FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

IoT Final Project Report
Group B

Members

Shabbir Hasan (2208301)
Fahim Muhtasim Hossain (Y69305376)
Kazi Nymul Haque (2207329)
Ijlal Khan (Y69325871)
Muhammad Syed (2209785)

Instructor

Abhishek Kumar

Technical Advisor

Huong Nguyen
Ummi Latif

January 2023

TABLE OF CONTENTS

1 INTRODUCTION

The use of IoT (Internet of Things) agriculture sensors in the farming industry has the potential to greatly improve crop yields, increase efficiency, and reduce waste. These sensors are equipped with various types of sensors that can collect data on important factors such as soil moisture levels, temperature, humidity, and light intensity. By analyzing this data, farmers can optimize irrigation systems, monitor crop health, and predict weather patterns. Additionally, by connecting these sensors to the internet, the data they collect can be accessed and analyzed remotely.

Water is a vital resource for agriculture, as it is necessary for the growth of crops and the raising of livestock. However, water is a finite resource and it is essential to use it efficiently in order to sustainably feed a growing global population. There are several reasons why water conservation is important in agriculture:

a. **Water scarcity:** Many regions around the world are experiencing water scarcity, making it crucial to use this resource efficiently.
b. **Climate change:** Climate change is affecting the availability and distribution of water, making it more important than ever to use water wisely.
c. **Cost:** Water is often a significant cost for farmers, and using it efficiently can help reduce these costs.
d. **Environmental impact:** Irrigation can have a significant impact on the environment, both in terms of water usage and the potential for pollution. Using water efficiently can help mitigate these impacts.

By using water efficiently, farmers can help ensure that this vital resource is used sustainably and help to meet the growing demand for food while preserving the environment. The aim of this thesis is to develop a cost-effective IoT hardware solution that is affordable for farmers and provides a high rate of return in terms of both yield and input cost. The goal of this project is to utilize IoT technology, specifically soil moisture sensors, to aid in water conservation efforts within the agricultural industry.

2 SYSTEM ARCHITECTURE

2.1 Figure of Overall System Connectivity

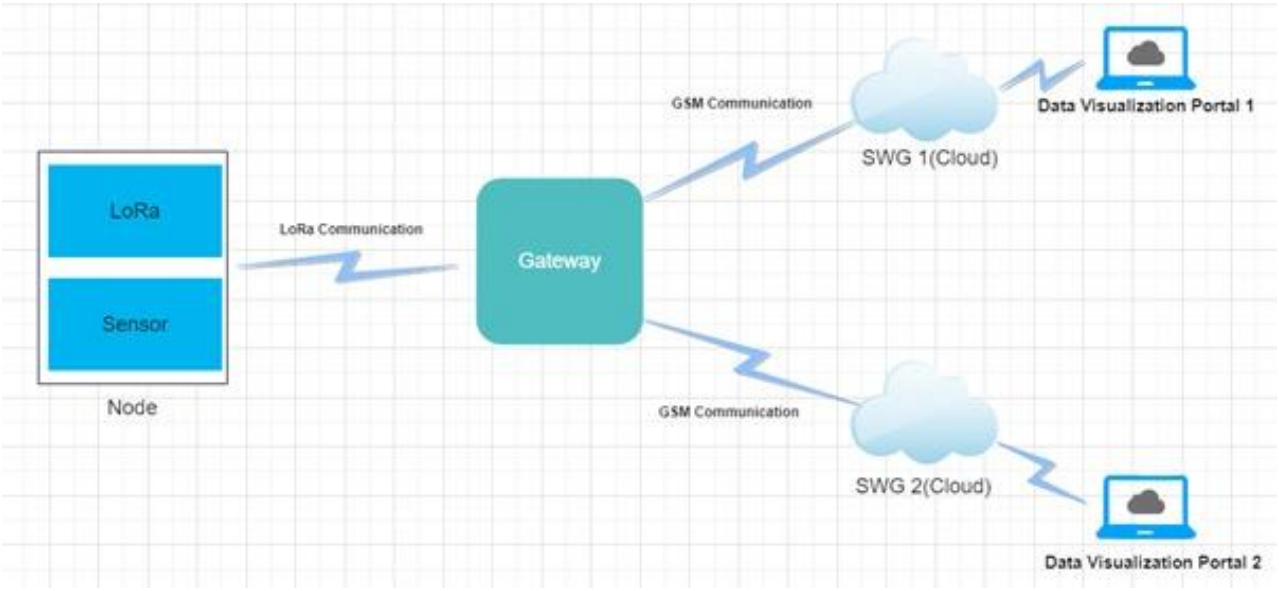

Fig 2.1: Generic architecture of the IoT solution

2.2 Workflow

As described in figure, this project consists at three components

1. Node (Field sensing instruments)
2. Gateway (Central reporting agent)
3. Cloud/Portal (Data visualization service)

Collection of data starts from Node that send field data periodically to the gateway over LoRa communication protocol. Multiple nodes install in field send data independently to gateway on its turns. Gateway first receive data from multiple nodes and stores in buffer. Based on first come first serve base, gateway transform received data to given cloud format and send it to cloud through GSM 2G/3G connectivity. On the cloud, portal (Smart Water Grid) fetch data from the cloud and display it to farmer for real-time assistance. Further details of each component are separately illustrated below.

2.3 Communication Protocol

Overall system is handling two major type of communication protocol. First one is for the backward compatibility (SWG-1) and second is for the upgraded packet to support integration with new portal (SWG-2).

Since, new device is supporting wireless communications between nodes and gateway over LoRa channel, therefore, first protocol is further split into two parts as mentioned in below section 2.3.1.

2.3.1 Communication Protocol for Backward Compatibility

Node is sending data packet to Gateway as per custom protocol especially designed for testing of multiple node integration with gateway over lora channel as mentioned in section 2.3.1.1. Main objective of this protocol is to keep low power consumption at node device and maximum data integrity at gateway side. After successful reception of data at gateway, it decodes node sensors data into old cloud/portal (SWG-1) format as mentioned in section 2.3.1.2.

2.3.1.1 Communication Protocol between Node and Gateway

	Synch word	Node ID	Data Byte 1	Data Byte 2	Data Byte 3	Data Byte 4	CRC
No. of Bytes	1	1	1	1	1	2	2

2.3.1.2 Communication Protocol between Gateway and Cloud/Portal (SWG-1)

	Header	IMEI	Portal IP	Signal Strength	Bit Error Rate (BER)	Device Voltage	Sensor value 1	Sensor value 2	Time	CRC
No. of Bytes	1	14	12	2	2	4	3	2	12	2

2.3.2 Communication Protocol for New Portal (SWG-2)

New protocol is designed to overcome the limitations of old format as it was only support two sensors. New protocol support multiple sensors integration with single device and able to connect large number of devices with single gateway. Below is the communication protocol floating between Node, Gateway, and Cloud/ Portal (SWG-2). Further details of this protocol are available in section 3.4.

Header	@som\r\n	
Device Info	MessageVersion\r\n	DeviceID\r\n MsgTimeString\r\n
Device parameters	Diagnostics\r\n	Dcodenumber,Time, Content\r\n
Sensor Data	Readings\r\n	Time, SensorName, SensorValue\r\n
Inner message	InnerMessages\r\n	DeviceMessage (Optional)
Packet validity	CRC\r\n	Value
Footer	@eom\r\n	

Fig 2.2: Packet structure of transported data

3 IMPLEMENTATION

3.1 Client Requirements

The client requires a system that can handle data from multiple sensors with varying storage capacities and has internet connectivity at each node to transfer data to the cloud. There are also constraints on the use of the GSM module and considerations for hardware reliability due to environmental factors. The client has provided the following requirements and constraints for the project:

3.1.1 Previous Problems

1. The system should be able to handle data from multiple sensors, each with different maximum capacities for data storage (e.g., sensor one holds up to four digits while sensor two holds up to three digits).
2. Every node must have internet connectivity to transfer data to the cloud.
3. The old system has hardware reliability issues due to environmental factors.
4. The GSM module (SIM 900D) is unavailable for use in the system.

3.1.2 Hardware Requirements

The hardware requirements for the system include:

1. **Smart power management:** This refers to the ability of the system to intelligently manage power usage to optimize performance and conserve energy.
2. **Smart battery charging:** This refers to the ability of the system to optimize the charging of batteries to extend their lifespan and improve efficiency.
3. **Multiple communication buses:** The system should be able to communicate using multiple protocols, such as I2C, SPI, and 485.
4. **Telemetry/debug:** The system should have multiple modes for telemetry (gathering and transmitting data) and debugging (identifying and fixing issues).
5. **Onboard SD-card data logging:** The system should have the capability to log data on an onboard SD card in case of connectivity failures.
6. **Smart management of GSM connectivity:** The system should be able to effectively manage GSM connectivity to ensure stable communication.
7. **System monitoring:** The system should have the ability to monitor its own performance and identify any issues that may arise.
8. **LoRa connectivity to nodes:** The system should be able to communicate with nodes using the LoRa protocol.
9. **Internal and external watchdog:** The system should have both internal and external watchdog mechanisms to ensure reliable operation and detect and recover from failures.

3.1.3 Software Requirements

The software requirements for the system include:

1. **Modular code:** The system's code should be modular, meaning it should be divided into self-contained units that can be easily reused and modified.
2. **Portable code:** The system's code should be portable, meaning it should be able to run on different platforms or environments with minimal changes.
3. **Distributed theme:** The system should have a distributed theme, meaning it should be designed to operate and function across multiple devices or locations.
4. **Explicit handlers of known problems:** The system should have explicit handlers in place to address known problems or issues that may arise.
5. **Support for remote debugging of firmware:** The system should support remote debugging of firmware, allowing for the identification and fixing of issues from a remote location.
6. **Fault-tolerant system:** The system should be designed to be fault-tolerant, meaning it should be able to continue functioning even if some components fail.
7. **Self-diagnostics and remedy action:** The system should be able to diagnose problems and take remedial action to resolve them.
8. **Offline fault logging:** The system should have the capability to log faults or errors that occur while it is offline.

3.1.4 Constraints

The constraints for the system include:

1. **Support for Arduino libraries:** The system must be compatible with and able to utilize Arduino libraries.
2. **Backward compatibility with the old Smart Water Grid (SWG**): The system must be backward compatible with the old SWG to ensure seamless integration.
3. **Support for multiple sensors at a single node:** The system must be able to support the use of multiple sensors at a single node.
4. **Handshaking protocol with the node and gateway:** The system must have a handshaking protocol in place to facilitate communication between the node and the gateway. This protocol ensures that the node and gateway are able to communicate and exchange data correctly.

3.2 Sprints Requirements

The sprint design for the system was focused on two goals: the development of new hardware that incorporates all user requirements and the integration of the new gateway with the old portal. The sprint was planned for six weeks with the aim of deploying the new device in the field.

The tasks for the sprint included:

1. **Development of new hardware:** This involved designing and building new hardware that meets all of the user's hardware requirements, including smart power management, smart battery charging, and multiple communication buses.
2. **Wireless sensing nodes for testing:** In order to test the new hardware, wireless sensing nodes were developed for use in the field.
3. **Firmware development:** The firmware for the new hardware was developed to ensure that it functioned properly and met all of the user's requirements.
4. **Integration of the new gateway with the old portal:** The new gateway was integrated with the old portal to ensure backward compatibility and seamless operation.
5. **Backward compatibility of the new gateway at the new portal:** The new gateway was tested for backward compatibility with the new portal to ensure that it could operate seamlessly with the updated system.

Week 1: Hardware designing
Week 2: Hardware designing
Week 3: Firmware designing
Week 4: Firmware designing for Hardware
Week 5: Firmware designing + Hardware testing at client side
Week 6: Firmware designing for old cloud/portal (SWG-1)

3.3 Hardware Development

The hardware for the system was developed based on the following components:

a) **Gateway:** The gateway is the central reporting agent that receives data from the nodes, stores it in a buffer, and sends it to the cloud/portal in a predetermined format using GSM 2G/3G connectivity.

Fig 3.1: Actual Gateway device view

b) **Test node:** The test node is a wireless sensing device that was developed for use in the field to test the functionality of the new hardware.

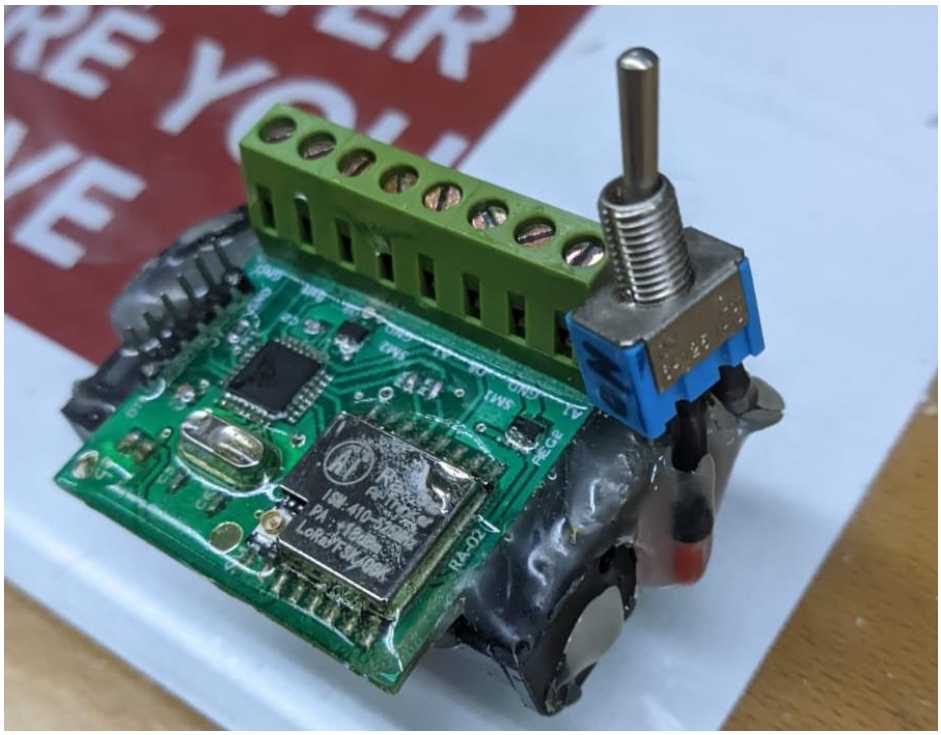

Fig 3.2: Actual Test node device View

c) **Upgraded node:** The upgraded node is a field sensing instrument that has been upgraded to meet the user's requirements, such as support for multiple sensors and handshaking protocols with the gateway. These nodes are responsible for collecting field data and sending it to the gateway using the LoRa communication protocol.

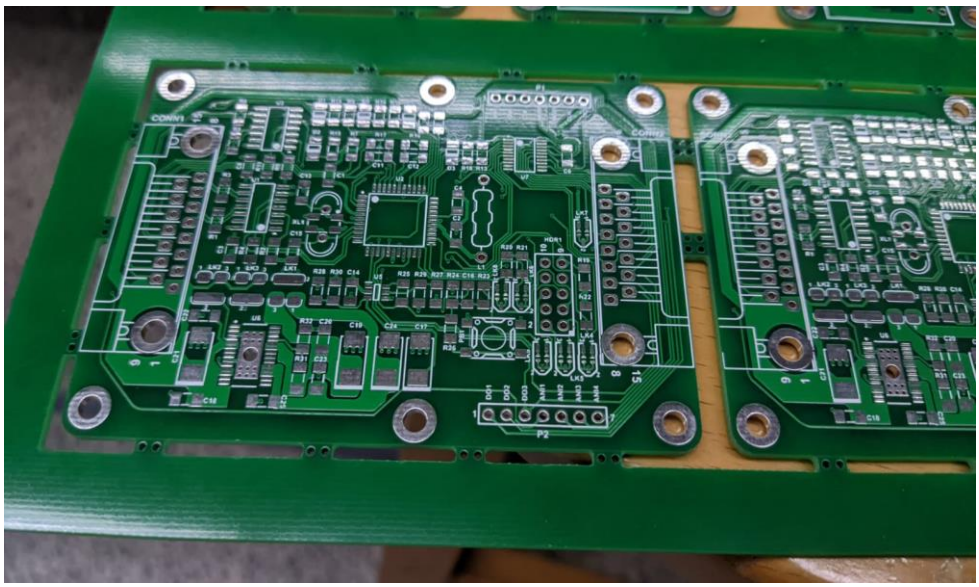

Fig 3.3: Actual upgraded node PCB view

3.4 Firmware Development

The system was designed in two sections and four layers to meet the user requirements:

3.4.1 Application Section

a) **Application Layer:** The application layer of the system contains information about individual applications and enables users to activate multiple applications that may be running concurrently. This layer is responsible for end-to-end connectivity of the device and enables the integration of multiple application flows, such as connectivity with multiple servers and connected sensors. The primary application in this layer is system monitoring and diagnosis, which involves the continuous monitoring of the system to identify and troubleshoot any issues that may arise. This layer is critical for ensuring the smooth operation of the system and its various components.

b) **Application Drivers:** The application drivers layer is an optional layer that is used to define drivers that handle connectivity for the application layer. This layer contains the API (Application Programming Interface) of individual servers, which are the primary functions of this layer. The API is a set of protocols and tools that enable communication between the application layer and the servers, allowing the system to access and utilize the resources and services provided by the servers. The application drivers layer is important for enabling communication and connectivity between the various components of the system.

3.4.2 Board Support Package Section

a) **Middleware:** The middleware layer is a critical part of the system's firmware, as it serves as a bridge between the application layer and the hardware. This layer contains the middleware drivers for the designed board and is responsible for making the overall firmware portable. The application layer does not need to have low-level information about the system, and the middleware layer enables communication between the application layer and the low-level system and the Arduino library. This layer is important for enabling the system to function properly and access the resources and services it needs.

b) **Embedded layer:** The embedded layer is responsible for connecting the low-level drivers with the hardware peripherals of the design board without the need for cumbersome management of low-level peripherals. This layer, also known as the on-chip driver management layer, enables the system to access and utilize the hardware resources of the design board without the need to directly manage the low-level peripherals. The embedded layer is important for enabling the system to function properly and access the resources it needs to perform its various tasks.

c) **On-board IC drivers:** The on-board IC drivers' layer is the lowest level of drivers in the system, operating at the chip level. This layer is responsible for isolating the overall applications from hardware dependencies, making the system portable across different platforms. The on-board IC drivers layer enables the system to access and utilize the resources of the on-board ICs without the need to directly manage the low-level

hardware dependencies. This layer is critical for ensuring the smooth operation of the system and its ability to function properly on different platforms.

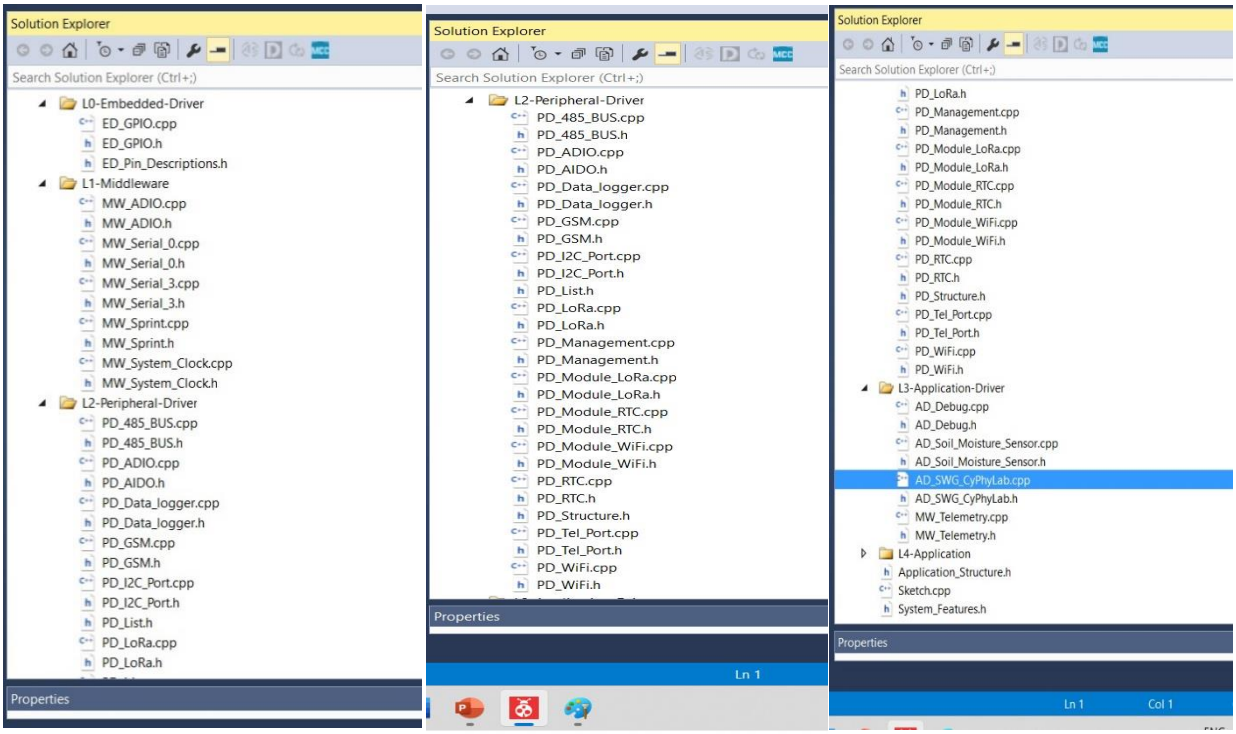

Fig 3.4: Modular API

Here we can see the code base infrastructure to build the application layer and drivers, Middleware, embedded layers and On-Board IC drivers

4 EVALUATION

4.1 SWG 1

The soil moisture data portal is a system that provides farmers with information about the moisture levels in the soil in their fields. This information is presented in the form of a curve, which is displayed in Figure 4.2. The curve is generated based on data collected by sensors that have been deployed in the field, and it shows the moisture levels over a specific period of time.

By analyzing the curve, farmers can determine the appropriate irrigation level for their fields. This allows them to take appropriate action to ensure that the soil is being irrigated at the correct level, which can help to optimize crop growth and productivity. The soil moisture data portal is designed to provide farmers with timely irrigation information, which can help them to make informed decisions about their irrigation practices.

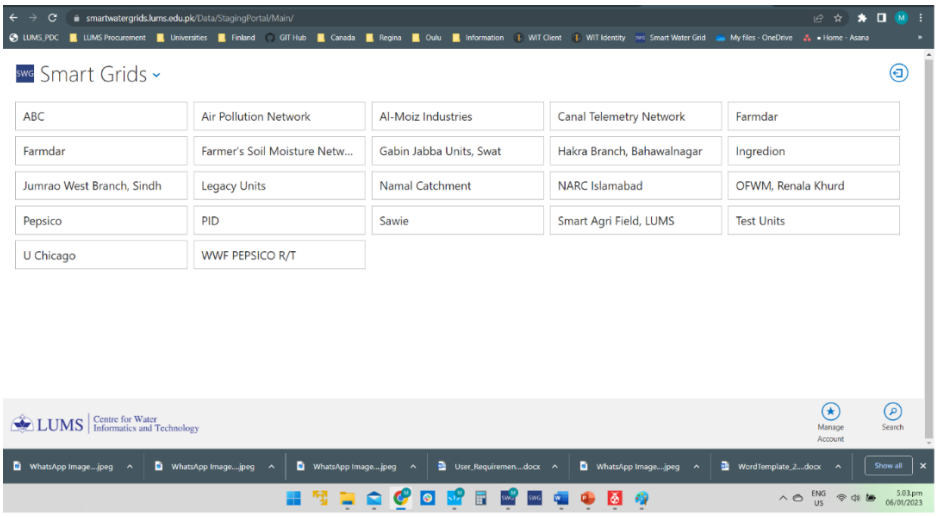

Fig: 4.1: front end view of the portal

Fig 4.2: Moisture data graph from the portal

4.2 SWG 2

Figure 4.3 presents a revised version of a visualization portal, which is a tool used to display data collected by sensors. This updated version has the ability to display data from multiple sensors, as well as multiple readings from each sensor, whereas the previous version could only display data from a single sensor. This enhanced capability allows farmers to more easily and effectively manage their irrigation systems by providing a more comprehensive overview of sensor data. Overall, the improved functionality of the updated visualization portal makes it a more convenient and useful tool for farmers.

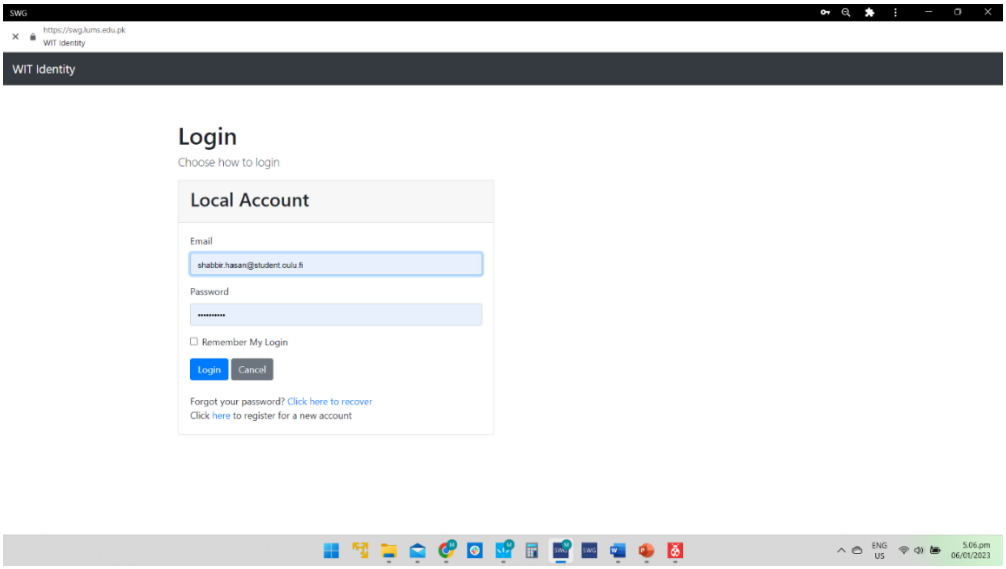

Fig 4.3: User login view

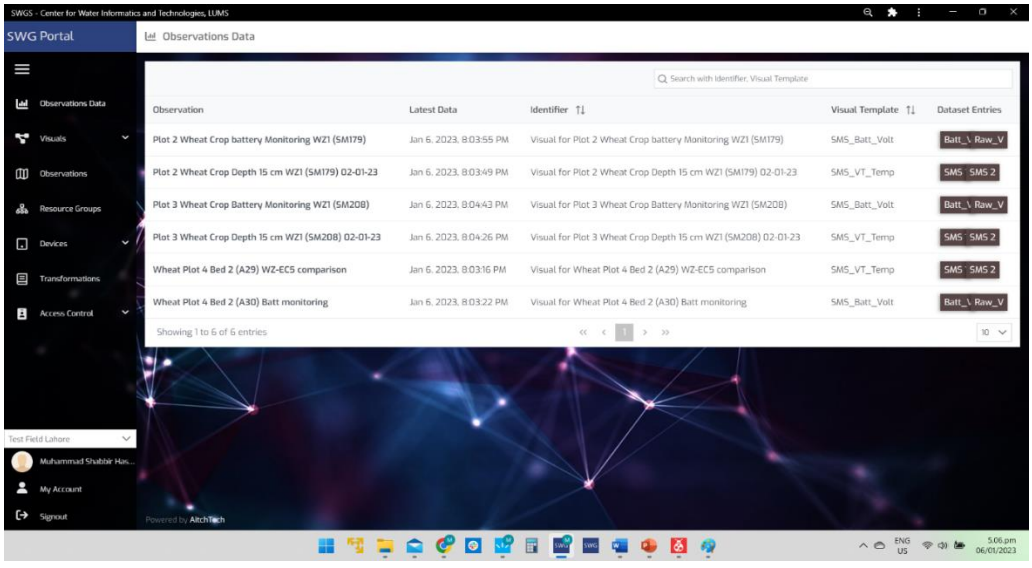

Fig 4.4: Node list connected with Gateway option from the portal

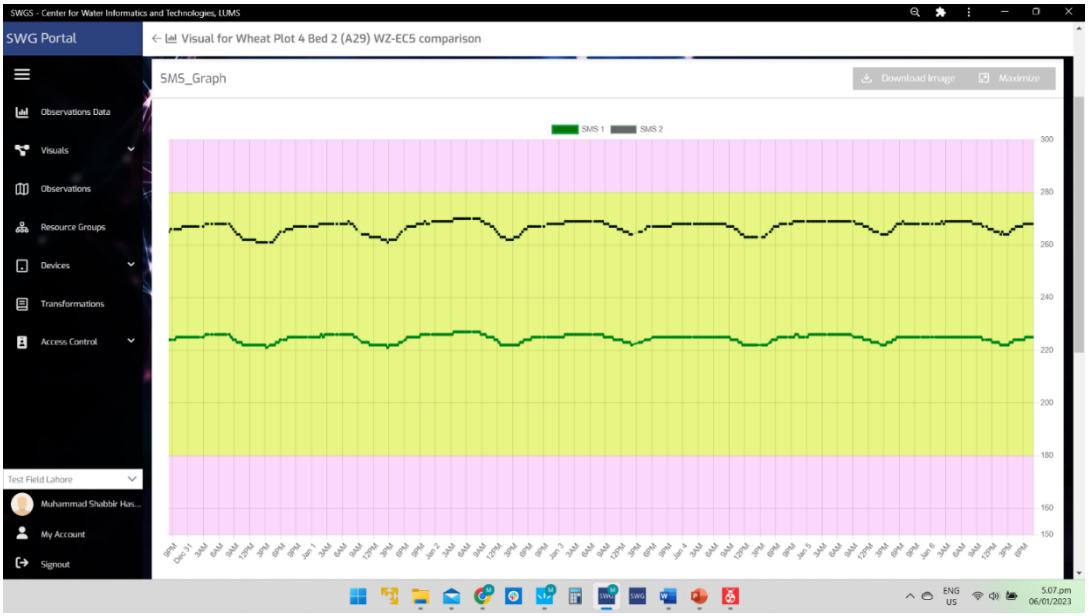

Fig 4.5: Moisture graph of single device at new portal

4.3 Schematic Development

4.3.1 Gateway PCB

The gateway PCB, or printed circuit board, is a device that is designed to support both old and new devices that communicate wirelessly. It is installed in a field and is responsible for validating data packets from nodes, identifying the server, and sending the data to both old and new portals. In addition, the gateway PCB is designed to handle any problems that may have existed in the previous system.

The gateway PCB has several different components, including a power schematic, control schematic, GSM schematic, LoRA schematic, and connection schematic. The power schematic is responsible for ensuring that the device has a reliable power source and can function correctly. The control schematic is responsible for controlling the various components of the device and ensuring that they work together smoothly. The GSM schematic is responsible for managing communication over a cellular network, while the LoRA schematic is responsible for managing communication over a long-range wireless network. The connection schematic is responsible for ensuring that the device can connect to other devices and systems as needed.

The gateway PCB also has a front and back view, which provide detailed information about the layout and design of the device. The front view typically shows the various components and their locations, while the back view provides a more detailed look at the PCB board and how it is constructed.

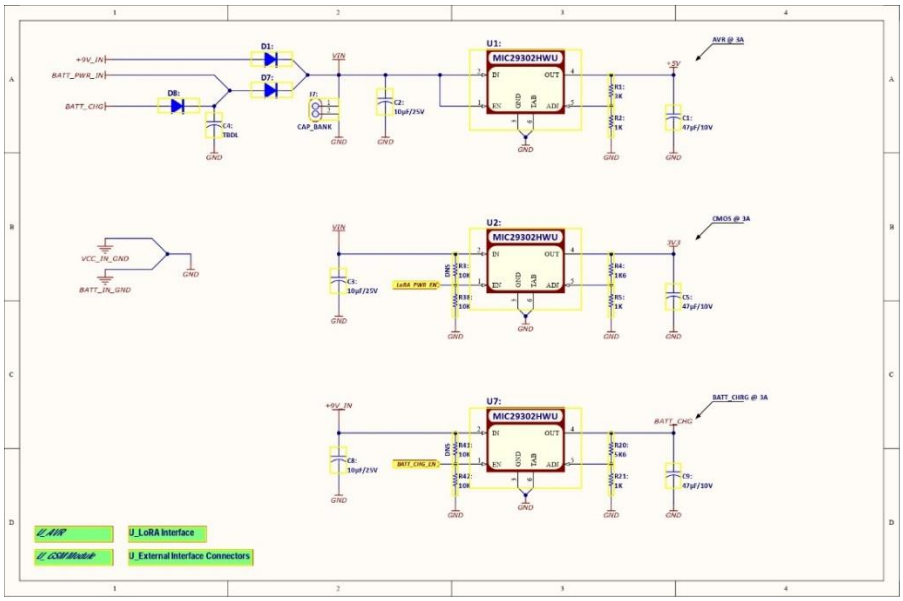

Fig 4.6: Power schematic

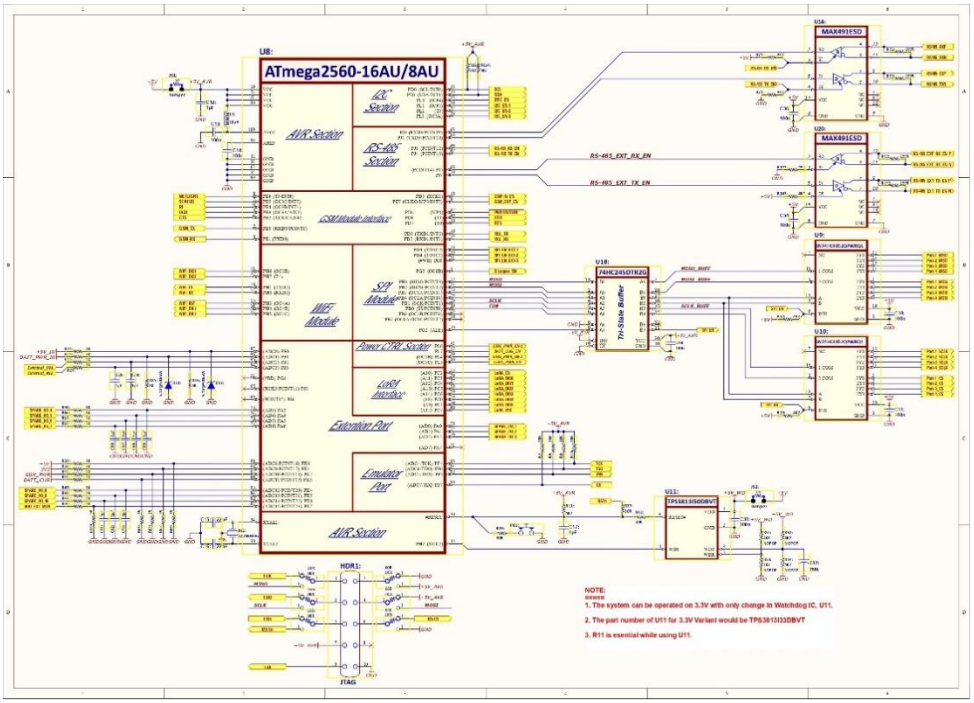

Fig 4.7: Control Schematic

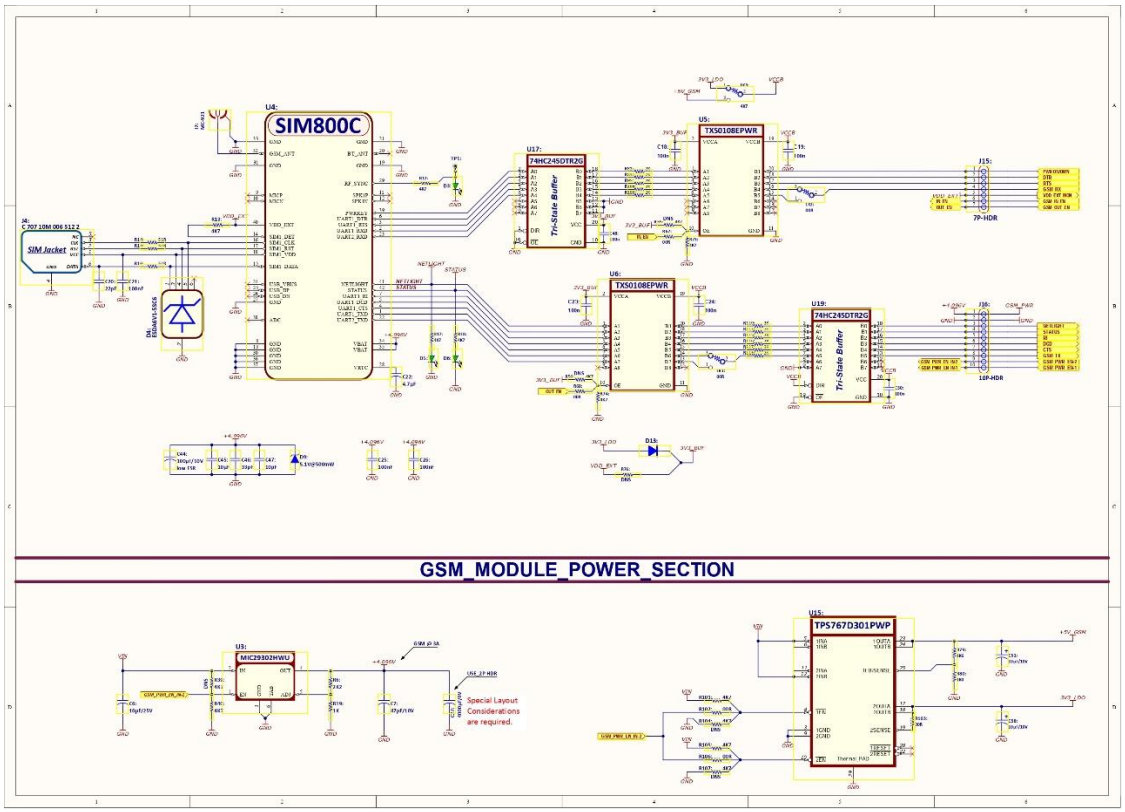

Fig 4.8: GSM schematic

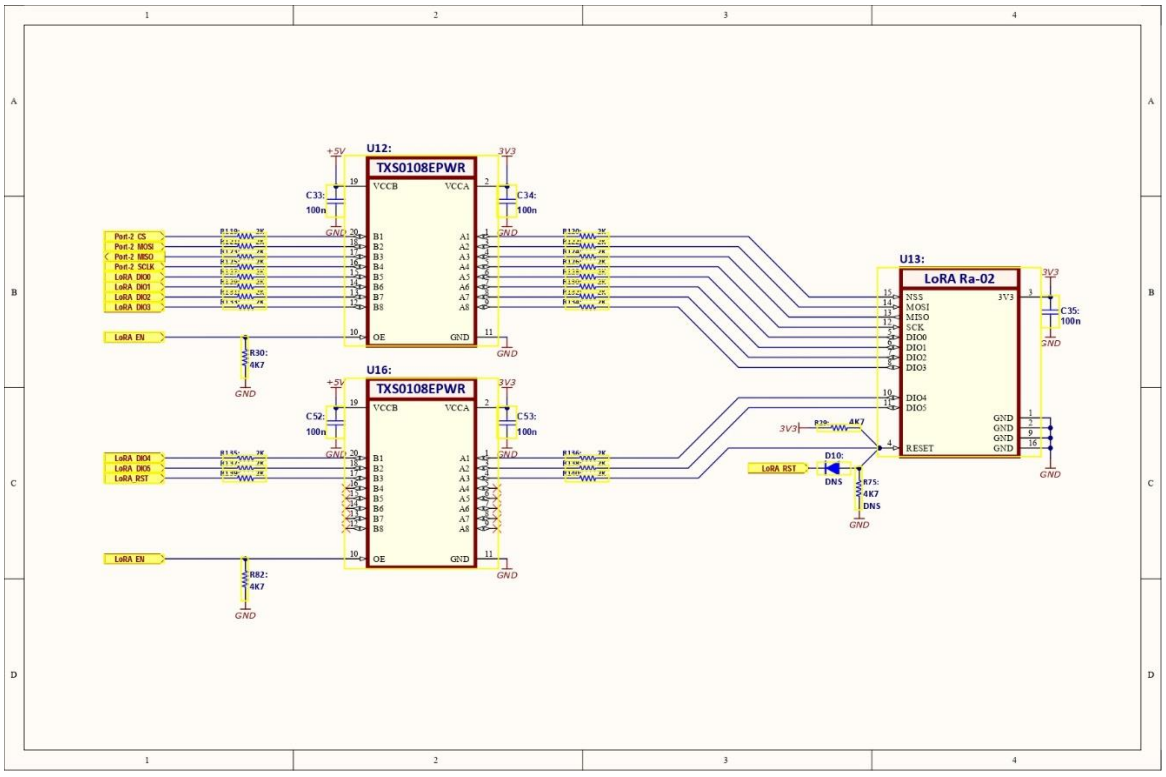

Fig 4.9: LoRA schematic

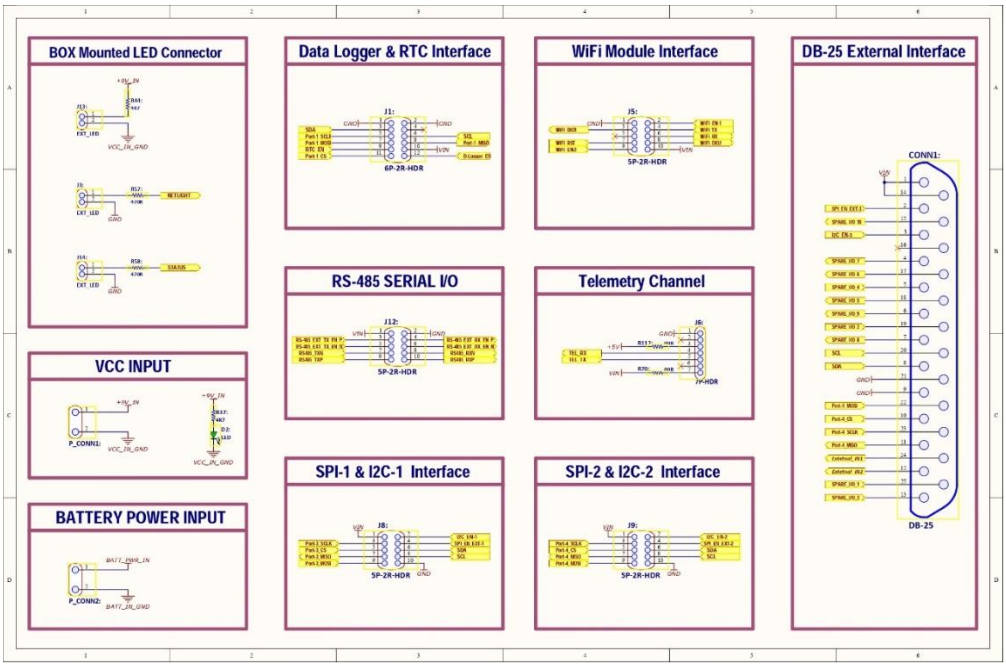

Fig4.10: Connection Schematic

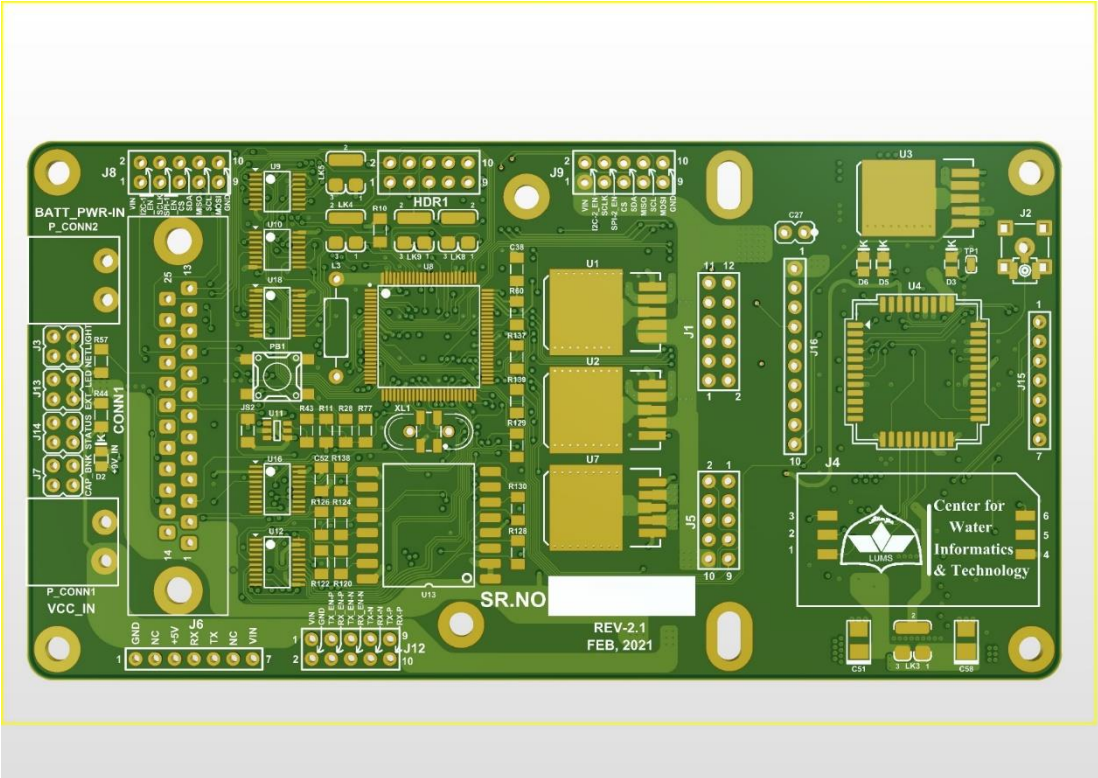

Fig 4.11: PCB front view

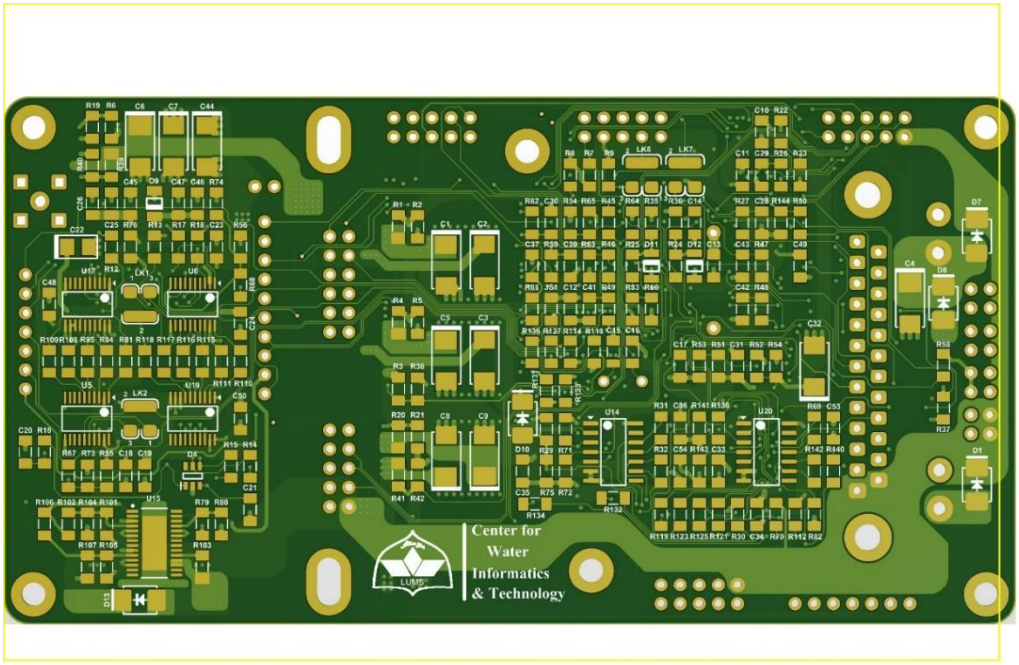

Fig4.12: PCB Back view

4.3.2 Updated Node PCB

The PCB showed in figure is the final design of a PCB that is designed to support multiple sensors. This means that it can be used with a variety of different sensors that may have different interface requirements, such as serial, SPI, I2C, analog, or digital interfaces. The PCB is designed to be attached to a node PCB, which is a separate device that is used to connect sensors and other devices to a network or system.

The ability to support multiple types of sensors with multiple interface options is a key feature of this PCB, as it allows it to be used in a wide range of applications and environments. By providing support for different interface types, the PCB is able to communicate with a wide range of sensors and devices, making it a versatile and flexible component that can be used in a variety of different systems and applications. Overall, the design of this PCB is intended to provide a reliable and flexible platform for attaching and communicating with multiple sensors in a variety of different settings.

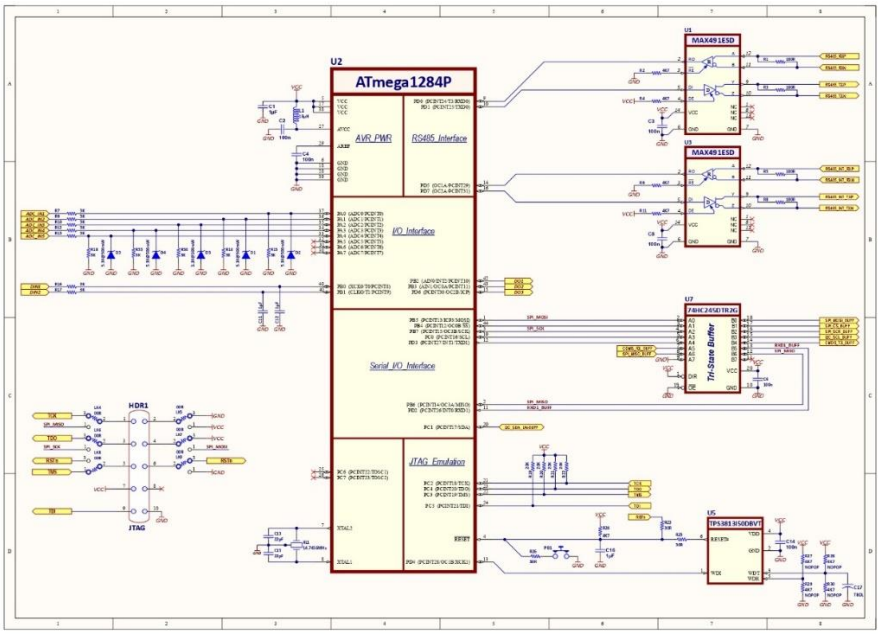

Fig 4.13: New node PCB schematic

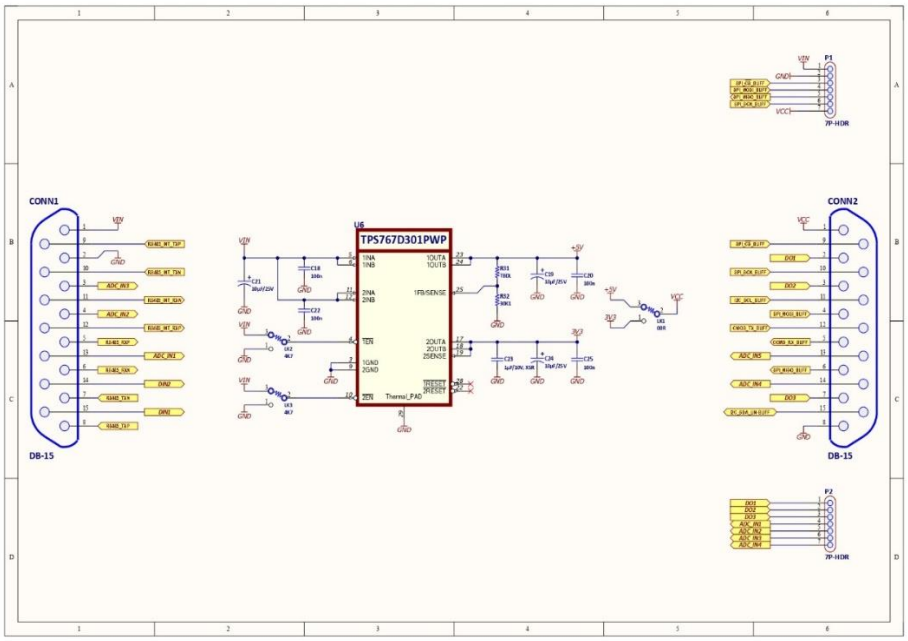

Fig 4.14: New Node connection schematic

Fig 4.15: Actual picture of the new device

The described system consists of a node and a sensor. The node is a printed circuit board (PCB) that is mounted on top of the sensor, which is installed in the field to detect moisture levels in the soil. The node PCB is equipped with a LoRa module, which is used to transmit data collected by the sensor to a gateway. The sensor is located at the bottom of the node setup, and is responsible for collecting moisture data from the soil. This data is then transmitted to the gateway via the LoRa module on the node PCB, allowing farmers to monitor soil moisture levels remotely. Overall, the described system allows for efficient and convenient monitoring of soil moisture levels in agricultural fields.

5 RASPBERRY PI PROTOTYPE

The SWG-2 portal is currently lacking temperature and humidity data, but these features are planned to be included in an upcoming version. In anticipation of this update, a prototype has been developed to incorporate temperature, humidity, and moisture data.

5.1 Set up

The prototype was built using a Raspberry Pi version 2 in this project. The Raspberry Pi was used to run a Python script that utilized the adafruit.DHT library to read data from the sensor pin. The adafruit.DHT library is a Python library specifically designed for reading data from DHT temperature and humidity sensors. It provides a simple and easy-to-use interface for accessing the sensor data. The prototype used this library to read the temperature and humidity data from the sensor and store it for future use.

The library provides functions for initializing a connection to the sensor and reading the temperature and humidity data. It also includes support for a variety of different DHT sensor models and can automatically detect the sensor type being used.

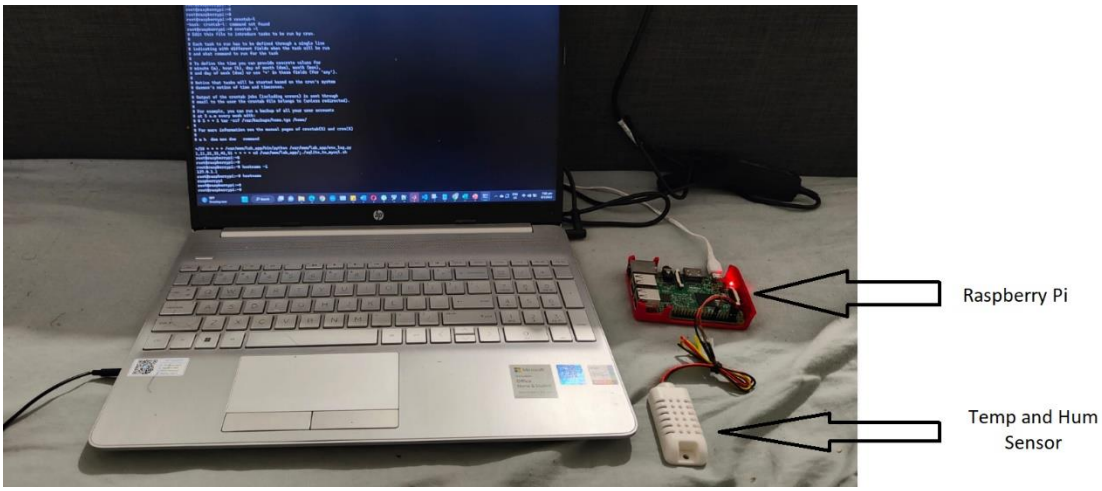

Fig 6.1: Raspberry Pi Set up

5.2 Reading the sensor data

We first initialized a connection to the sensor by specifying the sensor type and the pin number on the Raspberry Pi to which the sensor is connected. Once the connection was established, we used the library's functions to read the temperature and humidity data from the sensor. We used the "read_retry" function to read the current temperature and humidity from the sensor. The library handled all of the low-level details of communicating with the sensor and provides a simple interface for accessing the data.

```
humidity, temperature = Adafruit_DHT.read_retry(Adafruit_DHT.AM2302, 23)
# If you don't have a sensor but still wish to run this program, comment out all the
# sensor related lines, and uncomment the following lines (these will produce random
# numbers for the temperature and humidity variables):
# import random
# humidity = random.randint(1,100)
# temperature = random.randint(10,30)
if humidity is not None and temperature is not None:
        log_values("1", temperature, humidity)
else:
        log_values("1", -999, -999)
```

Fig 6.2: Using adafruit.DHT read retry function

5.3 Data storing

In the prototype, the Python code used the sqlite library to export the temperature and humidity data to a sqlite database installed on the Raspberry Pi. SQLite is a popular embedded database management system that is well-suited for use in small-scale projects like this one. The prototype used the sqlite library to connect to the database and create a table to store the sensor data. The data was then sensed at regular intervals using the adafruit.DHT library, and the resulting temperature and humidity readings were sent to the database using SQL INSERT statements. This allowed the prototype to continuously collect and store the sensor data over time.

```
import sqlite3
import sys
import Adafruit_DHT

def log_values(sensor_id, temp, hum):
        conn=sqlite3.connect('/var/www/lab_app/lab_app.db')  #It is important to provide an
                                                             #absolute path to the database
                                                             #file, otherwise Cron won't be
                                                             #able to find it!
        # For the time-related code (record timestamps and time-date calculations) to work
        # correctly, it is important to ensure that your Raspberry Pi is set to UTC.
        # This is done by default!
        # In general, servers are assumed to be in UTC.
        curs=conn.cursor()
        curs.execute("""INSERT INTO temperatures values(datetime(CURRENT_TIMESTAMP, 'localtime'),
         (?), (?))""", (sensor_id,temp))
        curs.execute("""INSERT INTO humidities values(datetime(CURRENT_TIMESTAMP, 'localtime'),
         (?), (?))""", (sensor_id,hum))
        conn.commit()
        conn.close()
```

Fig 6.3: Inserting temperature and Humidity data to SQLite db

5.4 Data visualization

In the prototype, Grafana was installed on the Raspberry Pi and configured to use the SQLite database as a data source. In this case, the Grafana installation on the Raspberry Pi was used to visualize the temperature and humidity data stored in the SQLite database. The prototype used the Grafana installation to create visualizations of the sensor data, such as line graphs showing the trends in temperature and humidity over time.

To make the Grafana portal accessible over the internet, the Raspberry Pi was configured to act as a server and the IP address of the Raspberry Pi was port forwarded from the router. This

allowed the Grafana portal to be accessed from any device with an internet connection by using the public IP address of the router.

Fig 6.4: Grafana visualization portal

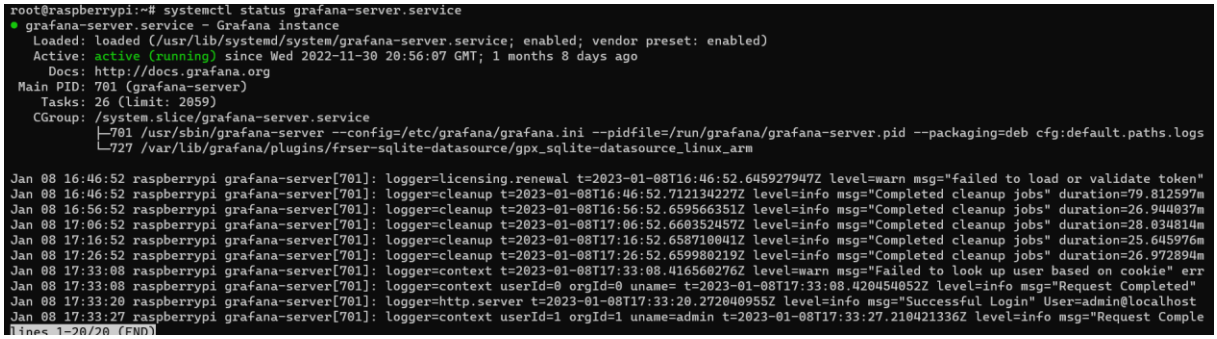

Fig 6.5 : Grafana Service running inside Raspberry Pi

6 MACHINE LEARNING PROTOTYPE

In this study, we have designed an ANN (Artificial Neural Network) model to decide whether irrigation is required based on temperature, humidity, and Lux (Intensity data) data. These variables are commonly used to predict the water requirements of plants, and they can be influenced by a variety of factors, such as the type of crop, the location, and the weather. By using an ANN model, we aim to improve the accuracy and efficiency of irrigation decision-making, and to help farmers optimize the use of water resources.

The ANN model was trained using a dataset of temperature, humidity, and light data collected from an open database, along with labels indicating whether irrigation was required or not. The model was then tested on a separate dataset to evaluate its performance. In the following sections, we will describe the design of the ANN model, the training and testing process, and the results of the study.

This model will be used to analyze and make predictions on the data set developed for the SWG-2 portal. The ultimate aim of this model is to utilize the insights and predictions generated by the model to improve the functionality and efficienc6y of the SWG-2 portal.

6.1 Data Pre-processing

Data pre-processing is a crucial step in the development of an Artificial Neural Network (ANN) model. It involves cleaning, transforming, and formatting the data in a way that makes it suitable for use by the model.

In the case of an ANN model with dependent and independent variables, data pre-processing involves a few key tasks. First, we identified the dependent variable (also known as the target variable), which is the variable that we wanted to predict using the model. Then we identified the independent variables (also known as the features), which are the variables that was used to make the prediction.

Next, we checked the data for any missing values, inconsistencies, or errors, and fix or remove these as necessary. This can help to improve the accuracy and reliability of the model.

Once the data is clean and consistent, we applied some transformations or scaling to the data.

6.1.1 Labelling and One Hot encoding

Labelling the dependent variable in an Artificial Neural Network (ANN) model involved assigning a unique label or value to each possible class or category in the data. This was done using a process called one-hot encoding, which involved creating a new binary column for each class and setting the value to 1 for the corresponding row and 0 for all other rows.

One-hot encoding the longitude and latitude columns in ANN model involved a similar process, where a new binary column is created for each unique longitude and latitude value.

Fig 5.1: Labelling

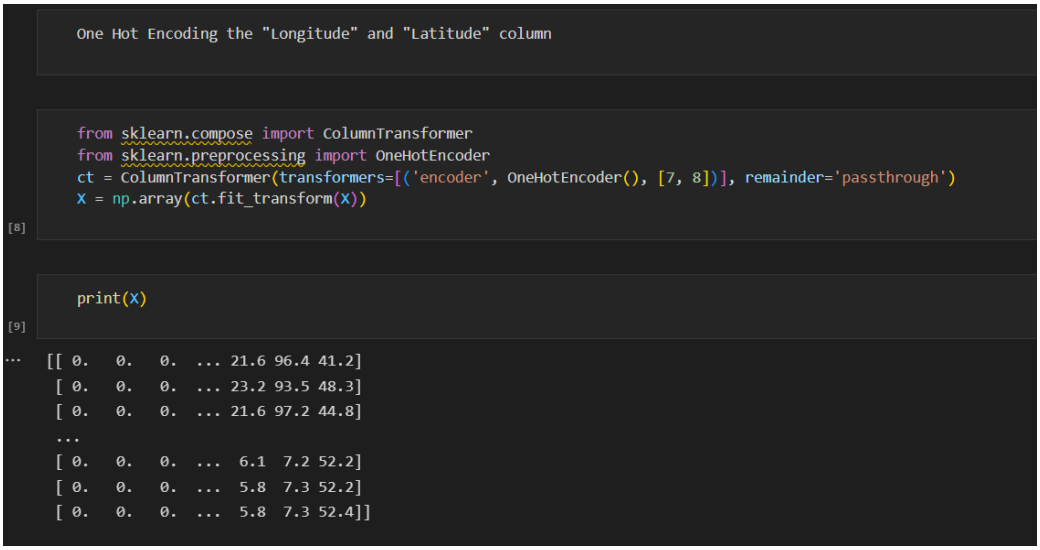

Fig 5.2: One Hot encoding

6.1.2 Data Splitting and Feature scaling

In this study, we have split the dataset into a training set and a test set in order to fit and evaluate the performance of our machine learning model. The training set was used to fit the model, while the test set was used to evaluate the model on unseen data. This helped us to ensure that the model was able to generalize well to new data and to avoid overfitting.

In addition, we have applied feature scaling to the data in order to transform the values of the features to a common scale. This was done to improve the performance of the model, as the features had different scales and ranges. We used the standardization method to scale the data, which involved subtracting the mean and dividing by the standard deviation for each feature. This helped to ensure that all the features were on the same scale and had similar variances.

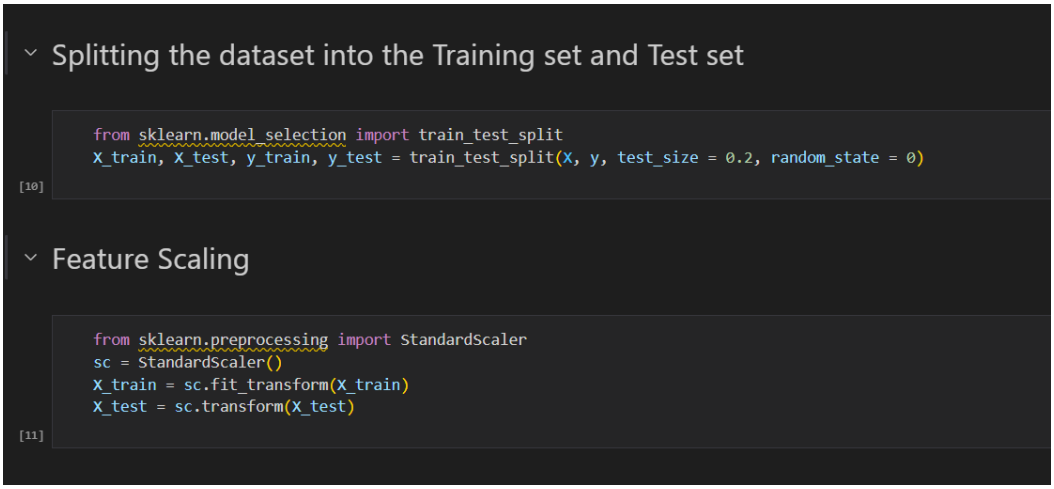

Fig 5.3: Data Splitting and Feature Scaling

6.2 ANN Model

6.2.1 Building the ANN model

This code uses the TensorFlow library to initialize an Artificial Neural Network (ANN) model in Python. The ANN is initialized as a Sequential model, which means that the layers of the model will be added in a linear fashion, with the output of one layer becoming the input of the next.

The code then adds three layers to the ANN: an input layer, two hidden layers, and an output layer. The input layer and the hidden layers are implemented using the Dense layer type, which represents a fully connected layer in the ANN. The output layer is also implemented using a Dense layer, but it uses a different activation function (sigmoid) to produce a binary output.

The number of units in each layer (6 for the input and hidden layers, and 1 for the output layer) determines the number of neurons in the layer, and the activation functions (relu for the hidden layers and sigmoid for the output layer) determine how the outputs of the neurons are transformed and passed to the next layer.

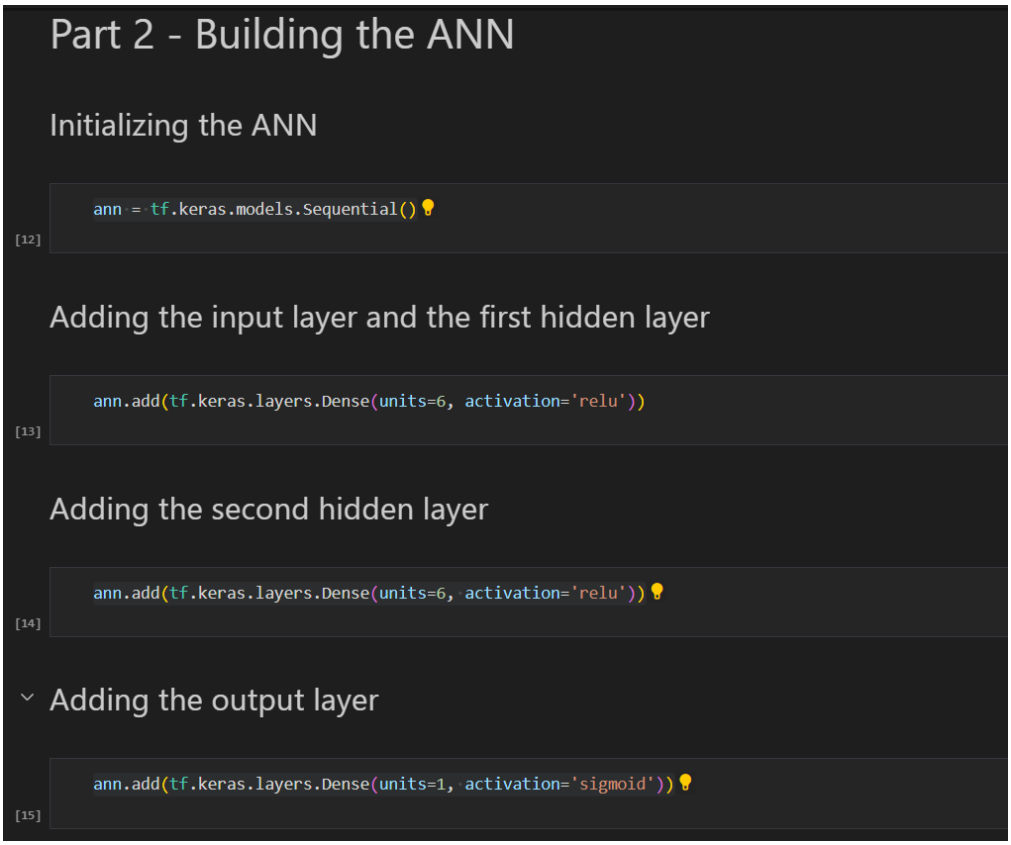

Fig 5.4: ANN building

6.2.2 Training the ANN model

This code is used to compile and train an Artificial Neural Network (ANN) model in Python using the TensorFlow library. The compile method is used to configure the training process by specifying the optimizer, the loss function, and the evaluation metrics.

The optimizer determines the algorithm that will be used to adjust the weights and biases of the model in order to minimize the loss function. The loss function measures the difference between the predicted outputs of the model and the true labels, and it is used to guide the optimization process. The evaluation metrics are used to assess the performance of the model during training and testing.

In this case, the code specifies the adam optimizer, the binary_crossentropy loss function (for binary classification), and the accuracy metric.

The fit method is then used to train the ANN on the training set. The batch_size parameter determines the number of samples that will be used in each iteration of training, and the epochs parameter determines the number of times the model will be trained on the entire training set. The fit method returns a history object that contains information about the training process, including the values of the loss function and the evaluation metrics at each epoch.

Part 3 - Training the ANN

Compiling the ANN

```
[16] ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

```
[17] ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

Fig 5.5: Training the Model

```
Epoch 1/100
1415/1415 [==============================] - 4s 2ms/step - loss: 0.3319 - accuracy: 0.8745
Epoch 2/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1809 - accuracy: 0.9608
Epoch 3/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1517 - accuracy: 0.9665
Epoch 4/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1333 - accuracy: 0.9690
Epoch 5/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1190 - accuracy: 0.9701
Epoch 6/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1083 - accuracy: 0.9710
Epoch 7/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.1004 - accuracy: 0.9709
Epoch 8/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0957 - accuracy: 0.9708
Epoch 9/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0921 - accuracy: 0.9713
Epoch 10/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0895 - accuracy: 0.9712
Epoch 11/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0873 - accuracy: 0.9724
Epoch 12/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0865 - accuracy: 0.9720
Epoch 13/100
...
Epoch 99/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0606 - accuracy: 0.9774
Epoch 100/100
1415/1415 [==============================] - 3s 2ms/step - loss: 0.0606 - accuracy: 0.9773
```

Fig 5.6: 97.73% accuracy in 100th epoch

6.2.3 Making Prediction and Evaluating the Model

This code is used to evaluate the performance of an Artificial Neural Network (ANN) model on a test set in Python. The predict method is used to generate predictions for the test set, and the resulting predictions are thresholded to create a binary output (where values greater than 0.5 are set to 1 and values less than or equal to 0.5 are set to 0).

The true labels for the test set (y_test) and the predicted labels (y_pred) are then printed side by side, using the np.concatenate function to combine the arrays into a single 2D array. This can be useful for comparing the predicted labels to the true labels and identifying any discrepancies.

The code then uses the confusion_matrix function from the sklearn.metrics module to calculate a confusion matrix for the test set. A confusion matrix is a table that shows the number of true positive, true negative, false positive, and false negative predictions made by the model. The diagonal elements of the confusion matrix (true positive and true negative) correspond to correct predictions, while the off-diagonal elements (false positive and false negative) correspond to incorrect predictions.

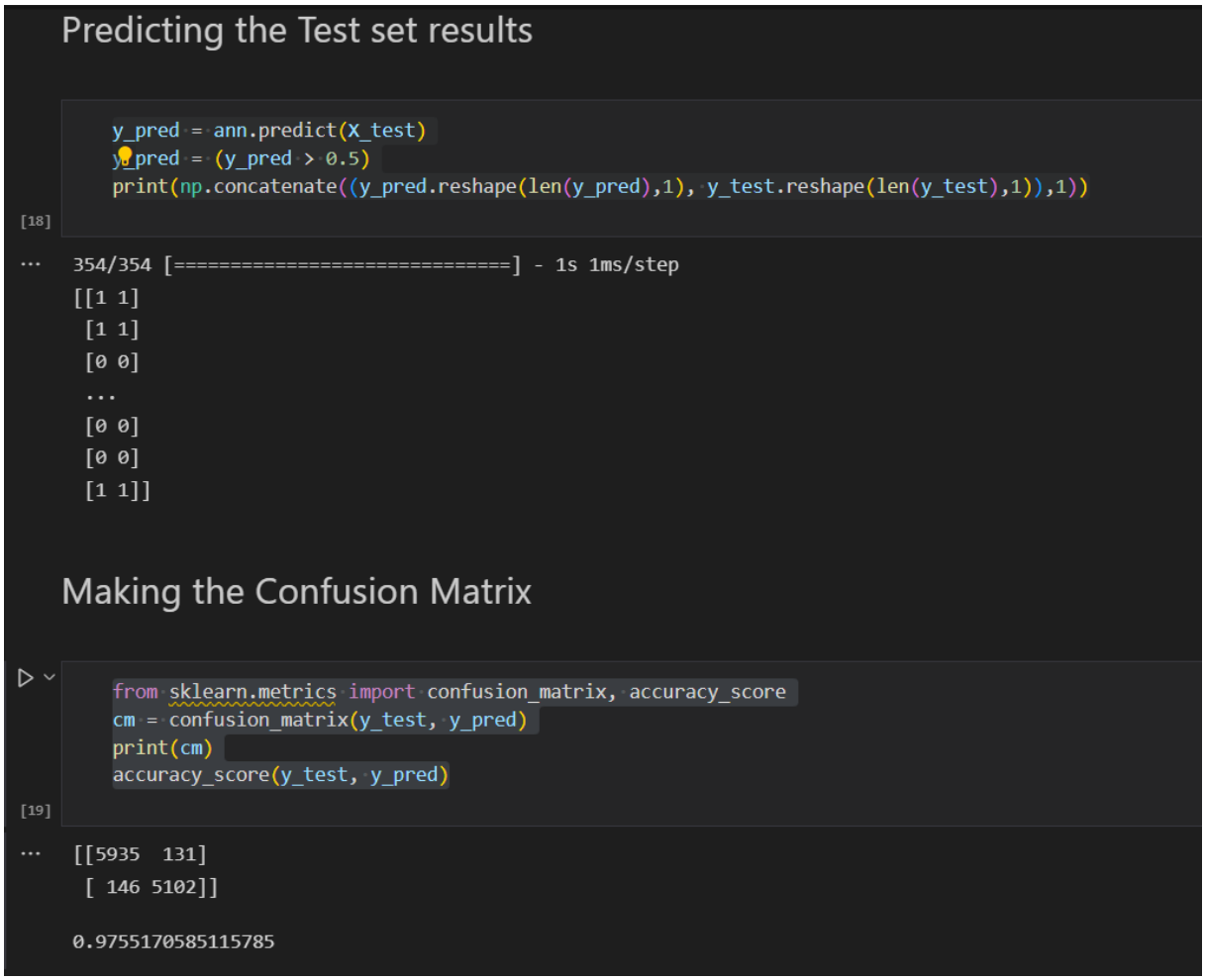

Fig 5.7: Predicting and Confusion Matrix

The diagonal elements of the matrix (5935 and 5102 in this case) correspond to correct predictions, while the off-diagonal elements (131 and 146 in this case) correspond to incorrect predictions. The values of the diagonal elements represent the number of true positive and true negative predictions, while the values of the off-diagonal elements represent the number of false positive and false negative predictions.

In this case, the model has made 5935 true positive predictions (i.e., it correctly predicted that Irrigation is needed), 5102 true negative predictions (i.e., it correctly predicted that the irrigation is not needed), 131 false positive predictions (i.e., it incorrectly predicted that irrigation is needed when it was not needed), and 146 false negative predictions (i.e., it incorrectly predicted that the irrigation is not needed when it was actually needed).

7 DISCUSSION

7.1 Future Work

The future work for the system includes the following tasks:

a. **Incorporation of a new communication protocol for unlimited sensors at a single node:** The current system is limited in its ability to support multiple sensors at a single node, and the new communication protocol will address this limitation and enable the use of an unlimited number of sensors.

b. **Incorporation of remaining firmware functionality:** There are still some firmware functions that need to be implemented in the system, and these will be added in future updates.

c. **Active improvement of the node for complete integration:** The node will be actively improved to ensure complete integration with the system and all of its components.

d. **Algorithm development for multiple data fusion**: In order to enable smart prediction capabilities, algorithms will be developed to enable the fusion of multiple data sources. These algorithms will be used to analyse and interpret the data in order to make accurate predictions.

7.2 Limitation

There are currently several limitations to the system:

a. **Solar power implementation at the node side has not yet been implemented:** The system does not currently have the ability to utilize solar power at the node side, which limits its flexibility and potential applications.

b. **Multiple sensor type integration has not been implemented:** The system is currently limited in its ability to integrate multiple types of sensors, which limits its capabilities and the data it is able to collect.

c. **Machine learning algorithms have not been implemented at the cloud:** The system does not currently have the ability to utilize machine learning algorithms at the cloud level, which limits its ability to analyze and interpret data in real-time. This limitation may impact the accuracy and effectiveness of the system's predictions and decisions.

There are several potential improvements that could be made to the system to address its current limitations:

a. **Incorporation of solar power at the node side:** By implementing solar power at the node side, the system would be able to operate more flexibly and in a wider range of locations and environments.

b. **Integration of multiple sensor types:** By adding the ability to integrate multiple sensor types, the system would be able to collect a wider range of data, increasing its capabilities and the insights it is able to provide.

c. **Implementation of machine learning algorithms at the cloud:** By adding the ability to utilize machine learning algorithms at the cloud level, the system would be able to analyze and interpret data in real-time, improving the accuracy and effectiveness of its predictions and decisions.

Overall, these improvements would significantly enhance the capabilities and performance of the system, making it more useful and valuable to users.