio project or go to: File menu –> New -> New Project

▪ Select the template called **Empty Views Activity**

▪ Click **"Next".** You will come to the Activity window

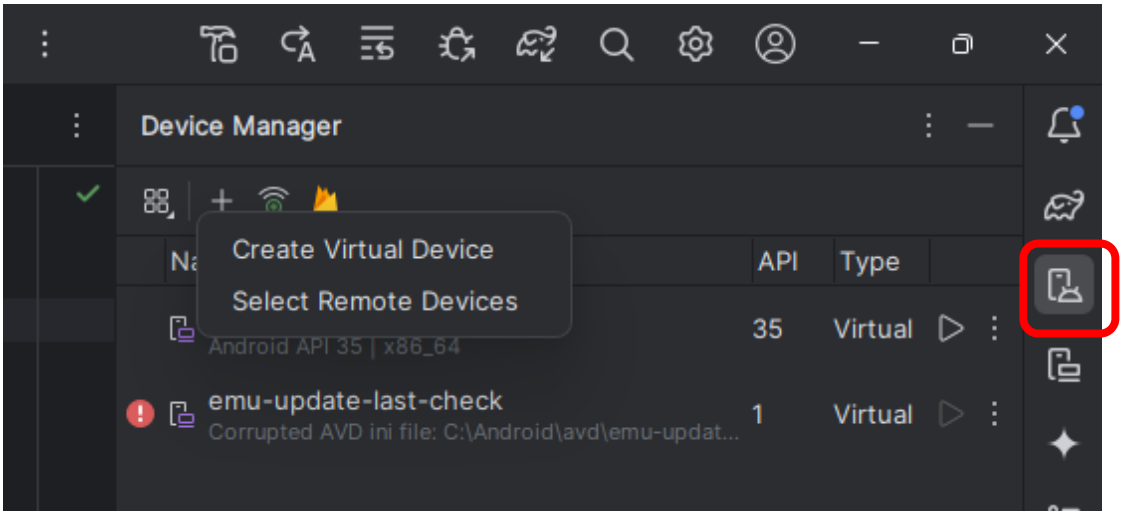
- Application Name: A name, e.g., IoTLabApp
- Choose Java as a language
- Leave rest as it is and press Finish

▪ Wait until the project is created for you.



Set up a virtual device

Suppose you have not used Android studios before. In that case, you might need to configure the AVD manager before you can use the Android emulator. On the right side, there is a menu click on the device manager tab as mentioned in the screenshot below. Click **"+"** and Create Virtual Device.



Pick the device you want to use. For this lab, feel free to pick any phone or tablet device. Click **Next**.

In the next window, select a system image and click on **Download** to download it. Read and accept the license agreement, click **Next**, and wait for it to download. When it is downloaded, select the system image and click next. Give the virtual device a name if you want, and click **Finish**.

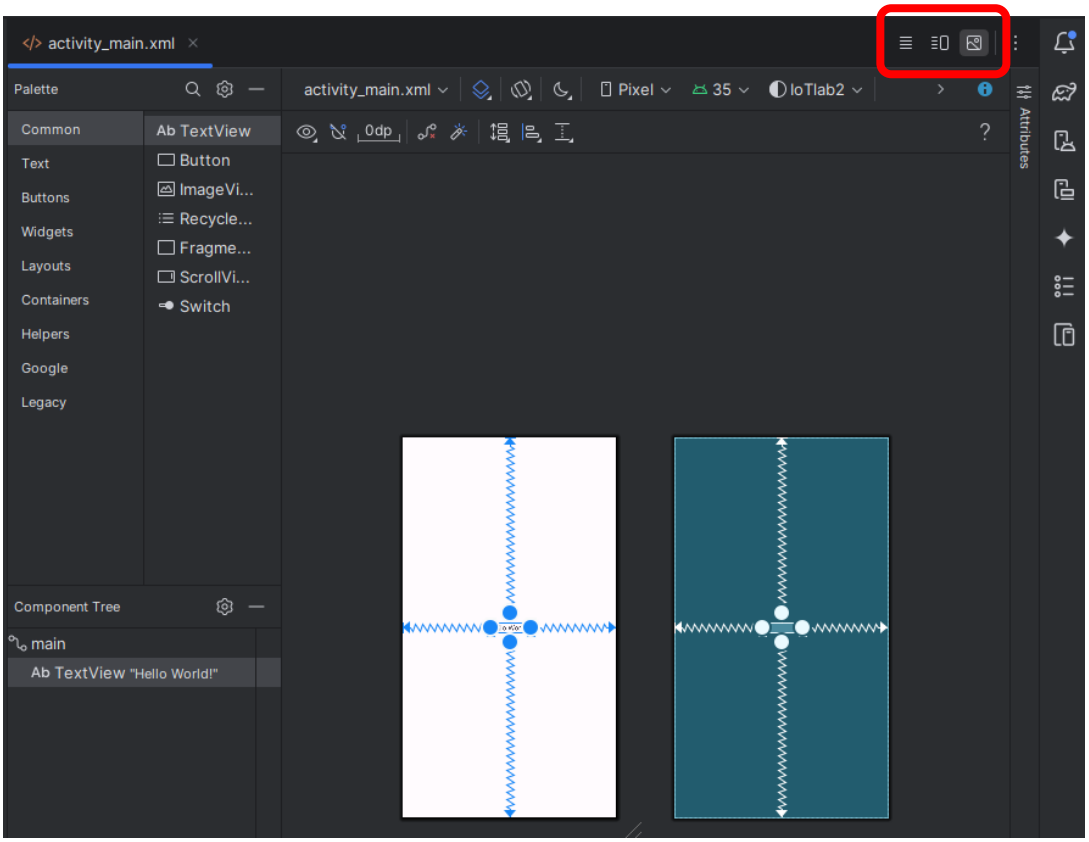
Close the Device manager. You can now run the app on the virtual device you downloaded.

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

As we have sensors and actuators, we need to have at least a Text view field to show temperature values and buttons or switches to affect actuators. Hence, our task would be to create a User Interface (UI) that takes care of the viewing of status on the app. For this, we will edit a layout XML file (in app -> res -> layout) which has at least these settings. Here, either replace the code at the already generated activity_main.xml or create a new layout.

We can create a new layout by right-clicking the new -> Layout resource file. Give it a name, for example, content_main, and choose LinearLayout

To save time, a layout XML file for UI has already been created for this lab which can be downloaded from nextiLearn, the file content_main.xml. Change the content of the layout file, activity_main.xml, or content_main.xml by selecting the text view. There, replace the existing code with the code provided for this lab.



In the layout file (by default activity_main.xml or the filename.xml), replace any existing code with the *content_main.xml* file.

We further will need to change the strings.xml and colors.xml file in app -> res -> values. Add the following in **strings.xml** file within the resources tags:

```
<string name="txv_title_temp">Smart Home</string>
<string name="txv_indoor_temp">Indoor temperature:</string>
<string name="txv_indoor_temp_show">11.11</string>
<string name="txv_outdoor_light">Lighting:</string>
<string name="txv_outdoor_light_show">Off</string>
<string name="txv_outdoor_light_on">On</string>
```

Add the following to the **colors.xml** file add the following lines within the resources tags:

```
<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">#ccf0f0</color>
<color name="activityfont">#3f3f3f</color>
```

Go back to src -> main -> res -> layout -> actitvity_main.xml (or the layout file you created). The view should now show temperature and a slider for lighting.

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. Indoor and outdoor temperature values and actuators' status should be shown immediately after opening the app. Therefore, we must use things' status in MainActivity's **onCreate** method (you can read more about onCreate in [9]). Below an example for one Text view and one button switch is shown. In app -> java -> com.example.<projectname> MainActivity.java add the following code:

Add and prepare text view:

//Above the onCreate method

```
TextView txv_temp_indoor = null;
```

//inside onCreate method

```
txv_temp_indoor = (TextView) findViewById(R.id.indoorTempShow);
txv_temp_indoor.setText("the fetched indoor temp value");
```

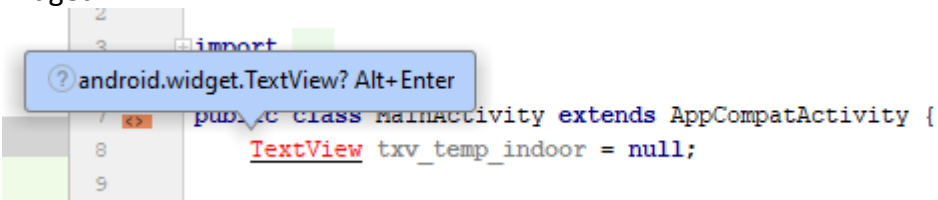
(you have to create the code to fetch the value, note that you might need to change the quotations ("") in android studios for the above code)

Make sure that setContentView is set to the same layout name as the layout. XML file you're using. If you named the layout file content_main.xml, you might want to change this to

```
setContentView(R.layout.content_main);
```

To avoid null pointer exceptions.

For some of the elements, you might need to import the needed widgets. To do this, click or mark code where Android Studios warns you by marking the element as red (as shown below). When the dialog shows, press **alt+enter** to let Android Studios import the right widget.



Add and prepare the button switch:

//Clicking on the button means to affect actuation, so we will need a listener which does //this. We will use the default setOnCheckedChangeListener method for switch

//Above the onCreate method

```
Switch lightToggle = null;
```

//inside onCreate method

```
lightToggle = (Switch) findViewById(R.id.btnToggle);
```

// default setOnCheckedChangeListener method declaration

```
  lightToggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                                         boolean isChecked) {
      // below you write code to change switch status and action to take
                if (isChecked) {

                  //do something if checked
                } else {

                  // to do something if not checked
            }}
        });
```

Add and prepare the button:

//Above onCreate

```
Button btnUpdateTemp = null;
```

//inside onCreate

```
btnUpdateTemp = (Button) findViewById(R.id.btnUpdateTemp);
```

//and add a listener to the button in onCreate

```
btnUpdateTemp.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Add code to execute on click
    }
});
```

3.2.4 SSH from Android/Java

We have used PuTTY during lab 1 to SSH raspberry pi, which obviously we cannot do now. We also do not want to use the credentials whenever we try to access things.

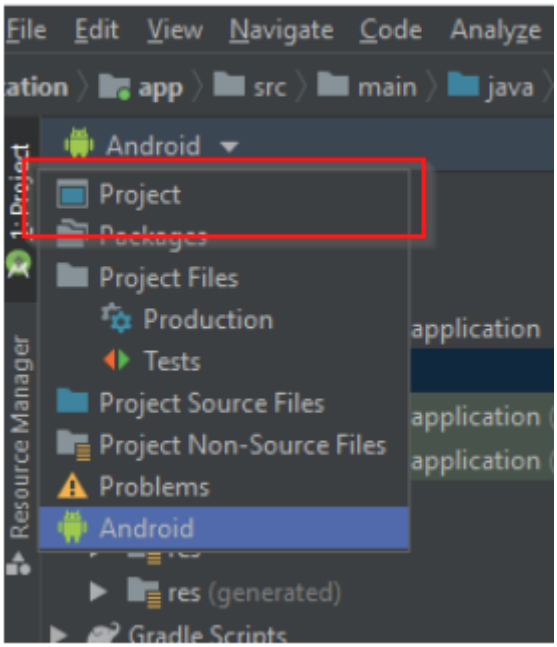
So we will use a library called Ganymed SSH-2 [11] to SSH from inside the app (can be downloaded from iLearn2).

The following code snippet shows how to use SSH using Java. We will use the concept from here to use in the app.

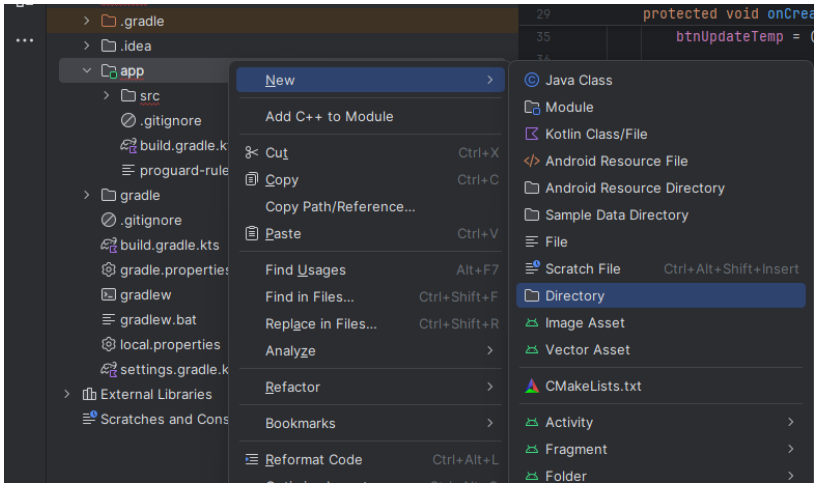
Before pasting the following code, add the Ganymed SSH library to the project!

Download the .jar-file from the above link and save it.

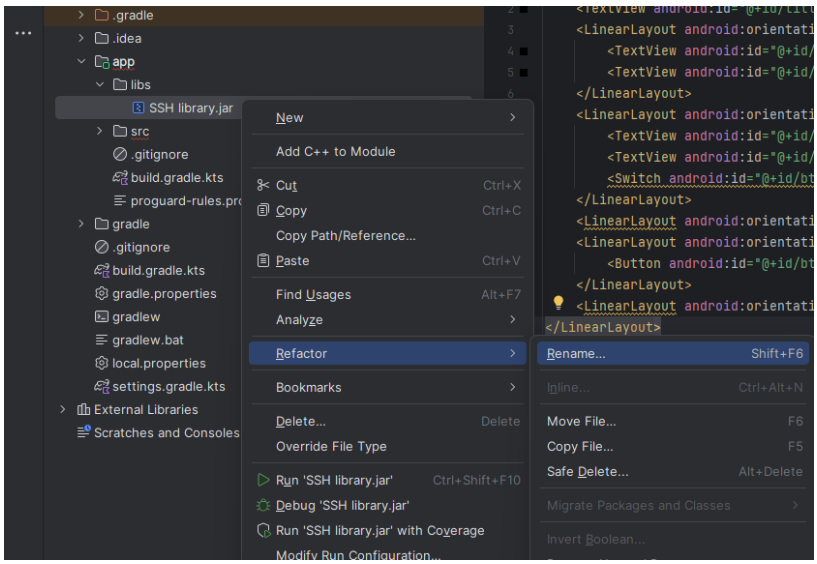
Then, switch to Project view in android studio by clicking on "Android" to open the drop-down menu and selecting Project:



Right click on app -> New -> Directory and give the name libs to the new directory.

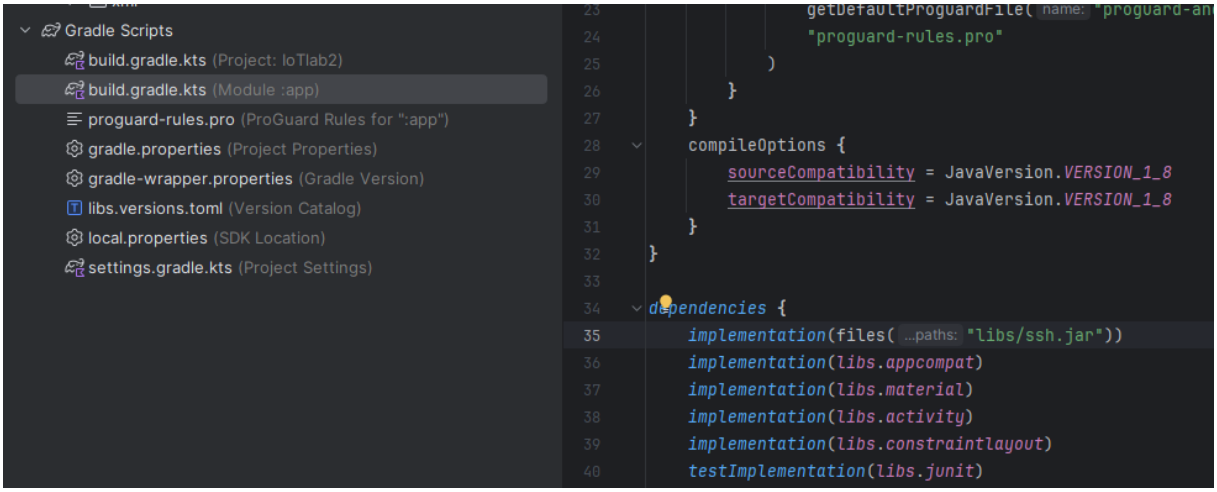


Drag and drop the downloaded jar file in libs directory and rename it to ssh.jar

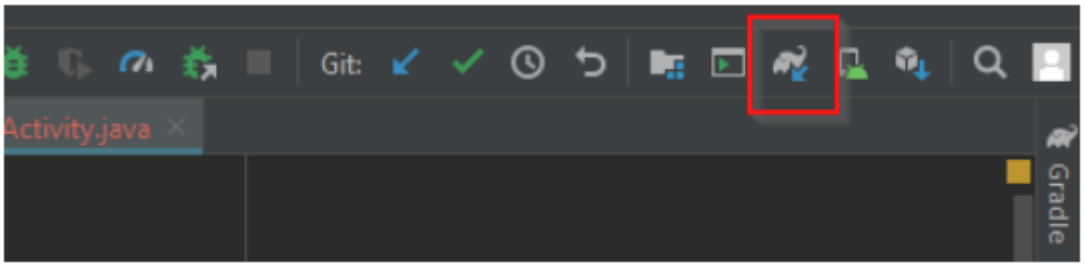


Change to Android again by clicking on the project and then go to build.gradle.kts (Module :app) and add the following line under dependencies.

implementation(files("libs/ssh.jar"))



Now update Gradle to synch with the newly added lib file. Click on the small elephant icon with an arrow in the upper right corner to Sync Project with Gradle Files.

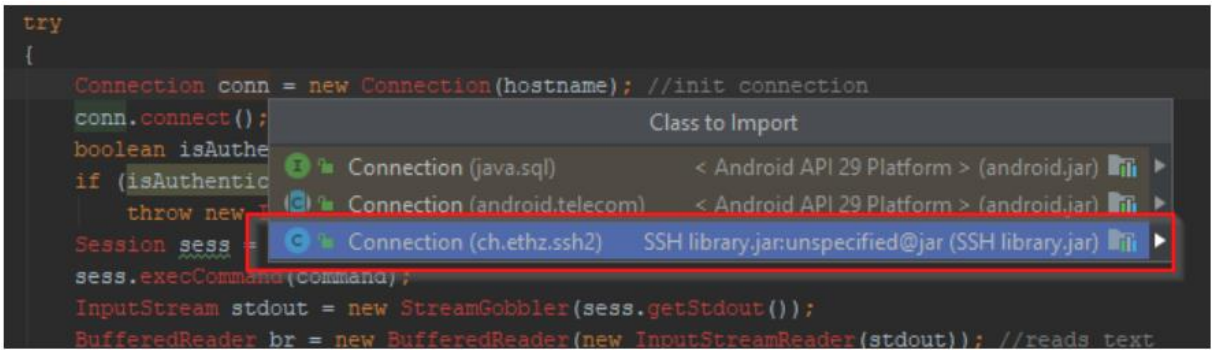


Add the following code to MainActivity.java, outside of onCreate, and change the value of hostname, username, and password according to your raspberry pi.

```
public void run (String command) {
                String hostname = "hostname";
                 String username = "username";
                String password = "password";
                 try
                 {
                 Connection conn = new Connection(hostname); //init connection
                conn.connect(); //start connection to the hostname
                boolean isAuthenticated = conn.authenticateWithPassword(username,password);
                if (isAuthenticated == false)
                                throw new IOException("Authentication failed.");
```

```
                Session sess = conn.openSession();
                sess.execCommand(command);
                InputStream stdout = new StreamGobbler(sess.getStdout());
                BufferedReader br = new BufferedReader(new InputStreamReader(stdout));
//reads text
                 while (true){
                                String line = br.readLine(); // read line
                                 if (line == null)
                break;
                System.out.println(line);
                }
                /* Show exit status, if available (otherwise "null") */
                 System.out.println("ExitCode: " + sess.getExitStatus());
                 sess.close(); // Close this session
                 conn.close();
                }
                 catch (IOException e)
                { e.printStackTrace(System.err);
                System.exit(2); }
}
```

For Connection and Session, be sure to import from the added SSH library:



//The above works very well with Java, but Android might give an error due to Network access //issue. So, add the following before you initiate a connection in the run method.

StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder() .permitAll().build();
StrictMode.setThreadPolicy(policy);

3.2.5 User Permission for Internet Access

The app needs to have permission for Internet access. Otherwise, the app will not be able to access the Internet. Hence, no communication with Raspberry Pi, and things will take place. To do so, add the following line to the AndroidManifest.xml file, in app -> manifests -> AndroidManifest.xml.

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

3.2.6 Android AsyncTask

Now, we want to get results from things by using SSH and show the results on the same UI. Therefore, we need to make use of AsyncTask of Android [more in ref. 10]. This helps us to work in the background and show results on the UI. That is exactly what we need. The code below can either be used within existing methods or as a new method(s).

```
 //add this code where you want to call on SSH to get data, for example in the listener for
//lightToggle and to update temperature-button.
new AsyncTask<Integer, void, void> () {
                @Override
                protected Void doInBackground(Integer... params) {
                //your code to fetch results via SSH
                return null;
                }
        }.execute(1);


//Alternatively, you can use the code template below if you want to execute something after
the code in //doInBackground has been completed, useful for changing UI values:
new AsyncTask<Integer, void, void> (){
                @Override
                protected Void doInBackground(Integer... params) {
                // Add code to fetch data via SSH return null;
                }
                @Override
                protected void onPostExecute(Void v) {
                // Add code to preform actions after doInBackground
                }
 }.execute(1);
```

Need more help with how to use AsynchTask? Check extended instructions: using async task for instructions.

3.3 Modify the code and make use of strings

Through the previous steps, you got the needed "code skeleton" to finish the lab. The goal is to:

- Fetch the temperature from a sensor, and visualize this in the app
- Through the app, be able to turn the actuator on and off through the toggle switch

When you're done, show the workshop leader.

The following steps are hints for the lab to make use of results from the gateway.

- Save the lines from SSH as StringBuilder
- Split each new line and save it as a string array
    - E.g., string. split("\\n"); // to split
- Read each line and make use of results from each line
- You can change run() to return a value
- Feel free to add any additional variables
- Change the text value of a text view through the element.setText("your text here");

4 Lab report

After 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 an answer to each question):

1. Write briefly about the steps you had undertaken to access things' status via the android app (2)
2. What status (things) did you get on your app? (attach a screenshot) (1)
3. Write down a minimum of 3 (three) ideas for expanding the app in this lab. Feel free to include additional sensors not used in the lab or expand through other functionality in the app using the already provided data (2)

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.
No submission will be entertained unless you perform the actual lab during scheduled lab time.

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/runningapp.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] Telldus Live, https://telldus.com/se/telldus-live/

[7] Telldus API, https://api.telldus.com/

[8] Python documentation, dictionary:
https://docs.python.org/3.5/tutorial/datastructures.html#dictionaries

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

[10] Android AsyncTask, https://developer.android.com/reference/android/os/AsyncTask.html

[11] Ganymed SSH-2, http://www.ganymed.ethz.ch/ssh2/

6 Extended instructions

Extended instructions: get a specific value

Remove the line that prints the whole JSON response or comment it out (line 29).

To only print out the value for temperature, do the following:
Remove or comment the last line with print, and add the following line to the end of the file:

```
print(responseData['sensor'][0]['temp'])
```

This will print only the value from the very first sensor (index 0 in the array of sensors). Change the 0 to receive values from another sensor.

If you were only to print the humidity value from the sensor, change 'temp' to 'humidity'.

Extended instructions: using Asynch task

For this lab, the only value we want to show is the temperature value in the text view txv_temp_indoor.
This means that we do not need to use onPostExecute for simply turning the actuator on and off.

In the listener for lightToggle, add one async task in isChecked and another in the else section.
In the doInBackground for tasks, call the run() method and use the command you want the raspberry pi to perform as a parameter. For example, "ls" would return a list of all the files in the current directory (home) of the pi. "python filename.py" would run a python script.

In the listener for btnUpdateTemp, add an async task with onPostExecute.
In the doInBackground, add a run() and the command you want to perform to access the temperature value.

We cannot change the values of the UI during doInBackground async task; instead, this can be done through onPostExecute.
In the onPostExecute, set the value of txv_temp_indoor by using

```
txv_temp_indoor.setText("the new value here");
```

Extended instructions: save and use values from an SSH session

During the session, the BufferedReader will read each line from the Raspberry Pi. This is why we wanted to modify the output from the sensors in 3.1.4.

To read and save data from the BufferedReader, use a StringBuilder to which you append each line.
In the run method, add

```
StringBuilder output = new StringBuilder();
```

To add each line from the buffered reader to the string builder, add `output.append(line);` to the while-loop.

Now that we created a string builder with all the data gathered from the GET-call done through SSH, we need to save it to use it.
Create a String variable outside of onCreate, and give it a name. For example, *outputValue*.
You can initialize it as an empty String:

```
String outputValue = "";
```

To save the value from the StringBuilder, in the run() method, outside of the loop, assign the output value to outputValue. You also need to convert it to a string by using .toString:

```
outputValue = output.toString();
```

If your list sensor-script prints more than just the temperature value, you can easily modify the string using the inbuild .replace() or .split() for Strings in Java.

Now you can use the saved output value to set the value of *txv_temp_indoor* in the OnClick listener for btnUpdateTemp.