Documentation

Project Software Engineering
2019/2020

SAXION

Index

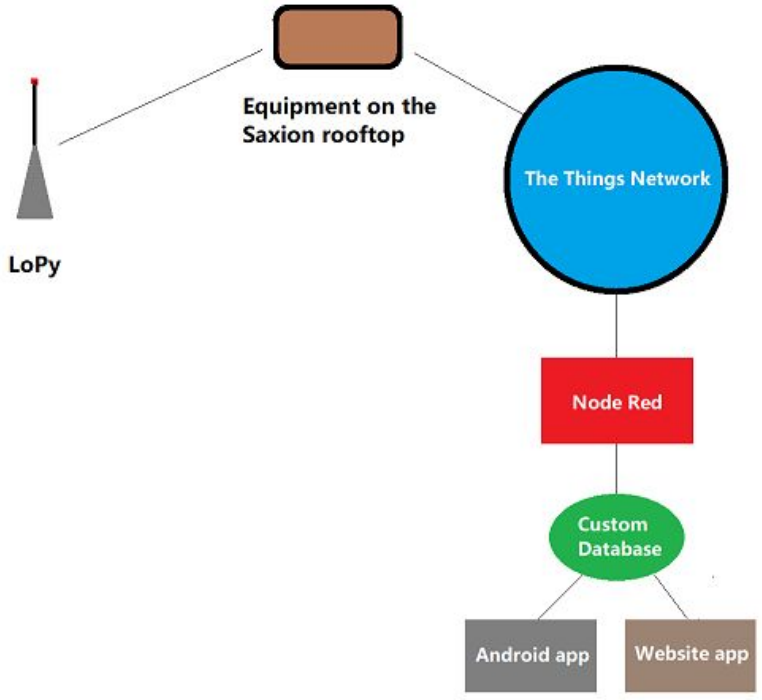
Introduction

In the Project Software Engineering, the challenge undertaken by the Team was to create a system for facilitating weather data collection by the Client. The system involves a LoPy, a device for sensing the weather, which has a connection to an antenna on the roof of the Saxion building. This data is subsequently sent to the things network and, through a broker, it is sent to the custom made database hosted on a website (the functioning of which is described below).

The Client has the availability of viewing this data in either an Android interface or a web app.

Architectural overview

Below is a visual representation of how the said system functions:



Coding style

The coding style explains how weathercorp writes the code for the application with strict rules to follow, which is heavily dependent on Google's coding standards for source code in Java and Python Programming Language.

Methods, parameters and variables naming

- Strictly followed the lowerCamelCase which always start with a lowercase letter
- Not contain any numbers
- No random, made up names (abbreviations are allowed) but meaningful names which describes its functionality if possible

Whitespaces

- Follow standard typographic rules for the use of spaces around punctuation.
- No whitespace inside parentheses, brackets or braces.
- No whitespace before the open paren/bracket that starts an argument list, indexing or slicing.
- No trailing whitespaces.

Global variables

- Are avoided since they have the potential to change module behavior during the import, because assignments to global variables are done when the module is first imported.

Comments

- When code is finalised there will not be commented-out code.
- Comments as headers to explain what the program does and who made it when.
- Every function/method should start with a comment block explaining what the function/method does.

Braces

- Are used with if, else, for, do, and while statements, even when the body is empty or contains only a single statement.
- Start on the same line as the definition but will end on a seperate line

Code of honor

- Code written by different users should be mentioned with a start and end comment with a link to the original code.

LoPy

The LoPy is a quadruple bearer MicroPython enabled development board (LoRa, Sigfox, WiFi, Bluetooth) perfect enterprise grade IoT platform for your connected Things. With the latest Espressif chipset the LoPy offers a perfect combination of power, friendliness and flexibility.

Lopy/Pysense sensors

The LoPy is connected with the Pysense, Pysense is an expansion board to include more functions and sensors to your LoPy. The sensors include: ambient light sensor; barometric pressure sensor; humidity sensor; temperature sensor and a 3 axis 12-bit accelerometer. For our implementation the 3 axis 12-bit accelerometer was not used.

Ambient light sensor

The ambient light sensor on the Pysense is the LTR-329ALS-01. The sensor gives 2 data points the red lux and blue lux. The data that is used by the application is the blue lux because it is a wider spectrum and more accurately adjust to light levels.
Lux is a SI unit of illuminance, 1 lux is equal to 1 lumen per square meter. This unit is used to measure the intensity of light as perceived by the human eye.

Barometric pressure sensor

The barometric pressure sensor on the Pysense is the MPL3115A2. The sensor has 2 different modes pressure and altitude. For the application the pressure mode is used since the sensor is being used for a weather station.
The sensor gives the pressure in pascal the LoPy then converts it into mBar which is more widely used.

Humidity sensor

The humidity sensor on the PySense is the Si7006-A20. The sensor measures the concentration of water vapour present in air.
The sensor gives the humidity in percentage which is then displayed on the application.

Temperature sensor

The temperature sensor is the same as the humidity sensory since the temperature is needed to calculate the humidity. The temperature is given in Celsius, the value is not accurate as the sensor is next to the LoPy which heats up the sensor. This problem scues the value with almost 10 degrees.

LoPy code

The LoPy is coded in MicroPython, an implementation of Python 3 which is optimized to run on microcontrollers and constrained environments. MicroPython has all the same features as Python 3 with the only difference being that it works with 256k of code space and 16k of RAM.

LoPy flowchart

For more clarity in code a flowchart has been made which can be found under attachments.

The Things Network

The Things Network offers open tools and a network to make IoT applications on. TTN is a cloud service which can store your uplink messages for a set period of time and make integrations on. Integrations can range from data storage to HTTP integration.

Data decoder

The TTN used by Weathercorp only makes use of one function in TTN and that is the decoder. The decoder transforms the payload from bytes to ASCII.

```
function Decoder(bytes) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};
  var counter = bytes[4];
  decoded.temp = bytes[0];
  decoded.hum = bytes[1];
  decoded.pres = bytes[2]*1000;
  decoded.light = bytes[3]+(counter*255);

  return decoded;
}
```

As shown in the picture the Decoder function has a parameter “bytes”, this is the incoming payload given in bytes. An array is made with the decoded fields of text, temperature and humidity are decoded simply. Pressure was altered to fit in a byte and then decoded to be the original value here. Light has an extra byte which stores a counter that counts every time light went over 255. This way light value can be over 255.

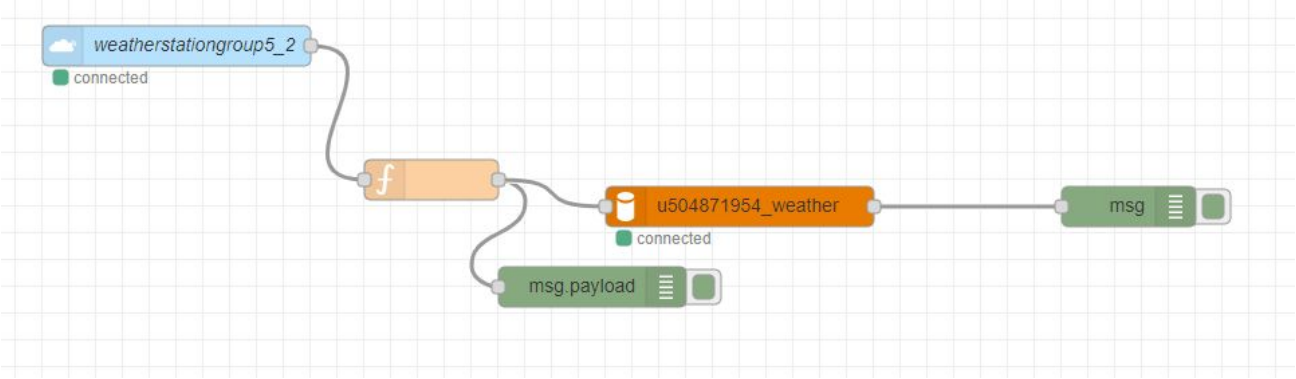
When everything is decoded the values are displayed on the TTN website.

Node-Red

Node-Red is a programming tool for wiring hardware devices with APIs and online services. It has a browser-based editor that makes it easy to make a comprehensive overview of what is done with ease debugging build in. The service is deployed with 1 click for easy testing and hosting is cheap.

Node-Red with TTN

Node-Red has many libraries(collections) that are built especially for use in TTN and databases, this makes it easy to connect to both TTN and the databse.



The blue node connects to TTN to get the data that already has been decoded to ASCII, then the function node puts the data in a MySQL query statement to be inserted in the database. The database node is connected to the database server and can execute the MySQL statements. The green nodes are debugging nodes and are used to read the data payload to see the data before it is sent to the database. The second debug node is used to read the message from the database in case it has an error.

Android mobile app

The Android app would be built from scratch using the Android Studio development environment and simulate in GenyMotion as well as Android mobile phone. The Android phone use API level 26 and Android version 8.1.0 (Oreo).

Android Studio and GenyMotion

The using Android Studio is built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. Android allows different layouts to be created for different screen sizes which would be determined based on the screen size of the current device.
However, for the simulation, GenyMotion is to be used on pair with Android Studio Simulator. It is an Android virtual devices that can control the environment and manage the infrastructure by creating virtual devices without GPU therefore simulating time is minimized, which is more efficient than using the built-in simulator in Android Studio.
At the finalize state, actual Android device is used in order to see the most realistic deployment of the application.

Coding style

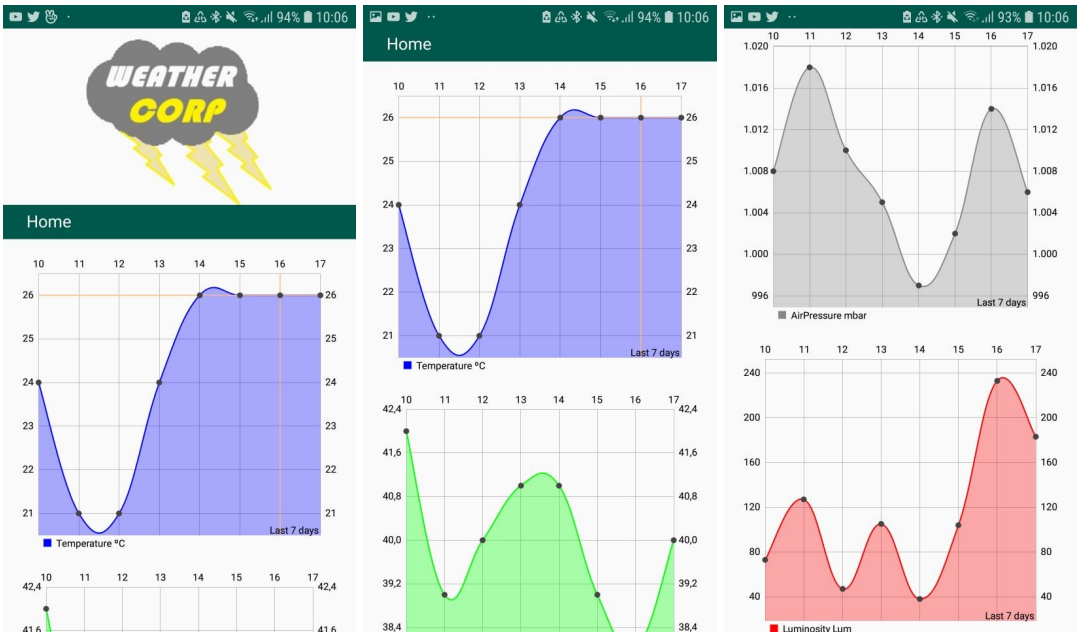
Since the Android Studio has its default IDE being IntelliJ, the coding style is conducted followed the mentioned coding style rules above.

App front-end

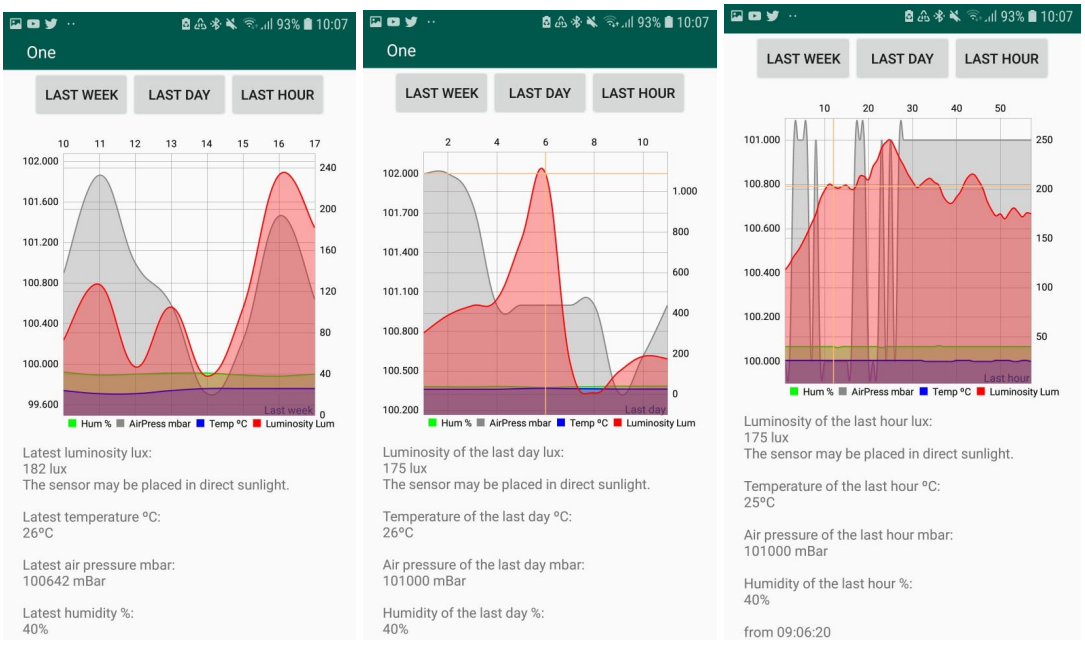
for user

icon of the application

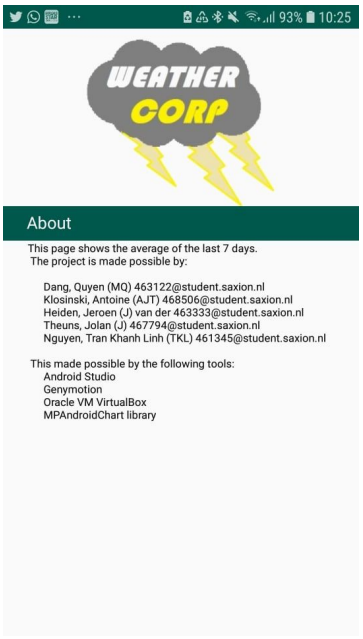
Although being built from scratch, the mobile app is still heavily influenced by the website’s design. The header navigation bar contains three options: Home, One and About with Home being the default screen. Each option would be directed onto its own screen.



In the Home screen, four separate graphs displaying the daily average measure value for the last seven days of Temperature, Humidity, Air Pressure and Luminosity, respectively.



In the One screen, all four parameters are displayed in one graph. The user can choose to see data of the last week, last day or last hour. The data showed in "*last week"* option is the same with data showed in Home screen. In *"last day"* option, hourly average measure value are shown and in *"last hour"* option, all data measured in the last hour are shown. Data for Air Pressure use the left hand side axis and other parameter use the right hand side for value reference. The last displayed value of all four parameters are clarified below the graph with it's time stamp.



The About screen contains information about group members and tools that are used for the making the app.

All the graphs that are featured in the app are animated and drawn with different color for different parameter. User can scale all the graphs up and down for better view or can tap in any of the data point, which will show a yellow curse that pointed to both vertical and horizontal axis for more precise data value.

After implemented in an actual device, the app can run remotely without connected to the computer and can show the most recent data with only the need of wifi (reminder: without wifi, the app will not run).

App back-end

for developer

The application uses MPAndroidChart library to display data. MPAndroidChart library is currently the most used free-resource to shows data in graph view and can be easily implement by adding github repository in gradle folder. The library is an effective to display a beautiful graph and it attracts a broad community of developer, who constantly discussed about now feature that supported by the library, which make the process of implement different function into the application easier and more efficient. However, some of the features will perform poorly when the app is installed in a low Android version device. The application is recommended to be used in device with at least API level 21 (Lollipop). The ability of displayed multiple yAsix is limited and the list of data that wanted to add to the graph also need to have x value sorted in ascending order.

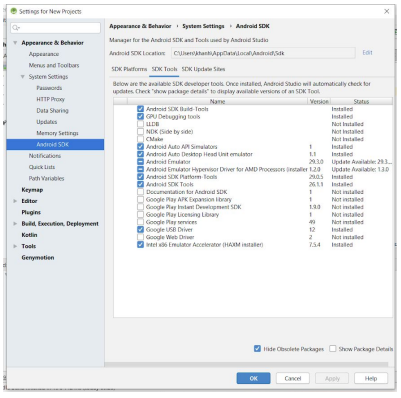
The data is received in json format from an URL which contains php file with sql request to the online database. All the php files are currently stored in Hostinger at the same directory with code for the website.

The app do not have a local database as well as a copy of the last requested data in order to reduce the size of the app and save memory for the device. Therefore, the app cannot run without wifi connection. The app can't reach any of the URL links and will return error which force it to stop. However, the user can fix this problem by simply reconnect to wifi and then rerun the app, the most recent data will be displayed.

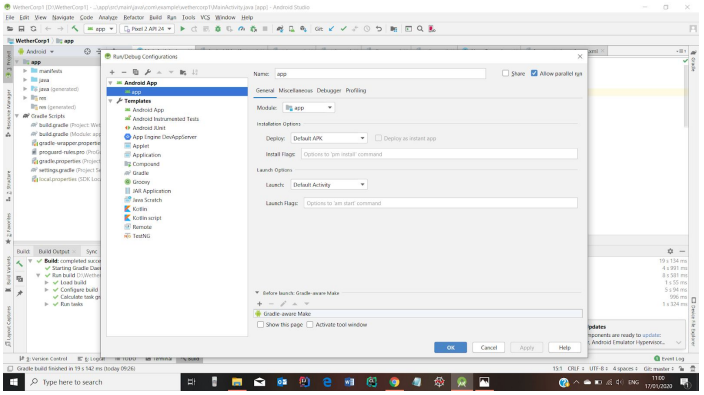
The app and the website use the same request connection to database. All the sql queries used for the app are the same as queries for the website but in separate files because the result from this requests need to be exported in json format in order for Android to get the data and parse into desired format for graph and text displayed later.

How to self-test and further modification for the app in the future.

1. Clone the app from github or directly download to computer.
2. Download Android Studio, JDK (if the developer do not have it already installed). [https://developer.android.com/studio/install]
3. Open the downloaded Android app file in Android Studio. If there is no option to choose for the run button, try reconfigure the app.
    3.1 Go to Tools -> SDK manager -> Android SDK -> SDK Platforms -> Make sure Android 5.1, 5.0 are checked.
    3.2 Make sure Android SDK Location is pointing to the right location.
    Ex. C:\Users\khanh\AppData\Local\Android\Sdk
    3.2 Check SDK Tools tap, make sure to install all the needed tools.



    3.3 Go to SDK Update Sites, make sure everything is up to date.
    3.4 Set the Run/Debug configurations with Android App option from Templates

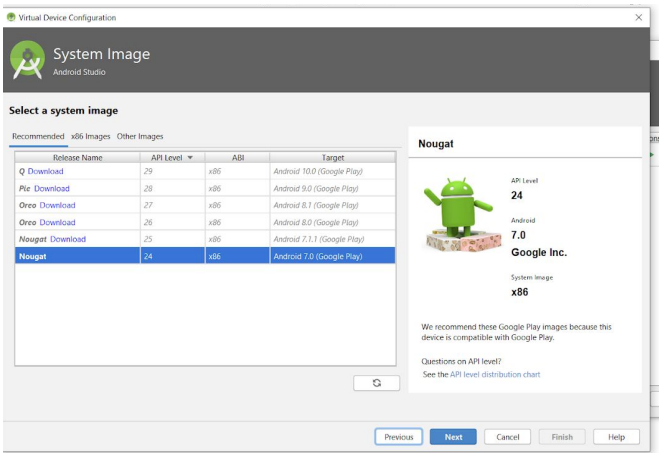


[https://developer.android.com/studio/intro/studio-config]

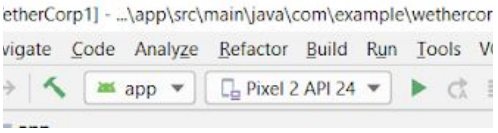
5. Testing device.

5.1 Incase the developer don't have an actual Android phone, download GenyMotion for testing. [https://www.genymotion.com/plugins/]

5.2 In order to use AVD build in device, go to Tools -> AVD manager -> Create Virtual Device (In case no option are shown) -> Choose any device with API level from 21 and up -> Download any required file that are asked from the below screen -> Click next.



If the configuration is done, the below option should be seen from Android Studio action bar. (The available device option could be different devices, depended on what the developer use between Genymotion, AVD build in virtual device or actual Android phone)



5.3 If the developer use actual Android device, first set up developer mode in the Android phone.

[https://developer.android.com/studio/debug/dev-options]

Follow instructions in the link, focus on *"Enable developer options and debugging"* and *"Debugging -> enable USB"* section for the simple way.

6. If the project is done building and is running on the Android device, the app will pop-up in the device screen or if there is no auto pop-up screen, navigate to the main apps display screen in the device and look for application with icon of **"WeatherCorp".**

Database layer

Database

The database uses MySQL. With PHP and Java Weathercorp access the database.

Name	Type	Default	Extra
id	int(11)		AUTO_INCREMENT
humidity	int(11)		
pressure	int(11)		
temperature	int(11)		
light	int(11)		
timedate	datetime	CURRENT_TIMESTAMP	

Every time the NODE-RED obtains the data from The Things Network it store it in to the database. The exact database looks like the table displayed above. When a new record gets created it will automatically give it a new ID (**AUTO_INCREMENT**). The **timedate** use the current system date (the date of the database itself) (**CURRENT_TIMESTAMP**).

Important query

There three major queries we use to obtain the data

This query is used the data from the last week:

1.	**SELECT ROUND(AVG(humidity)), DATE(timedate) FROM weather WHERE DATE(timedate) >= (DATE(NOW())- INTERVAL 7 DAY) GROUP BY ROUND(DATE(timedate))**
2.	**SELECT ROUND(AVG(light)), HOUR(timedate) FROM weather WHERE timedate >= DATE_SUB(NOW(), INTERVAL 1 DAY) GROUP BY DATE_FORMAT(timedate, \"%H %d\") ORDER BY timedate;**
3.	**SELECT * FROM weather WHERE DATE(timedate) >= (DATE(NOW())- INTERVAL 1 HOUR) GROUP BY ROUND(DATE(timedate));**

This queries shown above is the result for temperature, this could be changed to: **temperature**, **humidity**, **pressure** and **light**.

The first query is used to obtain the data from a 7 days back from the current day. It calculates the average of all the data points obtained from the day.

The second query is used the get obtain the data from a 24 hour back form the current it calculates the average of all the data points obtained from the hour.

The second query is used the get obtain the data from a 1 hour back form the current it. Every minute the database gets updated, so it's about 60 data points.

The other query that is used to show the most updated information is:

```
SELECT * FROM weather ORDER BY id DESC
LIMIT 1;
```

This query makes sure that the latest stored record will be returned.

The air pressure stored in the database is in pascal. But the GUI shows it in millibar. The pascal is divided by 100. So, it will show the result in millibar.

JavaFX

General explanation

During Project Software Engineering, another potential task was to design a JavaFX Graphical User Interface (GUI), with the purpose of displaying the weather obtained from a database hosted by my Team on a website. This was to be one of the three User Interfaces which the Team would present within their final product, alongside the website interface and the Android app. The idea was first introduced during Sprint 1, and the Team has committed myself to it by the start of Sprint 2.

Introduction

In principle, the GUI in question was to be a helpful solution for users trying to look at the weather either at a given moment in time, or within the last 7 days. The idea was to have several scenes in one window, which could easily be accessed one from another simply by pressing a button. This would include scenes which the Client could go to in order to see the different available weather quantities (Temperature, Pressure, Humidity, and Luminosity), depending on the button he clicked.

Structure

General structure

For the purpose of implementing the said functionalities, and integrating them well together, the program consisted of multiple classes and one interface, as well as a *.properties* file. The functionality of the most important of those files will be explained in the section below, with screenshots provided for facilitation.

Classes

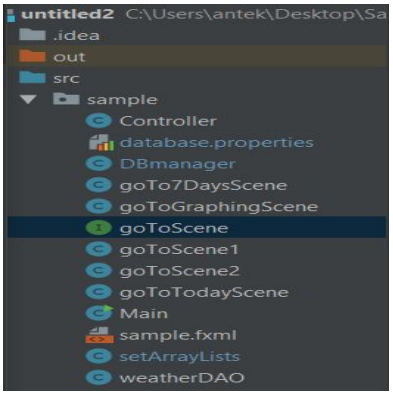
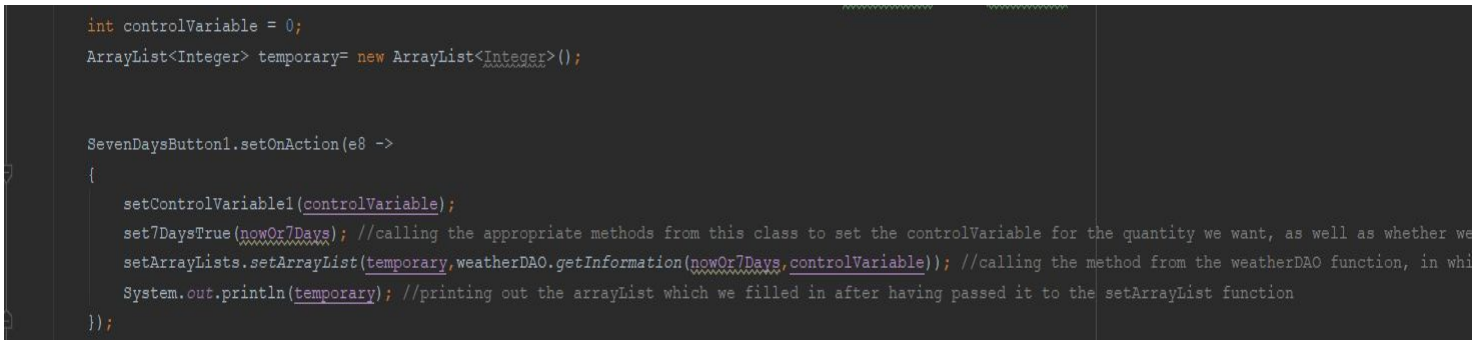


Figure 1 - The structuring of the GUI source files

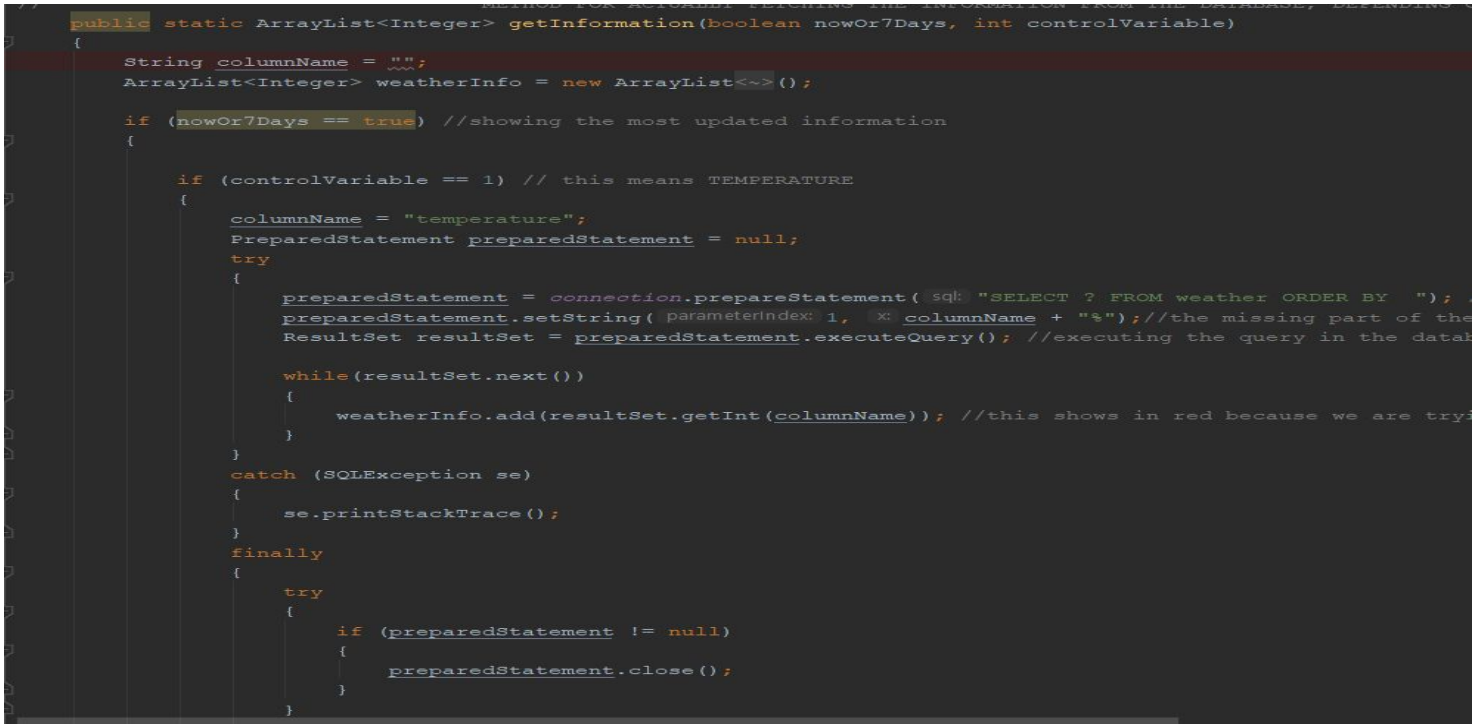
An important aspect of the program's functionality was the button structure in the Main class; as seen in Figure 3 below, the previewed optimal solution when tackling this challenge was to use lambda expressions, as it meant that the Main did not have to extend the Application class (hence making the code more understandable to people with no JavaFX background), and overall making the program more logically structured as a whole.



```
int controlVariable = 0;
ArrayList<Integer> temporary= new ArrayList<Integer>();


SevenDaysButton1.setOnAction(e8 ->
{
    setControlVariable1(controlVariable);
    set7DaysTrue(nowOr7Days); //calling the appropriate methods from this class to set the controlVariable for the quantity we want, as well as whether we
    setArrayLists.setArrayList(temporary,weatherDAO.getInformation(nowOr7Days,controlVariable)); //calling the method from the weatherDAO function, in whi
    System.out.println(temporary); //printing out the arrayList which we filled in after having passed it to the setArrayList function
});
```

Figure 2 - The implementation of a button used for fetching data from the database

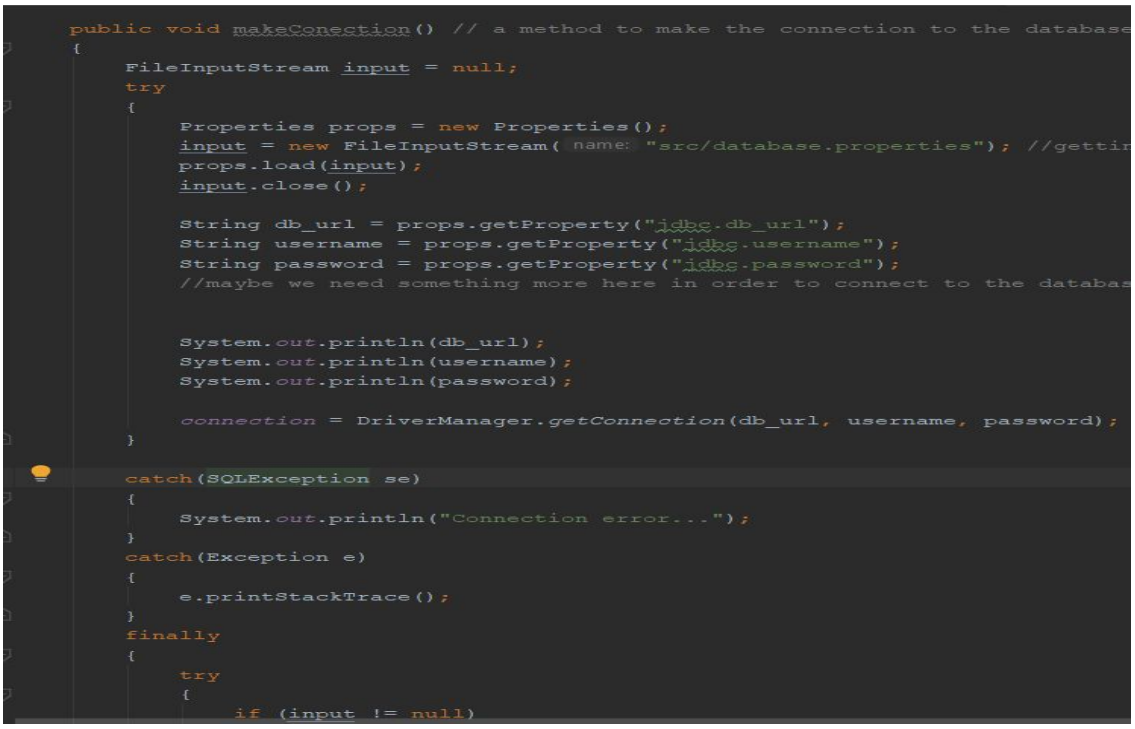
In addition to this, the reason the program failed to entirely fulfill its desired functionality was for a similar reason; as may be observed in Figure 4, the original plan encompassed the idea of there being several buttons which the Client could click to obtain specific data from the database. This, nevertheless, proved to be a big implementational obstacle for the Team, as the method was just returning an empty ArrayList. Moreover, a class used when performing the later-discontinued Java Database Connectivity was the weatherDAO class; it can be seen in Figures 3 and 4, and is explained below in more detail.



```
public static ArrayList<Integer> getInformation(boolean nowOr7Days, int controlVariable)
{
    String columnName = "";
    ArrayList<Integer> weatherInfo = new ArrayList<~>();

    if (nowOr7Days == true) //showing the most updated information
    {

        if (controlVariable == 1) // this means TEMPERATURE
        {
            columnName = "temperature";
            PreparedStatement preparedStatement = null;
            try
            {
                preparedStatement = connection.prepareStatement( sql: "SELECT ? FROM weather ORDER BY  ");
                preparedStatement.setString( parameterIndex: 1,  x: columnName + "%");//the missing part of the
                ResultSet resultSet = preparedStatement.executeQuery(); //executing the query in the datab

                while(resultSet.next())
                {
                    weatherInfo.add(resultSet.getInt(columnName)); //this shows in red because we are tryi
                }
            }
            catch (SQLException se)
            {
                se.printStackTrace();
            }
            finally
            {
                try
                {
                    if (preparedStatement != null)
                    {
                        preparedStatement.close();
                    }
                }
```

Figure 3 - getInformation method, for fetching specific information from the database

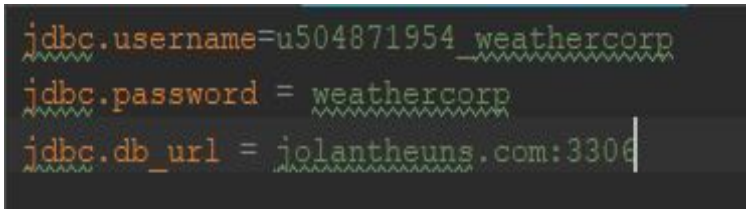
Another class which was of rather high significance, was the DBmanager class. This class contained methods responsible for checking whether the given database exists and, in the case it doesn't, either printing out an appropriate message or throwing an exception. It also contained methods responsible for making the actual connection to the database.



```
public void makeConection() // a method to make the connection to the database
{
    FileInputStream input = null;
    try
    {
        Properties props = new Properties();
        input = new FileInputStream( name: "src/database.properties"); //gettin
        props.load(input);
        input.close();

        String db_url = props.getProperty("jdbc.db_url");
        String username = props.getProperty("jdbc.username");
        String password = props.getProperty("jdbc.password");
        //maybe we need something more here in order to connect to the databas


        System.out.println(db_url);
        System.out.println(username);
        System.out.println(password);

        connection = DriverManager.getConnection(db_url, username, password);
    }

    catch(SQLException se)
    {
        System.out.println("Connection error...");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            if (input != null)
```

Figure 4 - makeConnection method in the DBmanager class

Finally, the last aspect of the program which was needed to make it run, was the *.properties* file; as the extension suggests, it contains the properties of the database in question. The use of this file in the program is necessary in order to establish a connection.



```
jdbc.username=u504871954_weathercorp
jdbc.password = weathercorp
jdbc.db_url = jolantheuns.com:3306
```

Figure 5 - Properties of the database, as specified in the database.properties file

UML diagrams

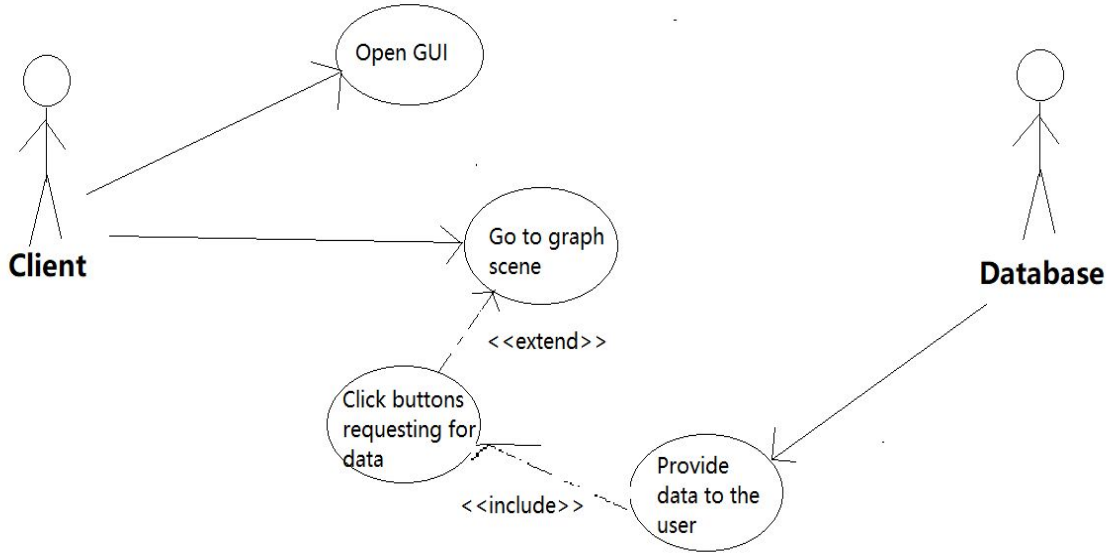


Figure 6 - The Use Case Diagram, methodologically describing the functioning of the GUI

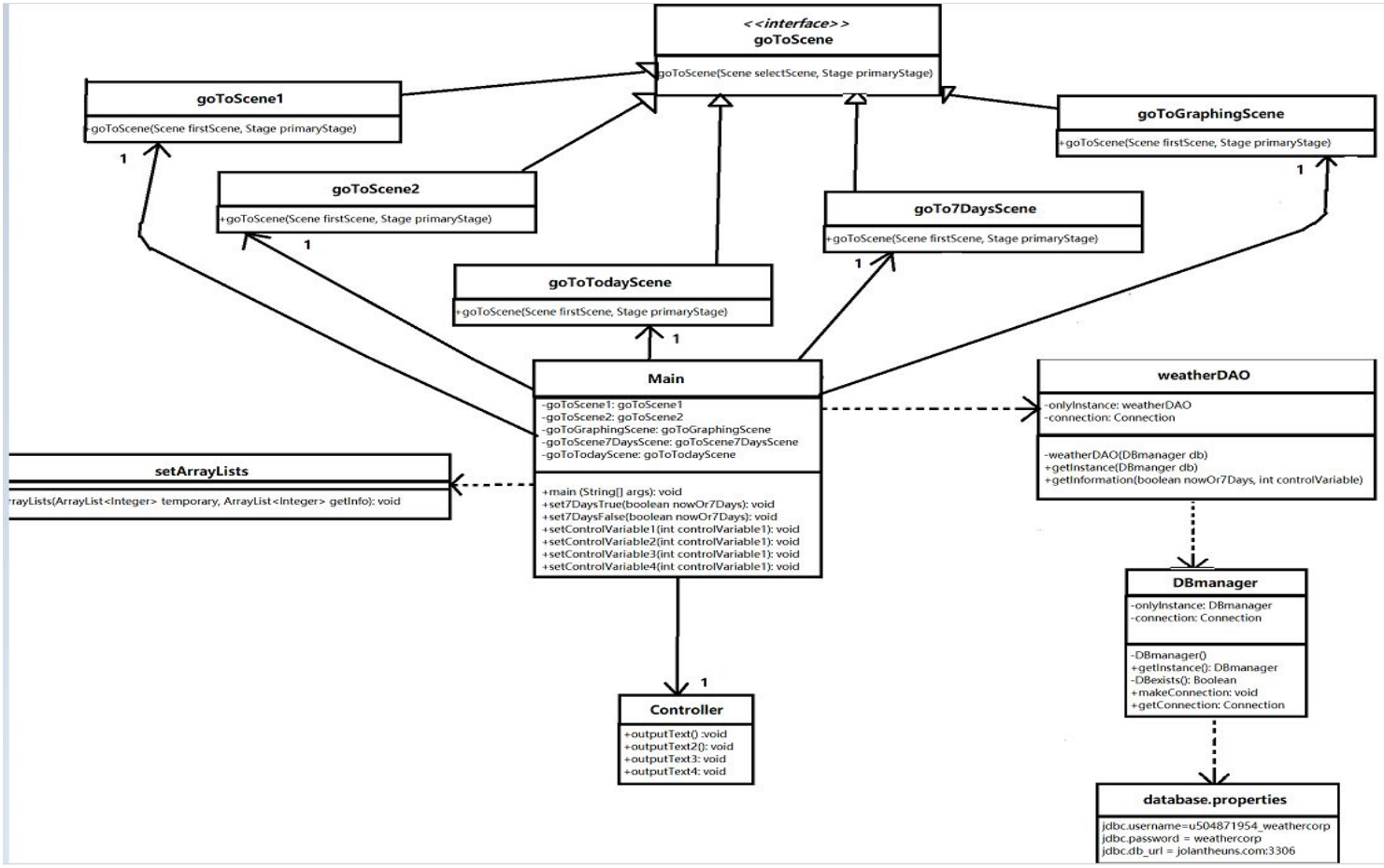


Figure 7 - The Class Diagram, showing all the classes in the program and their relationships

Visual representation

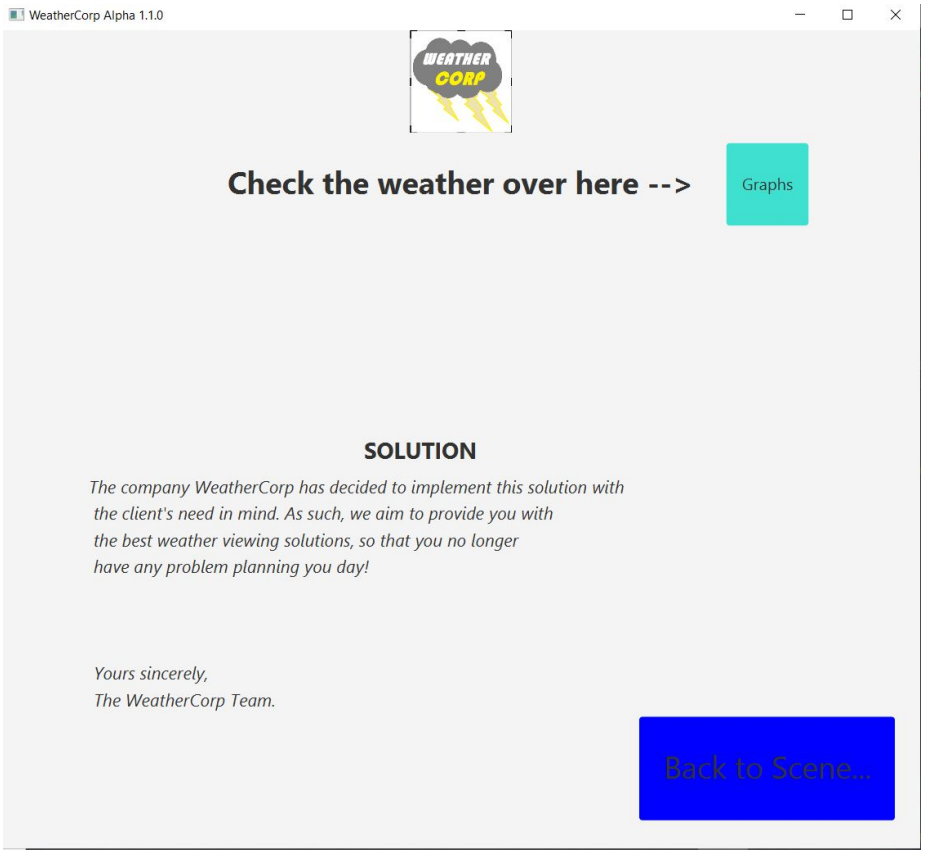


Figure 8 - The visual representation of the demo version of the GUI

Set up product

This is a comprehensive guide to set up the finished product and get it all running smoothly on every computer. This guide will include setting up TTN, setting up Node-Red and connecting to the database. All files that are needed can be found on github: “https://github.com/JolanTheuns/weathercorp”.

Setting up LoPy

To upload the code from the github directory “LoPy” the user should install the pymakr library for Visual studio code or Atom. In these programs the code can be directly uploaded to the board without any hassle.
Having any problems please follow: https://docs.pycom.io/pymakr/installation/.

Setting up TNN

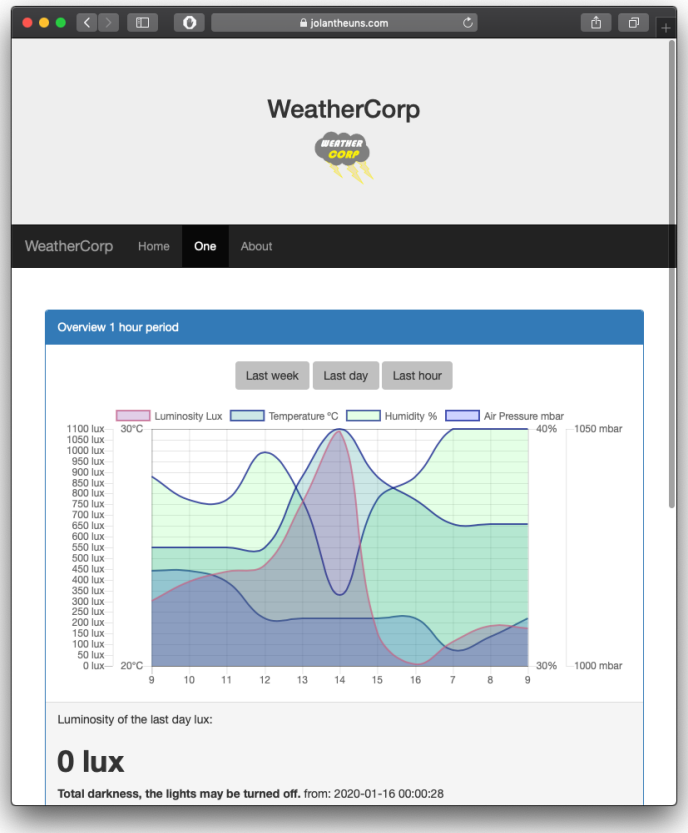
After setting up the LoPy and making sure the activation method is on ABP, the decoder should be added to “payload formats”.

```
function Decoder(bytes) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};
  var counter = bytes[4];
  decoded.temp = bytes[0];
  decoded.hum = bytes[1];
  decoded.pres = bytes[2]*1000;
  decoded.light = bytes[3]+(counter*255);

  return decoded;
}
```

Now the TTN is set up and ready to receive data. If there are any problems with connecting the board to TTN please follow the tutorials on the TTN to connect them together. If all has been done correctly you can now see data coming in and shown decoded on you application page.

TTN extra tutorial: https://docs.pycom.io/gettingstarted/registration/lora/ttn/.

Visual representation of the website.



Setting up the database

For the website you need to make sure there is a server which supports PHP and MySQL. Prior to this follow this link on how to configure it: https://netbeans.org/kb/docs/php/configure-php-environment-windows.html

On GitHub there is a folder named: “database” which contains an empty .sql file which contains the tables to initialize the database. The database which is supported is MySQL, after importing the sql file it is possible to run the Node-Red.

Setting up the website

On GitHub there is a folder: “website” which contains the whole website. You can upload the whole website to a web server and run the website, also make sure you initialize the database server first. There is a file called: “dbconnect.php” there are two varibles you need to change:

- $username = "";

- $password = "";
- $servername = "";

You need to fill in the username and password and address of the own database server.

Setting up Node-Red

Node-red can be hosted on a server, a Raspberry Pi or equivalent or locally on any computer. This setup will explain how to install it on a server.

IBM Cloud is a free Node-red hosting platform which makes it easier to deploy and an easy way to have it running 24/7. After creating an account you can start a Node-Red instance where in the drop down menu you can import the “flows.json” file from Github. This file ads all the flows automatically, the only thing that needs to be changed are the id’s of the TTN and database. This is done by double clicking on the node and pressing the change button, there you can change id’s and passwords.

After the details have been changed the Node-Red can be deployed on the IBM Cloud.

Link to IBM Cloud: “https://www.ibm.com/cloud”.

Attachments: