Final Technical Report

Cloudie

Place, Date: Enschede, 1st December2023

Drawn up by: Alexandru Ilies

Lorenzo Davoli

Cătălin Chiprian

Bart de Ruiter

Ali Mobini

Margarita Konnari

Matvey Seregin

Version: 1.1

Table Of Contents:

1. Contact Table 3
2. Introduction and Architectural 3
3. Data used 3
4. Communication Between the Components 6
5. Explanation of Key Concepts 7
6. Explanation of the GUI(s) 14
5.1 Weather Statistics Page: 15
5.2 Weather Graphs Page: 15
5.3 Device Information Page: 16
7. How Everything is Tested 16
8. Short Manual 18
8.1 Arduino setup manual 18
8.2 MQTT Data Reciever Manual 20
8.3 Cloudie Windows Application manual 20
9. Recommendations 21
10. List of of references 22

1. Contact Table

Name	Phone Number	Email
Alexandru Ilies	+40773775927	533540@student.saxion.nl
Lorenzo Davoli	+39 392 005 6345	530995@student.saxion.nl
Bart de Ruiter	0642142477	533200@student.saxion.nl
Catalin Chiprian	+31645509247	528729@student.saxion.nl
Margarita Konnari	+31681413902	527941@student.saxion.nl
Ali Mobini	+31644188301	528592@student.saxion.nl
Ronald Tangelder	+3188019 6347	r.j.w.t.tangelder@saxion.nl
Christiaen Slot	+31657262209	c.g.m.slot@saxion.nl

2. Introduction and Architectural

Overview of the System:

We are a team of students from Saxion University in the Netherlands, engaged in a project that aims to collect and analyse weather data for a weather forecast app. Our stakeholder, Saxion University, allowed us to create Cloudie, a reliable weather station network. Cloudie integrates LoRa sensors, specifically the Dragino LHT65 and the Arduino MKR WAN 1310, paired with the MKR ENV Shield. These sensors are strategically deployed to capture essential atmospheric parameters such as temperature, barometric pressure, humidity, and light intensity.

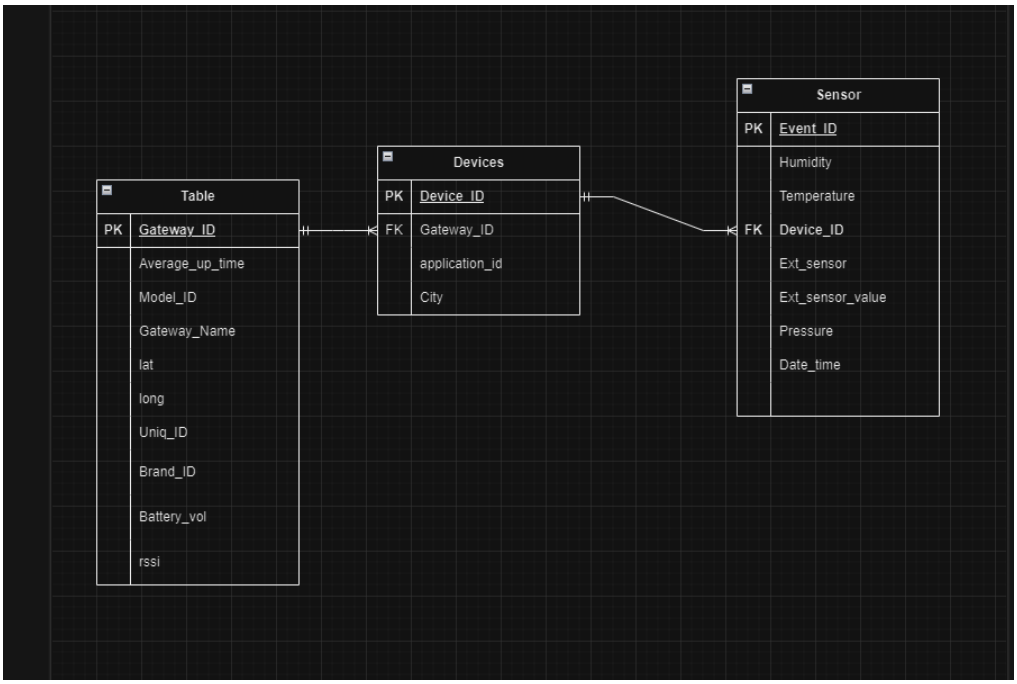
Using the capabilities of LoRaWAN technology, Cloudie transmits the sensor data wirelessly to The Things Network (TTN) through LoRa Gateways. This data is then channeled through an MQTT broker, ensuring seamless integration into our SQL database hosted on Microsoft Azure. The data pipeline is designed to handle large volumes of sensor data with high reliability, enabling real-time weather monitoring.

Project's Purpose:

Development of a weather station application using LoRa sensor nodes for environmental data collection and transmission.

3. Data used

ERD: The ERD shows the database schematic for a LoRa-based weather station project. The 'Device' table holds device identifiers and location data, meaning a relationship with gateways and applications through foreign keys. The 'Sensor' table logs environmental readings like temperature and humidity, tying them to specific devices. A third table, for 'gateways', tracks connectivity and performance metrics. This setup facilitates the collection and association of weather data with specific nodes and locations in the network.



LoRa payload: For our project we had our own Arduino mrkwan 1310, which had four sensors we could use. Those sensors are temperature, humidity, illumination and pressure. Since our objective was to send those four sensors to the The Things Network while using the least amount of bytes we had to design a payload encoder and a payload decoder.

```
float humidity = (float) ENV.readHumidity(); // the max value is: 100 and the lowest value is: 0.
float pressure = (float) ENV.readPressure(MILLIBAR); // the max value is: 1077.5 and the lowest value is: 950.
float illumination = (float) ENV.readIlluminance(); // the max value is: 255 and the lowest value is: 0
```

With these variables we have to get the appropriate precision by our clients specifications.

```
temperatureDecimal = temperatureDecimal - temperatureInteger;
```

```
// round the values down to their respective precision.
temperatureDecimal = round(temperatureDecimal*10)/10.0;
humidity = round(humidity);
pressure = round((pressure-950) * 2)/2;
illumination = round((illumination / 255) * 100);

// create payload for 5 bytes.
byte payload[5] = {};
payload[0] = static_cast<byte>((temperatureDecimal+1) * 10);
payload[1] = static_cast<byte>(humidity);
payload[2] = static_cast<byte>(pressure * 2);
payload[3] = static_cast<byte>(illumination);
payload[4] = static_cast<byte>(temperatureInteger + 127);
```

Precision:

TemperatureDecimal needs to have 1 decimal.

Humidity needs to have 0 decimals.

Pressure needs to go up by .5 increments and since we have to send it as byte the value can't exceed 255 so we remove 950.

Illumination needs to be in % from 0 to 100.

In the code above we want to send five bytes where we send the five variables as followed:

Byte[0] is temperatureDecimal where we add 1 since you can't add 0 * 10 and we do * 10 since you can't send a decimal character as a byte.

Byte[1] is humidity we can send this one as is since the character will always be between 0 or 100 and never will go outside the byte scope.

Byte[2] is pressure we do this * 2 so we can send 17.5 as 35 which makes us not send a decimal character which means we will not go outside the byte scope.

Byte[3] is illumination we can send this one as is since the character will always be between 0 or 100 and never will go outside the byte scope.

Byte[4] is the temperatureInteger which we add 127 so we can also have negative temperature numbers.

```
function Decoder(bytes, port) {
  var decoded = {};

  // bytes[0] contains the decimal part of the temperature value.
  // bytes[1] contains the humidity value.
  // bytes[2] contains the pressure value.
  // bytes[3] contains the illumination value.
  // bytes[4] contains the integer part of the temperature value.

  decoded.temperature = ((bytes[4] - 127) + ((bytes[0] / 10.0) - 1)).toFixed(1);
  decoded.humidity = bytes[1];
  decoded.pressure = ((bytes[2]/2) + 950).toFixed(1);
  decoded.illumination = bytes[3];
  return decoded;
}
```

In The Things Network we receive 5 encoded bytes.

To get the temperature we use bytes[0] and bytes[4] to get both the temperatureInteger and the temperatureDecimal part. We also have to decode the bytes.

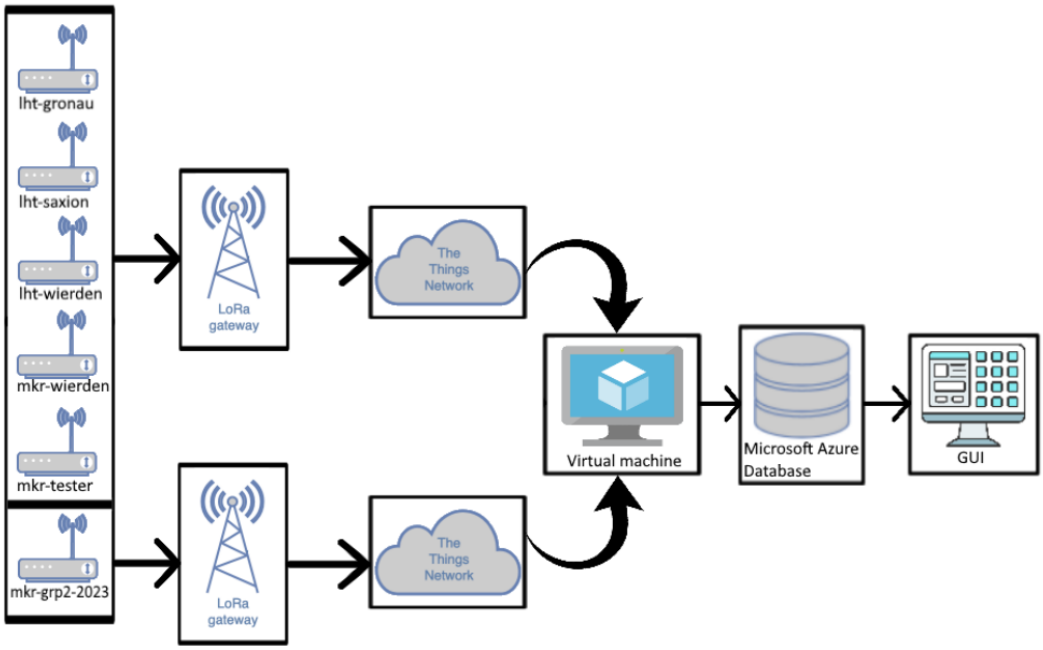
The humidity is byte[1] which we don't have to decode.

The pressure is byte[2] which we have to divide by two and add 950 to decode it to our original value.

The illumination is byte[3] which we don't have to decode.

Data conversions: Our project involved writing a Python script to extract data from The Things Network MQTT and then convert and parse that data before inserting it into an Azure MS SQL database. To accomplish this, we used Python libraries such as paho.mqtt, pymssql, and pyodbc. The Python script was designed to listen for MQTT messages and when it received one, it would parse the message and insert it into the database. All the data that was needed was inserted into the database to their relevant table and data that needed to be updated was updated.

4. Communication Between the Components



Overview (detailed explanation of the system on the image**):**

The architecture is an IoT setup where distributed sensors collect data, which is then aggregated and transmitted via LoRa to a cloud-based network. The data is processed by a server (virtual machine) and stored in a cloud database, with a GUI providing the interface for user interaction and data visualization.

- **Sensors** (lht-gronau, lht-saxion, lht-wierden, mkr-wierden, mkr-tester, mkr-grp2-2023): Those sensers are named according to their location or group, suggesting a distributed setup where multiple sensors are deployed across different locations. sensors send LoRa modulated wireless messages to the gateways. In our case this is a payload of 5 bytes.
- **LoRa Gateways**: Receive messages from the sensors and forward them to the Network Server. The sensors will communicate with nearby gateways and each gateway is connected to the network server this means the sensors don't need to peer with specific gateways. Messages sent from end devices travel through all gateways within range.
- **The Things Network**: The messages picked up by the LoRa Gateways are received by the Network Server. If the Network Server has received multiple copies of the same message, it keeps a single copy of the message and discards others. This is known as message deduplication. After deduplication we decode the message, we received by using our decoder function. The decoder function will translate the bytes into usable data.
- **Virtual Machine**: After the data is routed through TTN, it is sent to a virtual machine. It acts as a server that can process and filter the data before storing it.
- **Microsoft Azure Database:** Processed data is then stored in a Microsoft Azure-hosted database, which provides scalable and secure cloud storage. The database is used for persistent storage and can be accessed for further analysis.

- **GUI**: Finally, there's a GUI component that likely serves as a dashboard or control panel for users to interact with the system. Th further explanation of this part you can see in chapter 4.

5. Explanation of Key Concepts

The most important code snippet is the WeatherStats object which sets up all the data used throughout the application.

The CityLoad function loads all the cities and their gateway from the database.

The GateWayLoad function loads all the information about the gateways.

The DataLoad function loads all the data entries for the specific city given.

```
        /// <summary>
        /// Loads the names of the cities and their gateways.
        /// </summary>
        public async Task CityLoad()
        {
            try
            {
                string query = "SELECT * FROM DATA.Devices";
                var dt = await ExecuteQuery(query);

                Cities.Clear();

                    foreach (DataRow row in dt.Rows)
                    {
                        Cities.Add(row["City"].ToString());
                        GateWayCities.Add(row["Gateway_ID"].ToString());
                    }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

```
        /// <summary>
        /// Loads the gateway data.
        /// </summary>
        public async Task GateWayLoad()
        {
            try
            {
                string query = "SELECT * FROM DATA.Gateway";
                var dt = await ExecuteQuery(query);

                GateAwaysData.Clear();

                    foreach (DataRow row in dt.Rows)
                    {
                        var name = new WeatherData
                        {
                            Gateway_ID = row["Gateway_ID"].ToString(),
                            Avg_Up_Time =
Convert.ToDouble(row["Average_Up_Time"]),
                            Model_ID = row["MODEL_ID"].ToString(),
                            Lat = Convert.ToDouble(row["Lat"]),
                            Long = Convert.ToDouble(row["Long"]),
                            Uniq_ID = row["Uniq_ID"].ToString(),
                            Brand_ID = row["Brand_ID"].ToString(),
                            RSSI = row["rssi"].ToString(),
                            Battery_voltage =
Convert.ToDouble(row["Battery_vol"]),
                            Altitude = row["Alt"] == DBNull.Value ? double.NaN :
Convert.ToDouble(row["Alt"])
                        };
                        GateAwaysData.Add(row["Gateway_ID"].ToString(),name);
                    }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

```
        /// <summary>
        /// Loads the data for a specific city.
        /// </summary>
        /// <param name="City">The city.</param>
        public async Task DataLoad(string City)
        {
            if (City.EndsWith("-outside"))
            {
                City = City.Replace("-outside", "");
                Outside = true;
            }
            try
            {
                string query = "select * from DATA.Sensor as s join DATA.Devices as
d on s.Device_ID = d.Device_ID WHERE s.Date_Time >= DATEADD(day,-10, GETDATE()) AND
d.City = @City";
                var parameters = new List<SqlParameter> { new SqlParameter("@City",
City) };
                var dt = await ExecuteQuery(query, parameters);

                AverageData.Clear();
                    Data.Clear();
                    dt.DefaultView.Sort = "Date_Time ASC";

                    foreach (DataRow row in dt.Rows)
                    {
                        if (row["Ext_Sensor_Name"].ToString() == "Temperature
Sensor" && Cities.IndexOf(row["Device_ID"].ToString() + "-outside") == -1)
                        {
                            Cities.Add(row["Device_ID"].ToString() + "-outside");
                            //Find index of Device_ID in cities and copy that index
in Gateawaycities to the end of Gateawaycities

GateWayCities.Add(GateWayCities[Cities.IndexOf(row["Device_ID"].ToString())]);

                        }
                        var name = new WeatherData();

                        if (Outside)
                        {
                            name.Temperature = row["Ext_Sensor_Value"] ==
DBNull.Value ? double.NaN : Convert.ToDouble(row["Ext_Sensor_Value"]);
                            name.Ext_Sensor_Name = "Null";
                            name.Ext_Sensor_Value = double.NaN;
                        }
                        else
                        {
                            name.Temperature = row["Temperature"] == DBNull.Value ?
double.NaN : Convert.ToDouble(row["Temperature"]);
                            name.Ext_Sensor_Name = row["Ext_Sensor_Name"] ==
DBNull.Value ? "Null" : row["Ext_Sensor_Name"].ToString();
                            name.Ext_Sensor_Value = name.Ext_Sensor_Name ==
"Temperature Sensor" ? double.NaN : (row["Ext_Sensor_Value"] == DBNull.Value ?
double.NaN : Convert.ToDouble(row["Ext_Sensor_Value"]));
                        }

                        name.Humidity = row["Humidity"] == DBNull.Value ? double.NaN
: Convert.ToDouble(row["Humidity"]);
```

```
                        name.Device_ID = row["Device_ID"].ToString();
                        name.Pressure = row["Pressure"] == DBNull.Value ?
double.NaN : Convert.ToDouble(row["Pressure"]);
                        name.Date = Convert.ToDateTime(row["Date_Time"]);
                        Data.Add(name);
                    }

                    var a = Data.GroupBy(x => x.Date.Day)
                        .Where(group => group.Any(x =>
!double.IsNaN(x.Temperature)))
                        .Select(group =>
                        {
                            var temperatures = group.Where(x =>
!double.IsNaN(x.Temperature)).ToList();
                            return new
                            {
                                Day = group.Key,
                                AvgTemp = temperatures.Average(x =>
x.Temperature)
                            };
                        }).ToList();

                    foreach (var item in a)
                    {
                        var x = Data.Where(x => x.Date.Day ==
item.Day).FirstOrDefault();
                        x.AvgTemp = item.AvgTemp;
                        AverageData.Add(x);

                    }

                    Outside = false;

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

    }
```

```
        /// <summary>
        /// Executes the database connection and query execution.
        /// </summary>
        /// <param name="query">The query.</param>
        /// <param name="parameters">The parameters.</param>
        /// <returns>The data table.</returns>
        private async Task<DataTable> ExecuteQuery(string query,
List<SqlParameter> parameters = null)
        {
            using (SqlConnection connection = new
SqlConnection(ConnectionString))
            {
                await connection.OpenAsync();
                SqlCommand command = new SqlCommand(query, connection);
                if (parameters != null)
                {
                    command.Parameters.AddRange(parameters.ToArray());
                }
                SqlDataReader reader = await command.ExecuteReaderAsync();
                DataTable dt = new DataTable();
                dt.Load(reader);
                return dt;
            }
        }
```

The second most important snippet would be how changing pages works. I have a class called NavigationStore

```
using Cloudie.ViewModel;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Cloudie.Store
{
    /// <summary>
    /// Represents the navigation store.
    /// </summary>
    public class NavigationStore
    {
        private ViewModelBase _currentViewModel;

        /// <summary>
        /// Gets or sets the current view model.
        /// </summary>
        public ViewModelBase CurrentViewModel
        {
            get => _currentViewModel;
            set
            {
                _currentViewModel = value;
                OnCurrentViewModelChanged();
            }
        }

        /// <summary>
        /// Called when the current view model changes.
        /// </summary>
        private void OnCurrentViewModelChanged()
        {
            CurrentViewModelChanged?.Invoke();
        }

        /// <summary>
        /// Occurs when the current view model changes.
        /// </summary>
        public event Action CurrentViewModelChanged;
    }
}
```

This class stores the CurrentViewModel and I can change the viewmodels using this class. I first have to set it up in my App class

```
using Cloudie.Model;
using Cloudie.Store;
using Cloudie.ViewModel;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace Cloudie
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        private readonly NavigationStore _navigationStore;
        private readonly WeatherStats _weatherStats;

        public App()
        {
            _weatherStats = new WeatherStats();
            _navigationStore = new NavigationStore();
        }

        protected override void OnStartup(StartupEventArgs e)
        {
            _navigationStore.CurrentViewModel = new
MainPageViewModel(_navigationStore, _weatherStats);
            MainWindow = new MainWindow()
            {
                DataContext = new MainView(_navigationStore)
            };

            MainWindow.Show();
        }
    }
}
```

And then I link every button that represents a page change to an ICommand that calls the navigationstore's currentviewmodel and changes it. Let's take as an example ToGraphsCommand

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Cloudie.ViewModel;
using Cloudie.Model;
using Cloudie.Store;

namespace Cloudie.Commands
{
    /// <summary>
    /// Represents the command to navigate to graphs.
    /// </summary>
    public class ToGraphsCommand : CommandBase
    {
        private readonly WeatherStats _weatherStats;
        private readonly NavigationStore _navigationStore;

        /// <summary>
        /// Initializes a new instance of the <see cref="ToGraphsCommand"/> class.
        /// </summary>
        /// <param name="navigationStore">The navigation store.</param>
        /// <param name="weatherStats">The weather stats.</param>
        public ToGraphsCommand(NavigationStore navigationStore, WeatherStats
weatherStats)
        {
            _weatherStats = weatherStats;
            _navigationStore = navigationStore;
        }

        /// <summary>
        /// Executes the command to navigate to graphs.
        /// </summary>
        /// <param name="parameter">The command parameter.</param>
        public override async void Execute(object parameter)
        {
            _navigationStore.CurrentViewModel =
            new WeatherGraphViewModel(_navigationStore, _weatherStats);
        }
    }
}
```

As you can see I pass the global NavigationStore (which I initialized in App) to the ToGraphsCommand's constructor and in the execute function I change the CurrentViewModel to the WeatherGraphViewModel, so the UI gets updated as well and there is a different page now.

6. Explanation of the GUI(s)

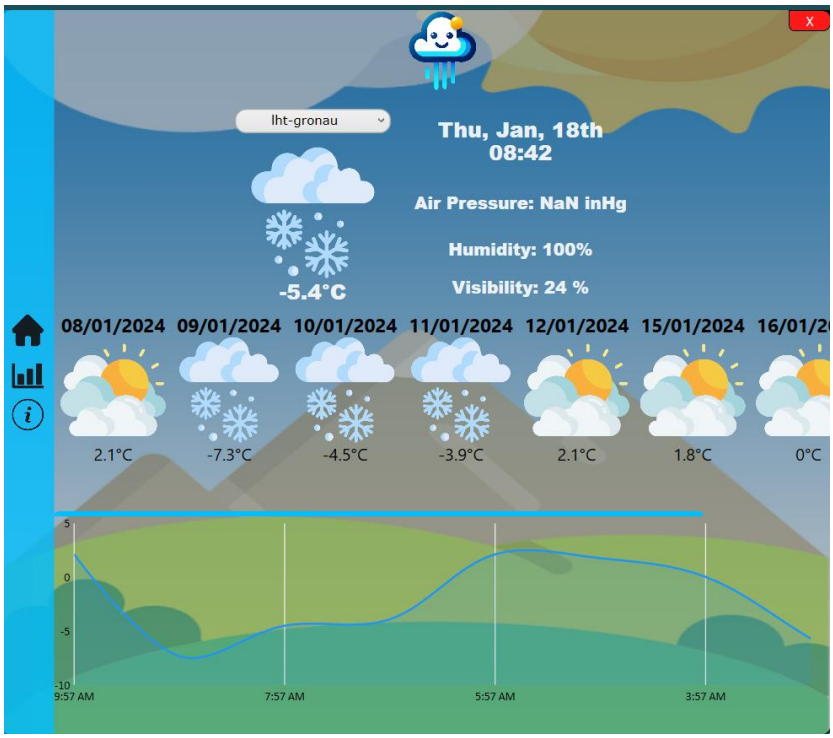
The Cloudie Application UI was developed using a colorful pallet and icons, making the presentation of weather statistics both accurate and esthetically pleasing. The application consists of 3 pages, each one of them having its unique features.

Some reoccurring features worth mentioning are the navigation bar that is used to traverse each page and the responsiveness of the application, allowing for the resizing of the window without any unwanted buggy displaying.

5.1 Weather Statistics Page:

The page consists of several key elements that enable the user to view short information about the current weather registered in all sensors used. Device selection is present at the top of the page, acting as a controller for the weather statistics displayed. Additional information for the current time and registered values are displayed in a short format for the user to quickly check the weather information.

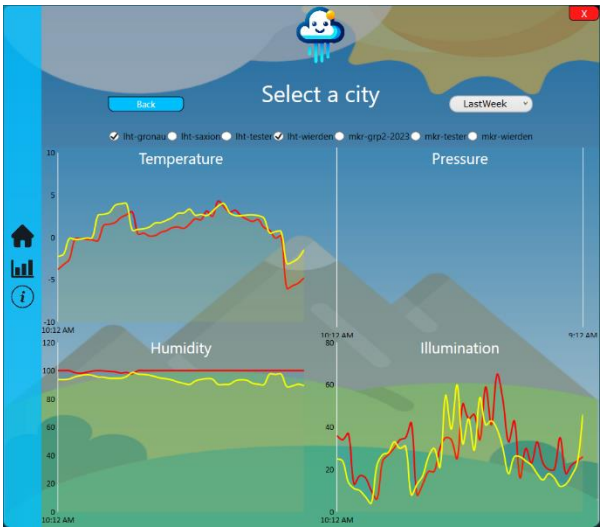
In the middle of the page, a short weather history is displayed, showing for each device multiple recordings done in the past to visualize the changes in weather conditions. At the bottom of the page, a graph can be viewed, displaying the average temperature for a specific period for each device available.



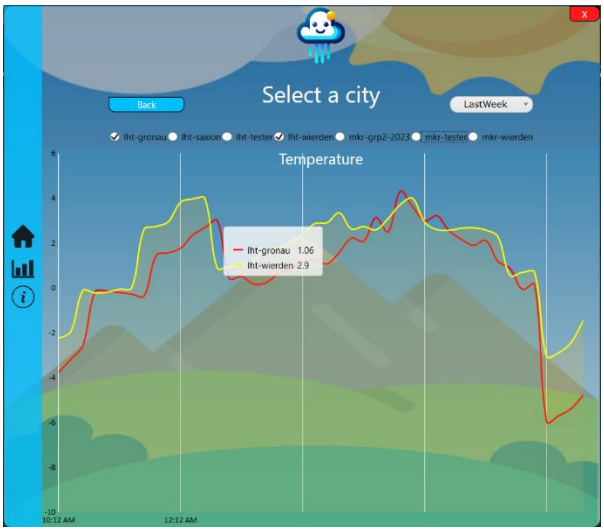
5.2 Weather Graphs Page:

As an important feature of the Cloudie weather application, multiple charts are present, making the visualization of gathered data pleasant and considerably increasing the readability. The charts can display multiple device recordings at the same time, making it possible to compare different values from around the Netherlands and neighboring zones.

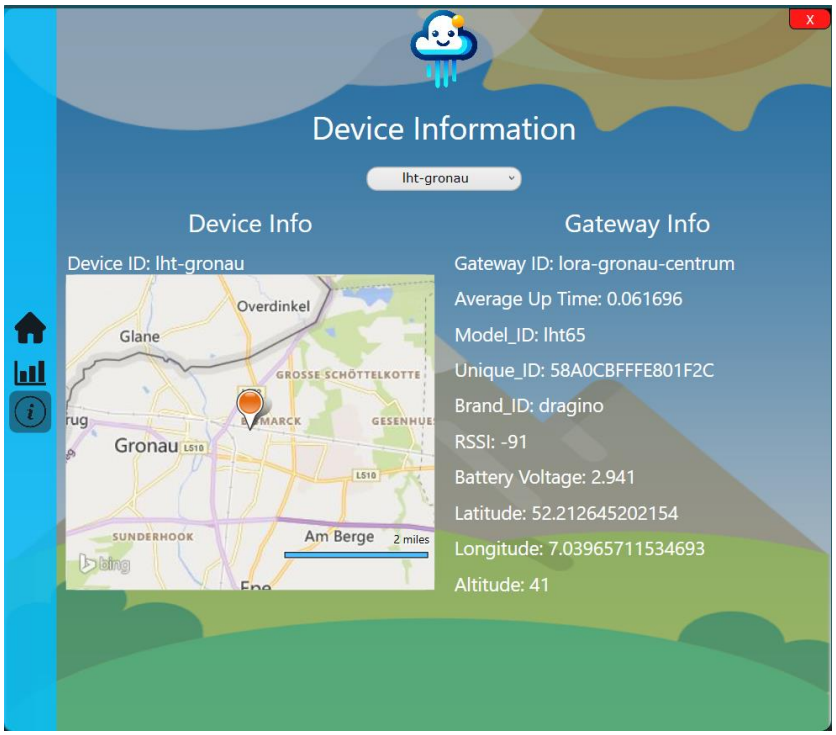
Filtering the data is also possible for users as they can choose different time periods. To better visualize a specific sensor recording, the user can press on the chart they want to inspect, making the chart bigger and more readable, while still maintaining the other features of the page.



5.3 Device Information Page:



Cloudie shows a high level of transparency by displaying the location of every device used on a very versatile and interactive Bing Map. The device selection is intuitive and consistent with previous page designs. Along with the device information and location visualization, the user can see information regarding the gateway used to receive data from the devices.



7. How Everything is Tested

We have another CloudieTesting project linked to our Cloudie project. In the CloudieTesting project we test the most important parts of the whole program, WeatherStats and NavigationStore, because if one of these does not work then the whole application does not work.

This is our WeatherStatsTest

```
using Cloudie.Model;

namespace CloudieTesting
{
    public class WeatherStatsTests
    {
        private WeatherStats _weatherStats;

        public WeatherStatsTests()
        {
            _weatherStats = new WeatherStats();
        }

        [Fact]
        public void Cities_WhenNew_ShouldNotBeNull()
        {
            Assert.NotNull(_weatherStats.Cities);
        }

        [Fact]
        public void GateWayCities_WhenNew_ShouldNotBeNull()
        {
            Assert.NotNull(_weatherStats.GateWayCities);
        }

        [Fact]
        public void GateAwaysData_WhenNew_ShouldNotBeNull()
        {
            Assert.NotNull(_weatherStats.GateAwaysData);
        }

        [Fact]
        public async Task CityLoad_ShouldLoadCities()
        {
            await _weatherStats.CityLoad();

            Assert.NotEmpty(_weatherStats.Cities);
        }

        [Fact]
        public async Task GateWayLoad_ShouldLoadGateways()
        {
            await _weatherStats.GateWayLoad();

            Assert.NotEmpty(_weatherStats.GateWayCities);
        }

        [Fact]
        public async Task DataLoad_ShouldLoadDataForCity()
        {
            string city = "lht-saxion";
            await _weatherStats.DataLoad(city);

            Assert.NotEmpty(_weatherStats.Data);
            Assert.NotEmpty(_weatherStats.AverageData);
        }
    }
}
```

+

And this is our NavigationTest (again, no need to double click on the code, fully visible)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Cloudie.Store;
using Cloudie.View;
using Cloudie.Model;
using Cloudie.ViewModel;

namespace CloudieTesting
{
    public class NavigationTest
    {
        private NavigationStore _navigationStore;
        private WeatherStats _weatherStats;

        public NavigationTest()
        {
            _navigationStore = new NavigationStore();
            _weatherStats = new WeatherStats();
        }

        [Fact]
        public void CurrentViewModel_WhenNew_ShouldBeNull()
        {
            Assert.Null(_navigationStore.CurrentViewModel);
        }

        [Fact]
        public void CurrentViewModel_WhenSet_ShouldUpdateCurrentViewModel()
        {
            var expected = new WeatherGraphViewModel(_navigationStore,
_weatherStats);

            _navigationStore.CurrentViewModel = expected;

            Assert.Equal(expected, _navigationStore.CurrentViewModel);
        }
    }
}
```

8. Short Manual

8.1 Arduino setup manual

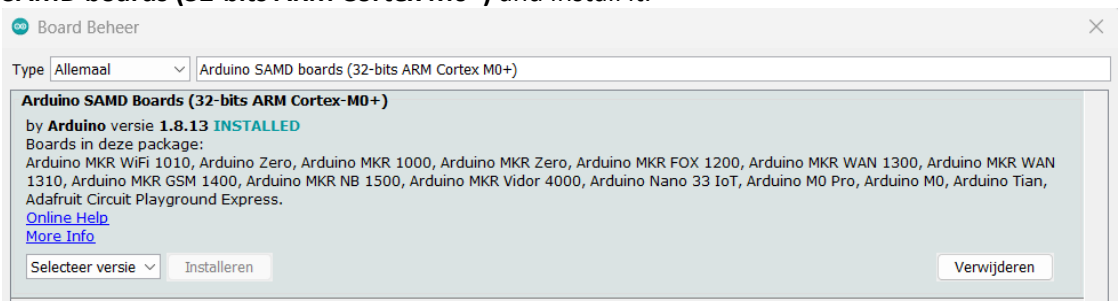
Setting up the Arduino file:

1. Inside of the repository go inside the map **Arduino** and download the file latest version of **final_ver_grp2_arduino_mkr_1310.ino**
2. (Before doing this step go to **Installing the drivers:** and do step 1 and 2)
   Open the file **final_ver_grp2_arduino_mkr_1310.ino** in the Arduino IDE.

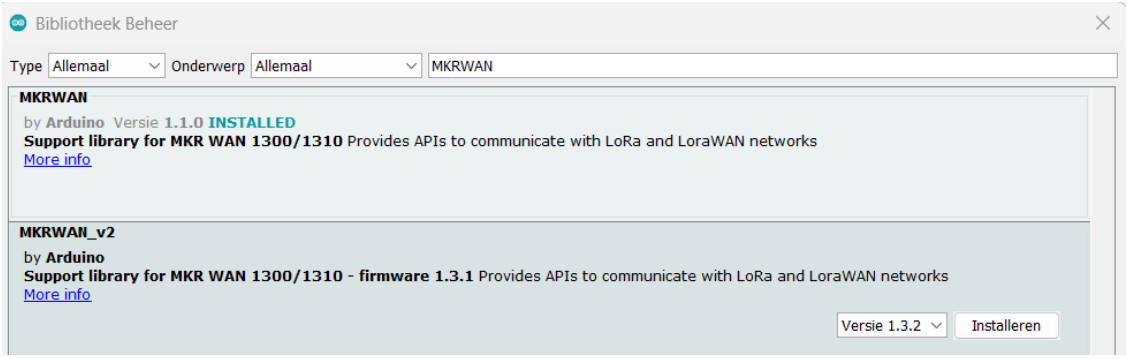
3. (Before doing this step complete **Installing the drivers:**)
   Run the file **final__ver_grp2_arduino_mkr_1310.ino** if the file doesn't compile go back to step 1 and redo **Installing the drivers:.**
4. Check the LED and compare your result to the **LED settings:**.
5. Succes the Arduino setup is complete.

Installing the drivers:

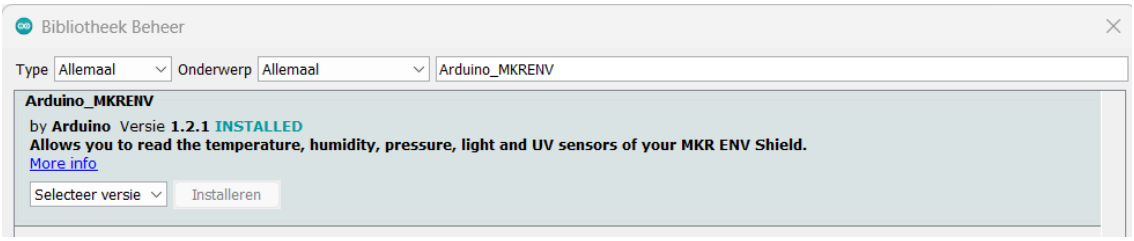
1. Download and install the Arduino IDE from this link: https://www.arduino.cc/en/software
2. Startup the Arduino IDE and go to **Tools > Board > Board Manager....** Here we need to look for the **Arduino SAMD boards (32-bits ARM Cortex M0+)** and install it.



3. Now we need to install the library **Tools > Manage libraries..**, and search for **MKRWAN** and install it (we are using the MKWRAN and not the MKWRAN_v2).



4. Now we need to install the library **Tools > Manage libraries..**, and search for **Arduino_MKRENV** and install it.



5. Now you have to select the board **Tools > Board > Arduino SAMD boards (32-bits ARM Cortex M0+) Boards > Arduino MKR WAN 1310**
6. Now you have to select the port **Tools > Port > COM (Arduino MKR WAN 1310)**
7. Now the driver setup is complete.

LED settings:

- If the LED stays on continuously there was an error connecting to the specified frequency plan (EU868).
- If the LED continuously blinks for 1 second it can't connect to the things network.
- If the LED blinks every 200 milliseconds there was an error sending the packet to the things network.

- If the LED turns on for 2 seconds and then turns off the packet has been sent to the things network.

8.2 MQTT Data Receiver Manual

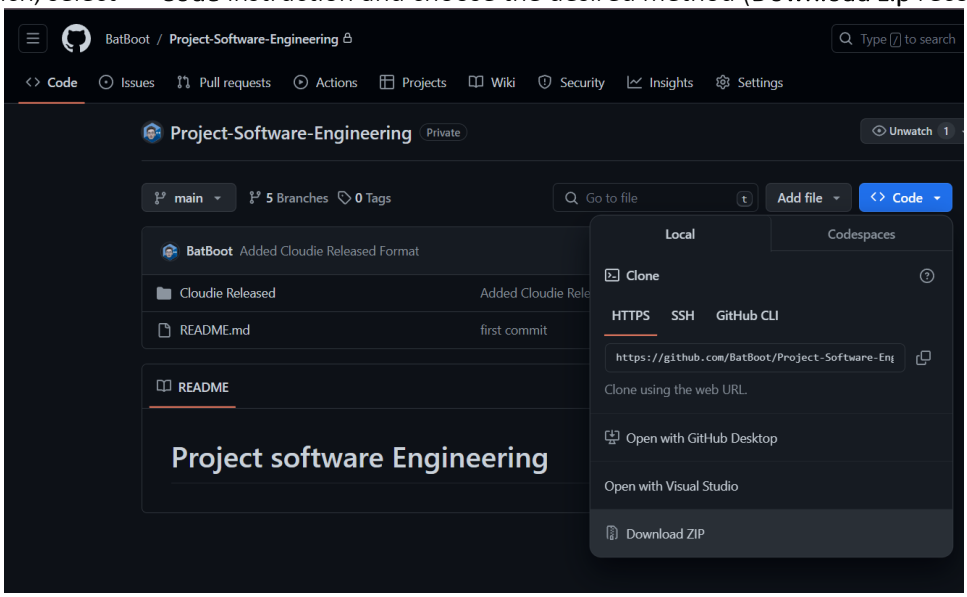
Installing The Requirements:

1. From GitHub repository find folder mqtt_data
2. Install python on your system.
3. Then install pip on your system.
4. Run **pip install -r requirement.txt**.
5. Run mqtt.py script for all the other sensor " python mqtt.py ".
6. Run mqtt_sen.py script for group sensor " **python mqtt_sen.py** ".

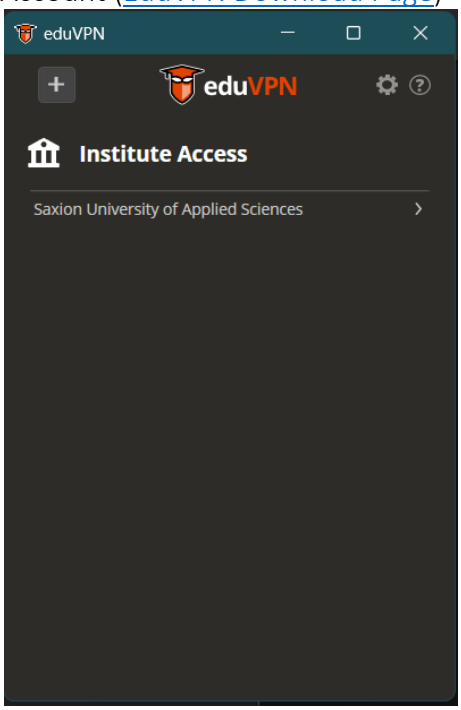
8.3 Cloudie Windows Application manual

Installing Cloudie:

1. Access the GitHub repository on this link: BatBoot/Project-Software-Engineering.
2. On the **main** branch, select **<> Code** instruction and choose the desired method (**Download zip** recommended)



3. To connect to the database when running the app, one condition out of the two is necessary:
   - Be connected on Eduroam inside Saxion;
   - Use the **EduVPN** with a Saxion Account (EduVPN Download Page)



8.4 The Things Network manual

1. go to https://weatherproject.eu2.cloud.thethings.industries/console

2. enter the username: projectadmin and password: adminProject98*

3. go to applications and select weatherapplication.

At live data you can see the data that our TTN receives.

If you click on End devices then select mkr-grp2-2023 and click on payload formatters you can see the decoder.

9. Recommendations

What bugfixes or added functionality would you implement if you had more time or another sprint?

- API and Security: If there were an additional sprint, a primary focus would be the implementation of an API and security enhancements for it. Integrating an API within the .NET framework of our project offers multiple

advantages (The API provides a secure gateway to the data, and it simplifies interactions between the front-end and the database). We had a program with API ready, but the API had to be done from the beginning, as we weren't able to implement it with the final program, since the program wasn't asynchronous when integrating the API. When programming the program, we should have been aware of details which are supposed to be taken into consideration before implementing API.

- Graphical User Interface (GUI): In perspective, GUI can be significantly improved to enhance user experience. Focusing on the GUI's usability will make the application not only more accessible but also more enjoyable to use.
- Graph Presentation: Our sprint review showed inaccuracies in graph labels across different charts, and based on the sprint experience we could suggest that the graph presentation would go more smoothly if we paid more attention to the details of the GUI user-experience. Improving the clarity and accuracy of graph presentations is crucial for users' comfort and for ones that relied on accuracy of our application.

10. List of references

- Navigation setup for C# wpf: https://youtu.be/bBoYHl3pLEo?si=IKPhjacDn_hgXlkX
- Database connection in C#: https://youtu.be/kBqkAZyWWUM?si=diBl84BytsOzxI_X
- Linking ComboBox to database: https://youtu.be/1sDQ7JEEzp8?si=TwvhVVq_cneekxa7
- Arduino mkr-env-shield: https://docs.arduino.cc/tutorials/mkr-env-shield/mkr-env-shield-basic#lps22hb-atmospheric-pressure-sensor
- Arduino mkrwan 1310 the things network: https://docs.arduino.cc/tutorials/mkr-wan-1310/the-things-network
- Arduino mkrenv library: https://www.arduino.cc/reference/en/libraries/arduino_mkrenv/
- Setting up the mkrwan with the things network: https://www.hackster.io/ZoLuSs/first-step-with-mkr-wan-1310-and-the-things-networks-9f6065
- The things network coverage: https://www.thethingsnetwork.org/map