Plant Environment Monitoring System

IOE Mini Project

Submitted in complete fulfillment of the requirements of the degree for the

IOE (Mini Project) Lab

Submitted by

Mr. Ninad Rao Roll No. **57**
Mr. V Krishnasubramaniam Roll No. **67**
Ms. Shalaka Waghamale Roll No. **68**

Under the guidance of

Prof. Dimple Bohra

Department of Information Technology

Vivekanand Education Society's Institute of Technology

University of Mumbai

2022-2023

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY (VESIT)

Chembur, Mumbai 400074

CERTIFICATE

This is to certify that **Mr. Ninad Rao (57)**, **Mr. V Krishnasubramaniam (67)** and **Ms. Shalaka Waghamale (68)** have satisfactorily carried out the project work, under the head - Internet of Everything Lab at Semester VII of BE in Information Technology as prescribed by the Mumbai University.

Prof. Dimple Bohra
Subject Teacher

External Examiner

Dr.(Mrs.) Shalu Chopra
H.O.D

Dr.(Mrs.) J.M.Nair
Principal

Date: 21/10/2022

Place: VESIT, Chembur

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

- - - - - - - - - - -
(Signature)

Mr
(Ninad Rao, 57)
B.E. INFT

- - - - - - - - - - -
(Signature)

Mr
(V Krishnasubramaniam, 67)
B.E. INFT

- - - - - - - - - - -
(Signature)

Ms
(Shalaka Waghamale, 68)
B.E. INFT

Contents

List of Figures

List of Tables

Abstract

The Internet of Things (IoT) is the network of physical objects. It simply means to monitor a physical device or machine or it is inter-networking of physical devices which is embedded with electronics, sensors, software and network connectivity to enable it to achieve greater value and services by exchanging data. IoT can be used in our daily life to monitor the growing plants and their environments so that we give them the best chance to grow by showing the user the optimum ways to give the plant the required conditions. We propose a system that will capture all the details about the environment of a plant, including the light intensity, soil moisture and temperature by means of different sensors, and notify the user to water the plant or move the plant to a place where there is more sunlight. We also perform multivariate time series analysis for predicting the time at which we should water the plant.

Keywords - ***IoT, plant environment monitoring, temperature, humidity, soil moisture, light intensity, time series prediction***

Chapter 1

Introduction

1.1 Introduction

As we can see in today's world only some devices like PC's and mobiles are connected to the internet. Now-a-days world is fully overtaken by the internet and internet of things. The Internet is used for the basic needs of all human beings. The Internet of Things (IOT) is the network of physical objects. It simply means to monitor a physical device or machine or it is inter-networking of physical devices which is embedded with electronics, sensors, software and network connectivity to enable it to achieve greater value and services by exchanging data with the manufacturer. IOT permits objects to be sensed or controlled remotely across the network infrastructure. The result improves accuracy, economic benefits, efficiency and reduces intervention of humans.

The temperature as well as humidity and light are measured by means of sensors and depend up on the outcome further processing can be performed. We propose a system that will capture all the details about the environment of a plant, including the light intensity, soil moisture and temperature by means of different sensors. We are trying to demonstrate IOT in a Plant Environment Monitoring system. NodeMCU is used as a microcontroller to implement the control unit. The set up uses the light intensity, temperature sensor, moisture sensor and humidity sensor which measure the approximate temperature, moisture and humidity in the soil. This value enables the system to use appropriate quantities of water which avoids over/under watering.

1.2 Literature Survey

The research paper [1] has proposed a system that uses NodeMCU as a micro-controller. NodeMCU comes with the inbuilt ESP8266 WiFi module which connects our system to the Flutter app using WiFi. The program which controls the functioning of the whole system is fed into the micro-controller using Arduino IDE which is an environment which integrates code with the hardware. Soil moisture sensor continuously detects the level of moisture in the soil and displays it on the Flutter app. Real-time values from the DHT11 and light intensity sensor are also displayed on the app. Excessive heat from the sun can be harmful to plants to prevent them from dying we introduced notifications to move the plant to a shaded area. Temperatures more than 30 0C can cause shrivelling of plants. When temperature increases this limit the motor rotates and causes the shade to move automatically. The user is notified about each and every

step through the notification feature of the Flutter app. Hence, this system monitors and controls the plants requirements remotely.

They suggested that the performance of the system could be further improved in terms of the operating speed, memory capacity, and instruction cycle period of the micro-controller by using other high end controllers. The number of channels can be increased to interface more number of sensors which is possible by using advanced versions of controllers. The system can be modified with the use of a data logger and a graphical LCD panel showing the measured sensor data over a period of time.A speaking voice alarm could be used.The device can be made to perform better by providing the power supply with the help of renewable source. Time bound administration of fertilizers, insecticides and pesticides could be introduced.

In research paper [2], the focus is to collect and summarize various advanced forecasting models for multivariate time series dataset. They have discussed the inherent forecasting strengths and weaknesses related to these time series modelings. Also, the main section deals with the experience of using such data in econometric analysis. The time series processes we have discussed so far are all stationary processes, but many applied time series, particularly those arising from economic and business areas, are non-stationary. With respect to the class of covariance stationary processes, non-stationary time series can occur in many different ways. The paper has used the auto-regressive integrated moving average models (i.e) ARIMA models. Some useful differencing and variance stabilizing transformations are introduced to connect the stationary and non-stationary time series models.

1.3 Objectives

Our IoT-based plant environment monitoring system has the following objectives:

- Notify the user to move the plant to an area with more sunlight when it does not have enough light conditions
- Notify the user to water the plant when the soil does not have enough moisture content
- Getting the estimated time for watering the plant before the soil moisture decreases below a threshold
- Provide the home plant of the user with appropriate environmental conditions for better growth

1.4 Hardware and Software requirements

Hardware Requirements:

- NodeMCU ESP8266 Board
- Soil Moisture sensor
- DHT11 Sensor

- BH1750 Light intensity sensor

Software Requirements:

- AWS IoT Core
- Flutter
- Flask (Python)

1.5 Wireless Technology Used

- MQTT - for data collection and storing
- HTTP (REST API) - for fetching prediction from analytics application
- Wifi - for connection of NodeMCU board to AWS

1.6 Cost Estimation

The requirement of the system is to be as low in cost as possible, so that whenever a user needs more than one of these systems for their plants, it is easy and cost effective to buy. The NodeMCU board is the cheapest sensor board that provides Wifi connectivity that is enough for AWS connectivity.

Component	Price (in Rs.)
NodeMCU Board	380
Soil Moisture Sensor	90
DHT11 Sensor	140
BH1750 Sensor	190
Total estimated cost	800

Table 1.1: Estimated costs for the system

Chapter 2

IoT System Design

2.1 Purpose and Requirements

Plant plays a vital role in maintaining the ecological cycle and forms the foundation of a food chain pyramid thus to maintain the plant's proper growth and health adequate monitoring is required. Hence the aim at making plant monitoring systems smart is to use automation and Internet of Things (IoT) technology and it is very important for their fast growth. In this busy world, people usually forget to water their plants which leads to bad growth and health of their plants. For ensuring the complete development of plants it is necessary to develop proper surrounding conditions in which plants grow. This highlights various features such as smart decision-making based on soil moisture real-time data.

Plants at home requires as much care as any plant present in a garden or a science lab. If they are not watered regularly or cared for enough, they may wilt and eventually die. Sometimes watering the plant regularly or just moving it to a more sun-lit area can make huge improvements to the plant's environment providing it with a greater chance of flourishing and/or blooming fully. Reminding the user to water the plant at a predicted time when the soil of the plant is about to dry-up will allow the user to plan the watering schedule for that plant if he/she is not available when the soil dries up.

For this purpose, sensors like the Soil Moisture sensor to sense the amount of moisture present in the soil, the DHT11 sensor to sense the temperature and the humidity and the Light Intensity sensor to sense the amount of sunlight are used. This is prepared by using a NodeMCU board. This network helps with obtaining data through the Internet of Things. The whole system consists of nodes which are situated at different parts of the board. The data or the value from the sensors is collected by the NodeMCU interfacing board and uploaded to the cloud i.e. AWS through the NodeMCU board.

2.2 System Design

2.2.1 Process Model Specification

The process specification shows that the sensors are read after fixed intervals and the sensor measurements are stored.

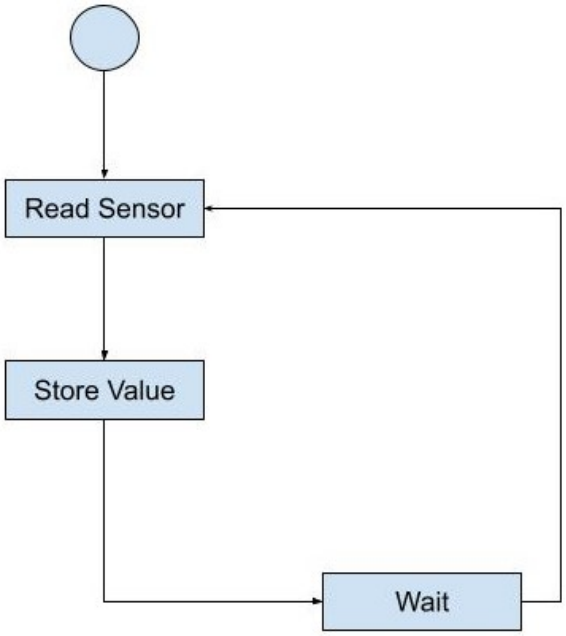

Figure 2.1: Process Model Specification

2.2.2 Domain Model Specification

In this domain model the physical entity is the environment which is being monitored. Hence, there is a virtual entity for the environment. Sensors connected to the single-board mini computer include temperature sensor, humidity sensor, soil moisture sensor and light intensity sensor. Resources are software components which can be either on-device or network-resources.

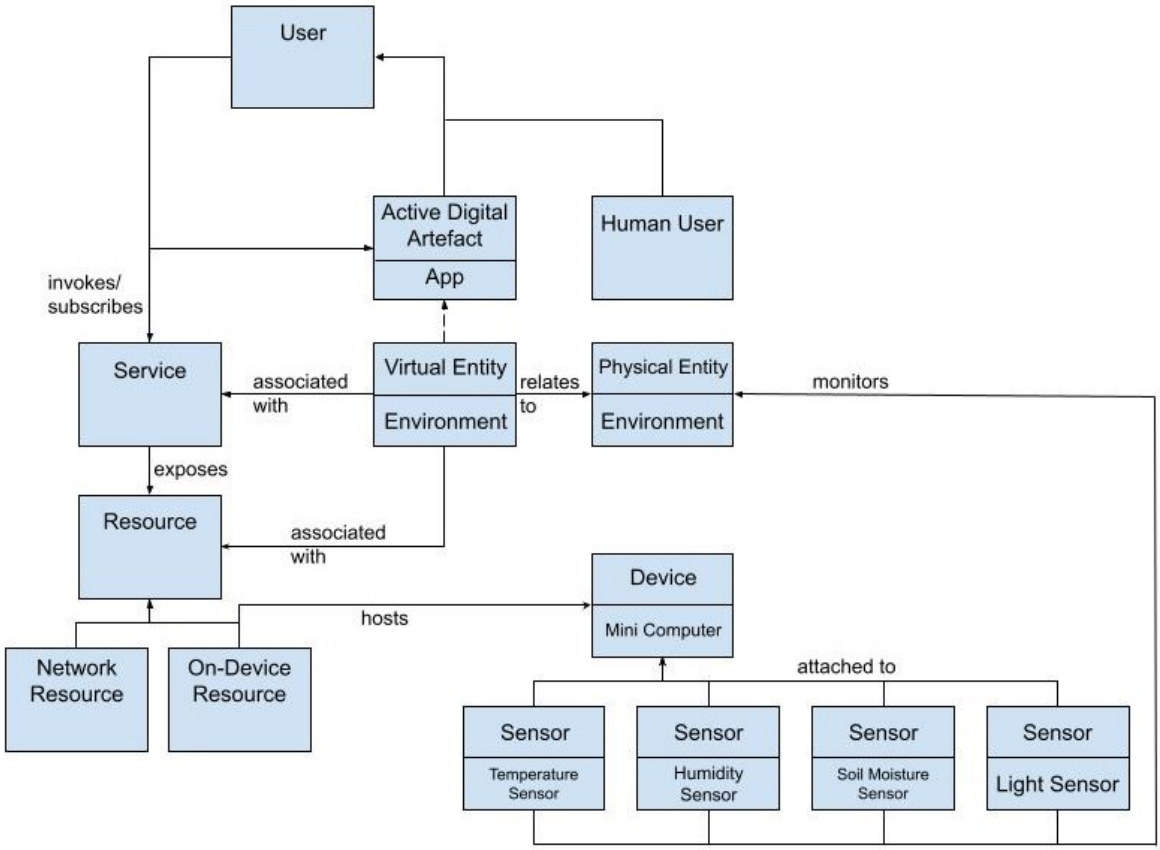

Figure 2.2: Domain Model Specification

2.2.3 Information Model Specification

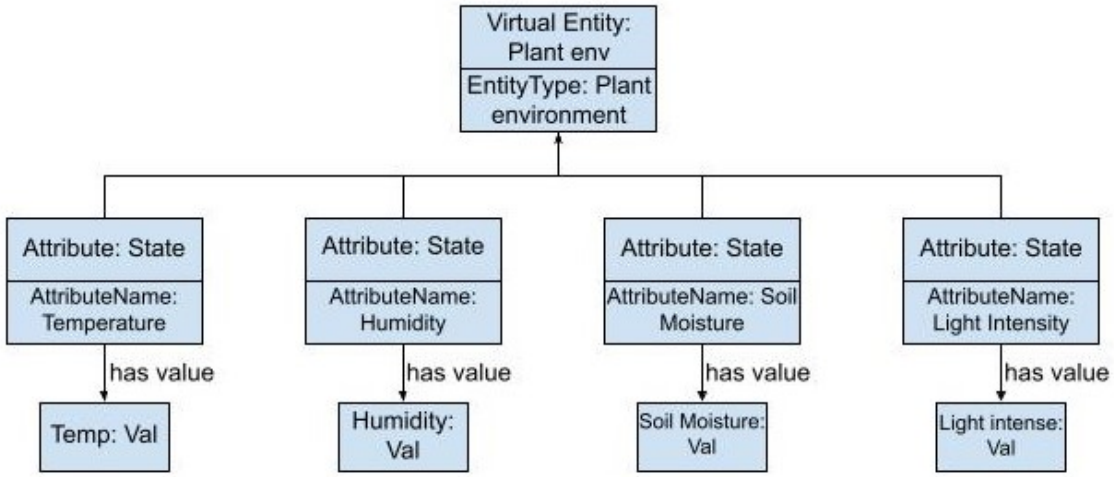

Figure 2.3: Information Model Specification

2.2.4 Service Specification

Services includes the controller service. The controller service runs as a native service on the device and monitors temperature, humidity, soil moisture and light once every 1 second. The controller service calls the AWS MQTT service to store these measurements in the cloud.

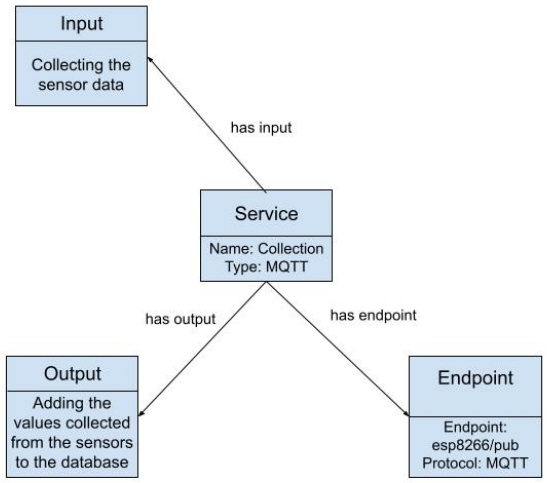

Figure 2.4: Service Specification for Data collection

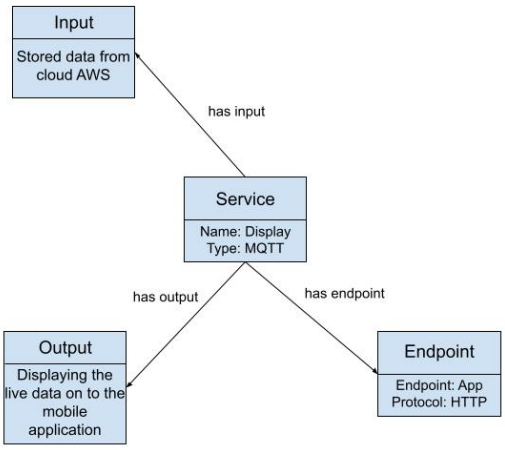

Figure 2.5: Service Specification for displaying the Data

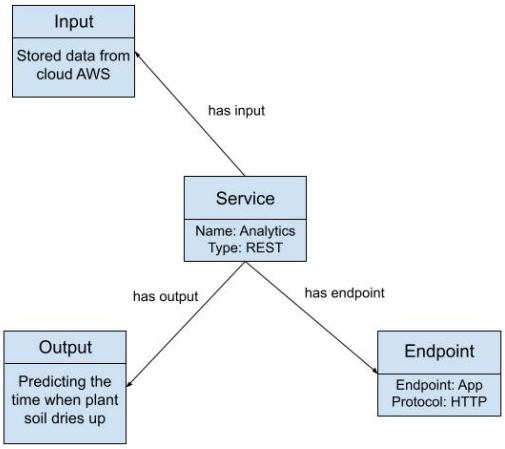

Figure 2.6: Service Specification for Analysis of Data

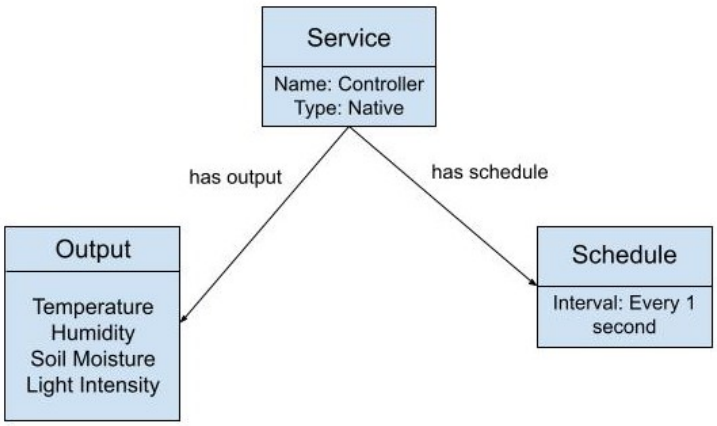

Figure 2.7: Controller Service Specification

2.2.5 IoT Deployment Level Specification

Our IoT system only has one node. In the cloud, a database and an application are created. It is appropriate for solutions involving large amounts of data and computationally intensive research criteria. Data analysis is done on the cloud side and based on analysis control action is triggered using a Flask web app. Thus our IoT Deployment level under use is Level 3, as shown in the diagram below.

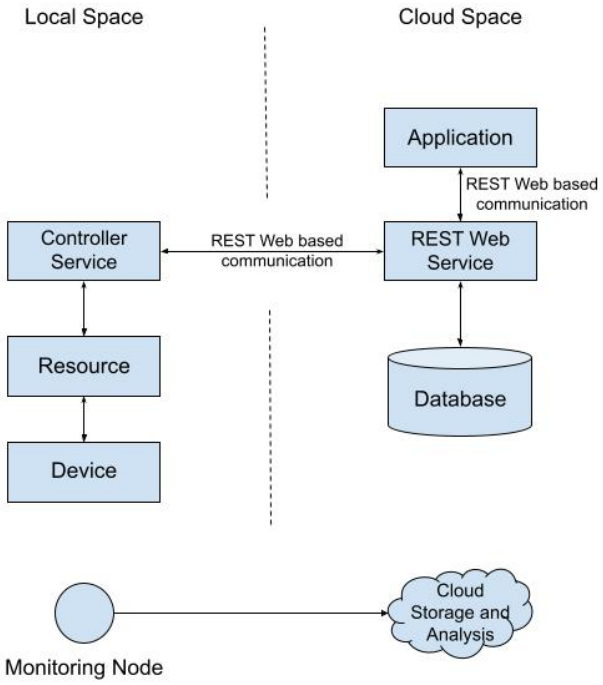

Figure 2.8: IoT Deployment Level Specification

2.2.6 Functional View Specification

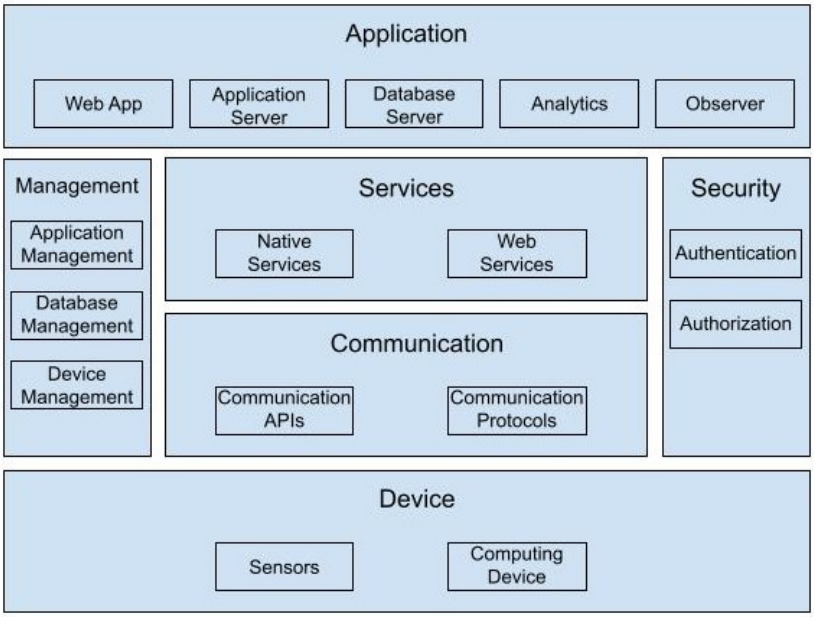

Figure 2.9: Functional View Specification

2.2.7 Operational View Specification

Native Service:	Controller Service
Web App:	Flutter App
Application Server:	Flutter Server
Database Server:	Lively Cloud Storage
Analytics:	Python
Observer:	Cloud App, Mobile App
Application Management:	Flutter App Management
Database Management:	AWS DynamoDB Management
Authentication:	Web App
Authorization:	Web App
Device Management:	NodeMCU and ESP8266 Management
Communication APIs:	REST APIs
Communication Protocols:	Network Layer: IPv6, Transport: TCP, Application: HTTP
Computing Device:	NodeMCU, ESP8266
Sensor:	Temperature, Humidity, Soil Moisture, Light sensors

Table 2.1: Mapping of functional groups with operational entities

2.2.8 Device and Component Integration

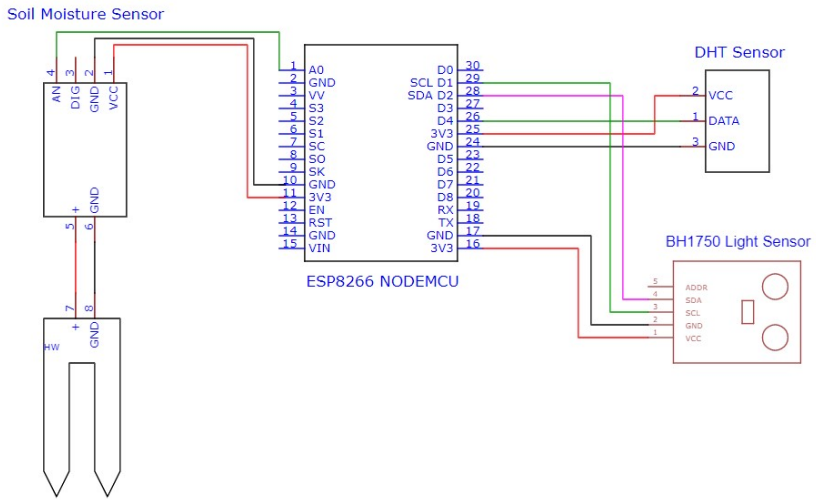

Figure 2.10: Circuit Diagram

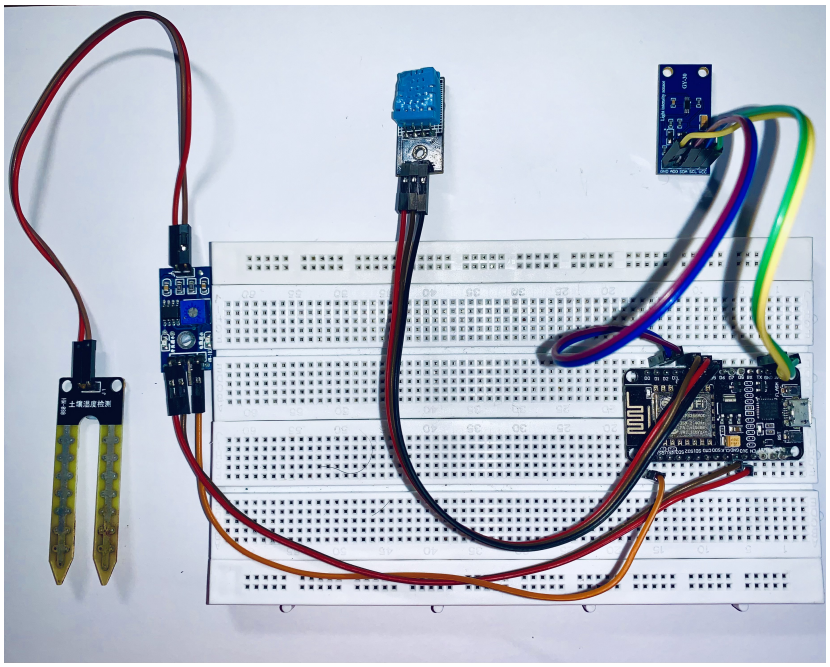

Figure 2.11: Implementation of the Circuit

2.3 Block Diagram

The system has functions as shown in parts as shown in the block diagram. The three sensors, namely soil moisture, light intensity and temperature humidity sensor publish the data to the AWS IoT Core using MQTT protocol. The IoT Analytics application and the Flutter application access the data from AWS MQTT client and analyse or display the data.

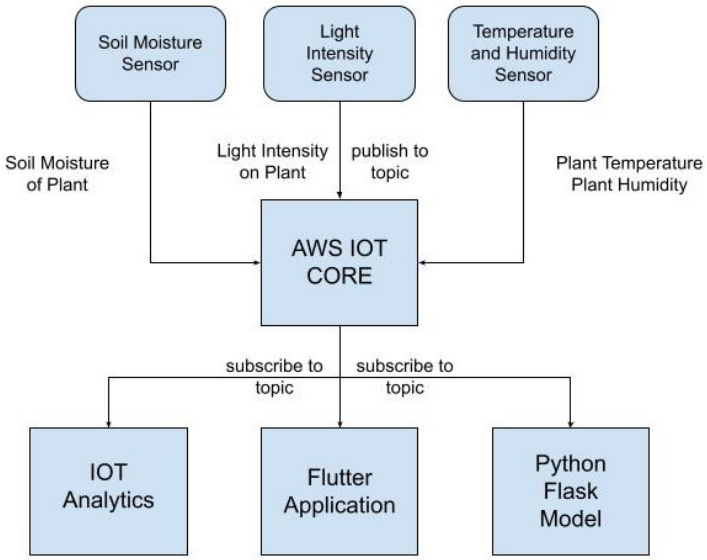

Figure 2.12: Block Diagram

Chapter 3

Implementation and Result Analysis

3.1 Interfacing with Real-time Data

3.1.1 Physical Interfacing with Real-time Data

Connect the sensors to the NodeMCU board as shown in the circuit diagram. We connect the NodeMCU board to the computer using the Micro USB cable to upload the code from Arduino IDE. Install the required Arduino libraries. Write the following code that collects data from the respective sensors and prints the same on the serial monitor.

```
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include "DHT.h"
#include <Wire.h>
#include <BH1750.h>

//SETTING UP DHT SENSOR PIN
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

//SETTING UP SOIL MOISTURE SENSOR PIN
const int sensor_pin=A0;

BH1750 lightMeter;

void setup()
{
  Serial.begin(9600);
  Wire.begin();
  lightMeter.begin();
}
```

```
void loop()
{
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  float moisture_reading = analogRead(sensor_pin);
  float moisture_percentage = ( 100.00 -
  ( (moisture_reading/1023.00) * 100.00 ) );
  float lux = lightMeter.readLightLevel();

  Serial.print("Soil Moisture(%) = ");
  Serial.print(moisture_percentage);
  Serial.print(", ");
  Serial.print("Humidity = ");
  Serial.print(humidity);
  Serial.print(", ");
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.print(", ");
  Serial.print("Light = ");
  Serial.print(lux);
  Serial.println(".");
  delay(1000);
}
```

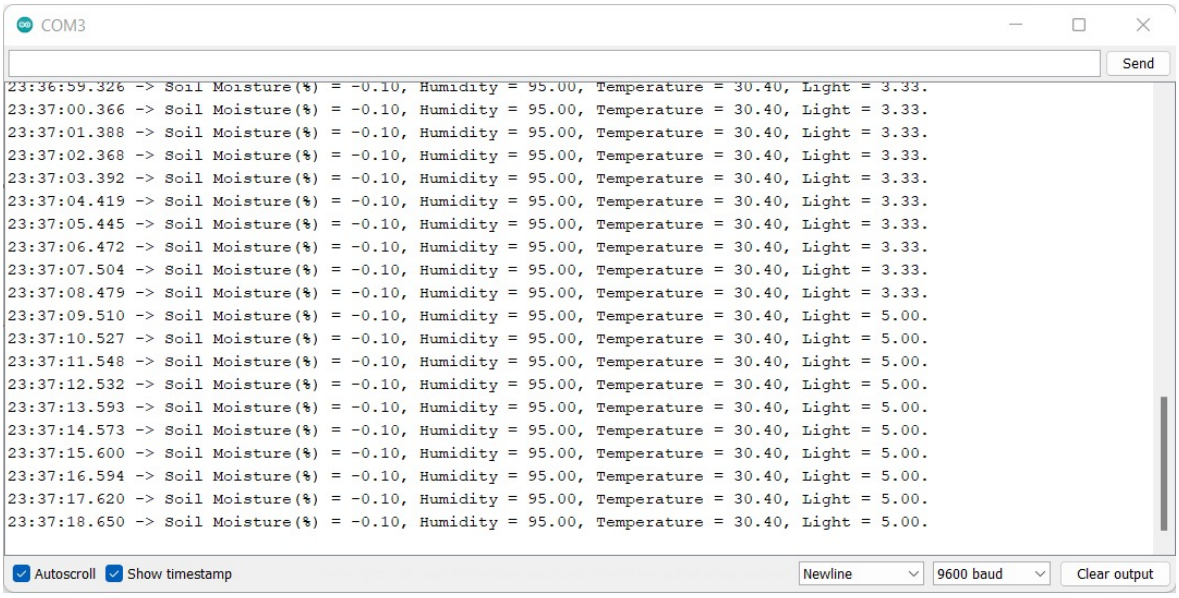

Figure 3.1: Screenshot of serial monitor

3.1.2 Cloud Interfacing with Real-time Data

AWS IoT provides cloud services and device support that you can use to implement IoT solutions. AWS IoT Core provides the services that connect your IoT devices to

the AWS Cloud so that other cloud services and applications can interact with your internet-connected devices.

AWS IoT lets you select the most appropriate and up-to-date technologies for your solution. To help you manage and support your IoT devices in the field, AWS IoT Core supports these protocols:

- MQTT (Message Queuing and Telemetry Transport)
- MQTT over WSS (Websockets Secure)
- HTTPS (Hypertext Transfer Protocol – Secure)
- LoRaWAN (Long Range Wide Area Network)

We are using AWS MQTT client for publishing real-time data to AWS from the ESP8266 board. MQTT is a lightweight and widely adopted messaging protocol that is designed for constrained devices.

Create an account on AWS using the email ID and password. Go to the IoT Core service from the AWS console. Create a thing associated with our NodeMCU ESP8266 board. Specifying thing properties, configure device certificate and attach policies to certificate. Download and store the certificates and keys in the same folder as the .ino code file.

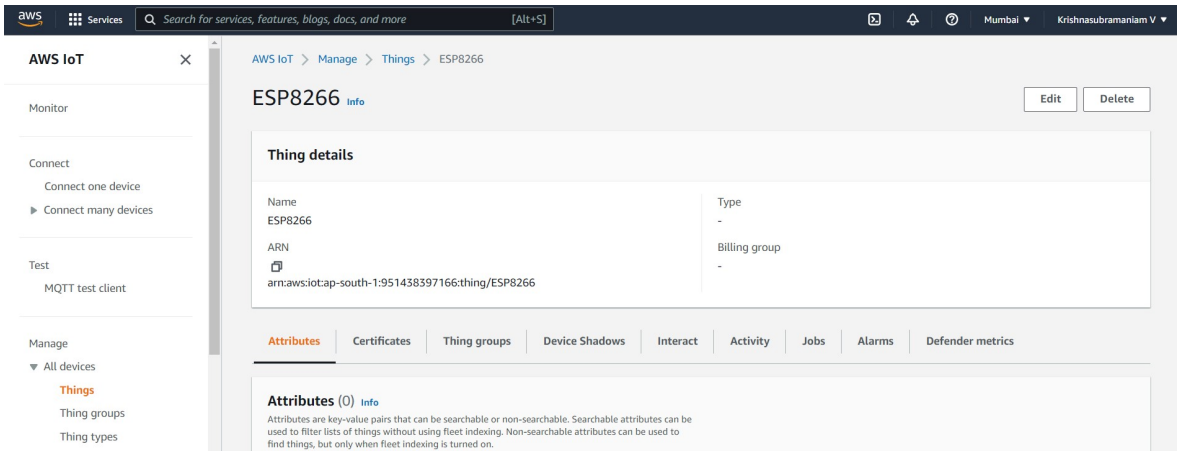

Figure 3.2: ESP8266 thing created on AWS

Install the required libraries and write the following code that connects and publishes the code to the AWS MQTT client.

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <time.h>
#include "secrets.h"
```

```
#include "DHT.h"
#include <Wire.h>
#include <BH1750.h>

#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

const int sensor_pin=A0;

BH1750 lightMeter;

unsigned long lastMillis = 0;
unsigned long previousMillis = 0;

#define AWS_IOT_PUBLISH_TOPIC   "esp8266/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp8266/sub"

WiFiClientSecure net;

BearSSL::X509List cert(cacert);
BearSSL::X509List client_crt(client_cert);
BearSSL::PrivateKey key(privkey);

PubSubClient client(net);

time_t now;
time_t nowish = 1510592825;


void NTPConnect(void)
{
  Serial.print("Setting time using SNTP");
  configTime(TIME_ZONE * 3600, 0 * 3600, "pool.ntp.org", "time.nist.gov");
  now = time(nullptr);
  while (now < nowish)
  {
    delay(500);
    Serial.print(".");
    now = time(nullptr);
  }
  Serial.println("done!");
  struct tm timeinfo;
  gmtime_r(&now, &timeinfo);
  Serial.print("Current time: ");
  Serial.print(asctime(&timeinfo));
}
```

```
void messageReceived(char *topic, byte *payload, unsigned int length)
{
  Serial.print("Received [");
  Serial.print(topic);
  Serial.print("]: ");
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}


void connectAWS()
{
  delay(3000);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println(String("Attempting to connect to SSID: ") +
  String(WIFI_SSID));

  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(1000);
  }

  NTPConnect();

  net.setTrustAnchors(&cert);
  net.setClientRSACert(&client_crt, &key);

  client.setServer(MQTT_HOST, 8883);
  client.setCallback(messageReceived);


  Serial.println("Connecting to AWS IOT");

  while (!client.connect(THINGNAME))
  {
    Serial.print(".");
    delay(1000);
  }

  if (!client.connected()) {
    Serial.println("AWS IoT Timeout!");
```

```
    return;
  }
  // Subscribe to a topic
  client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);

  Serial.println("AWS IoT Connected!");
}


void publishMessage(float moisture, float humidity,
float temperature, float light)
{
  StaticJsonDocument<200> doc;
  doc["time"] = millis();
  doc["moisture"] = moisture;
  doc["humidity"] = humidity;
  doc["temperature"] = temperature;
  doc["light"] = light;
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer); // print to client

  client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}


void setup()
{
  Serial.begin(9600);

  Wire.begin();
  lightMeter.begin();

  connectAWS();
}


void loop()
{
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  float moisture_reading = analogRead(sensor_pin);
  float moisture_percentage = ( 100.00 -
  ( (moisture_reading/1023.00) * 100.00 ) );
  float lux = lightMeter.readLightLevel();

  now = time(nullptr);
  if (!client.connected())
  {
```

```
    connectAWS();
  }
  else
  {
    client.loop();
    if (millis() - lastMillis > 1000)
    {
      lastMillis = millis();
      if(moisture_percentage > 0 && !isnan(humidity) &&
      humidity<100 && !isnan(temperature)){
        publishMessage(moisture_percentage, humidity,
        temperature, lux);
      }
    }
  }
}
```

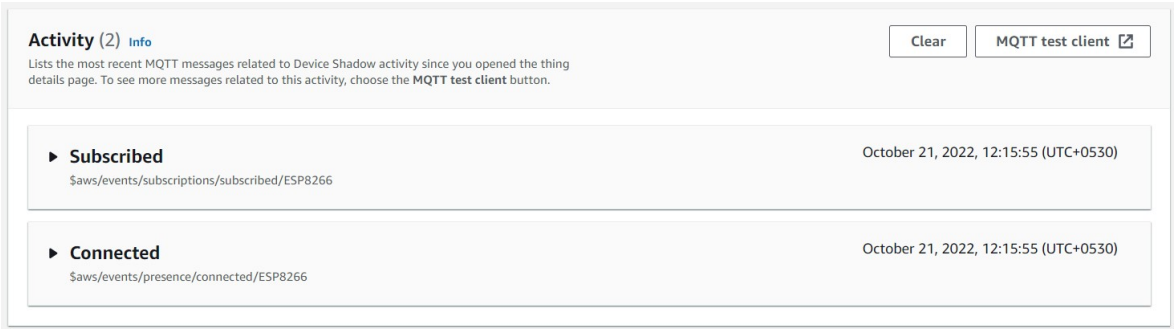

Figure 3.3: AWS Thing showing the live device activity

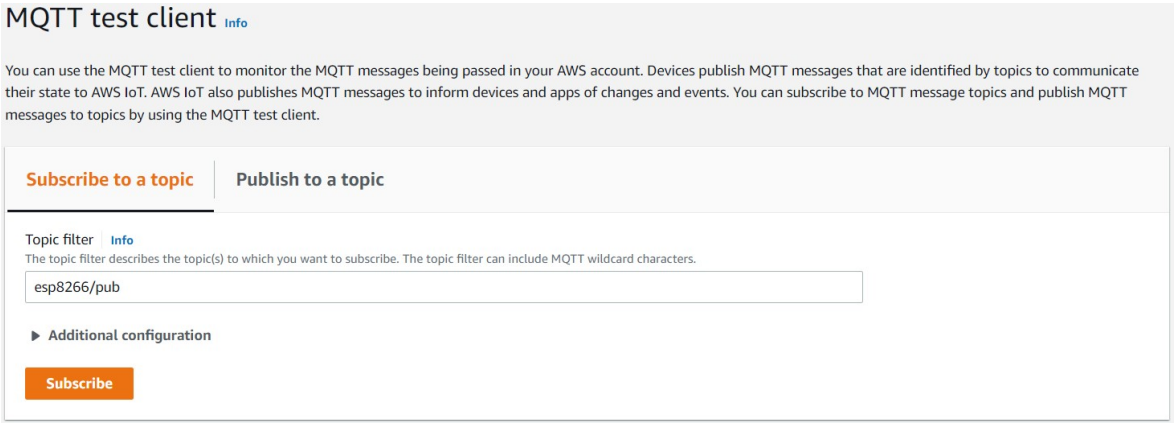

Figure 3.4: Subscribing to esp8266/pub on Test MQTT Client

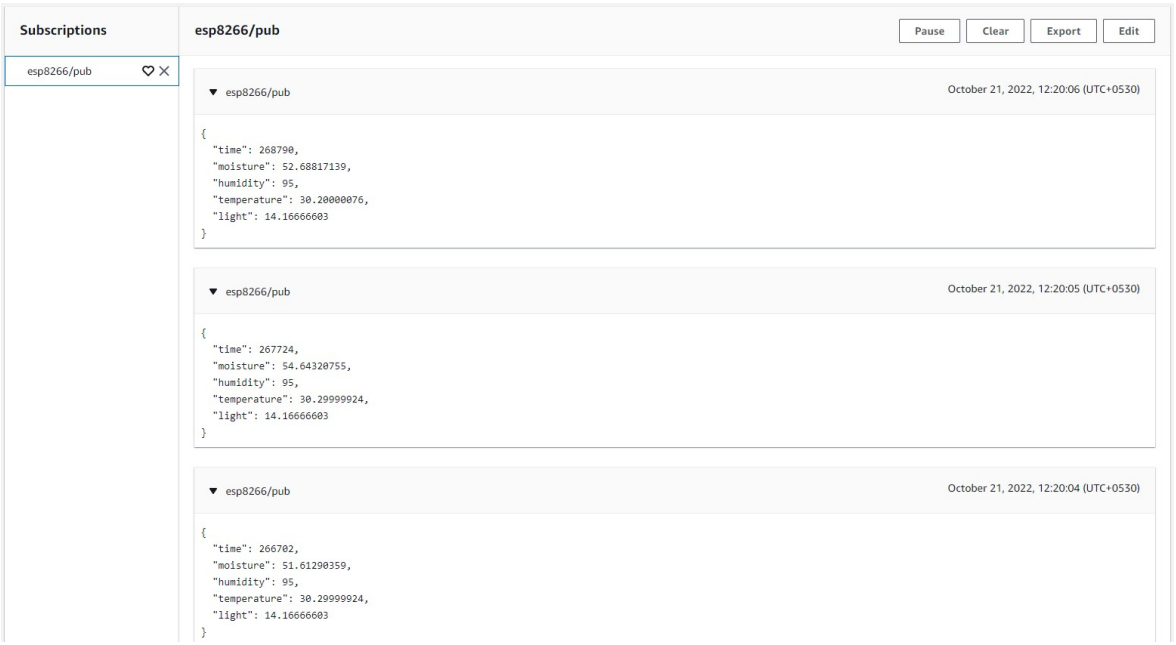

Figure 3.5: Live MQTT data published from ESP8266 board

3.2 Outcome of Analytics

We have built the analytics application using python Flask framework. Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core. It does have many cool features like url routing, template engine. It is a WSGI web app framework. Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine. Flask is often referred to as a microframework. It is designed to keep the core of the application simple and scalable. Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.

Our real-time data is called a time series data. Time series data is data collected on the same subject at different points in time. The analytics application captures the MQTT data for a fixed number of times and stores it in a dataset that will be used later. The stored data can be visualised using the same app as shown.

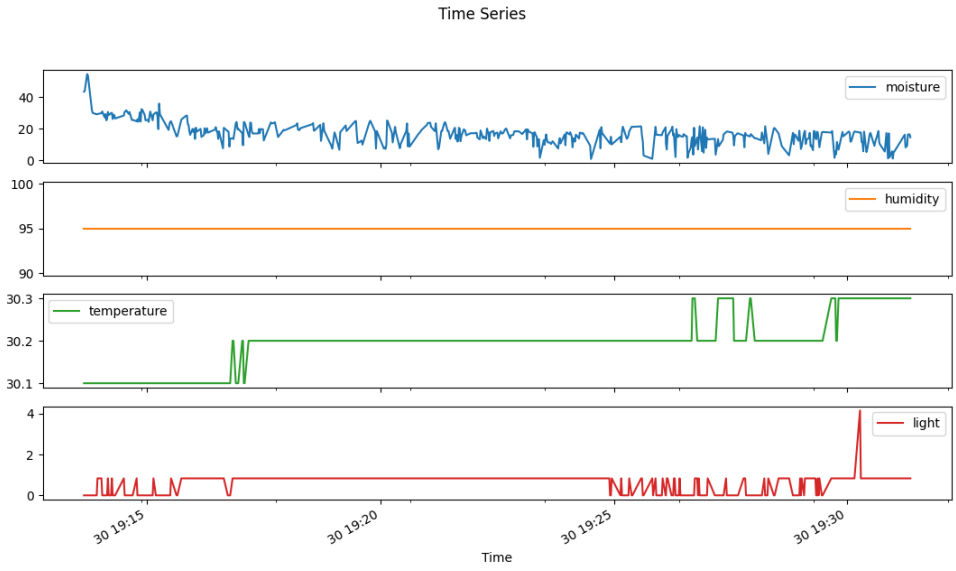

Figure 3.6: Stored dataset visualization

Time series forecasting is exactly what it sounds like; predicting unknown values. Time series forecasting involves the collection of historical data, preparing it for algorithms to consume, and then predicting the future values based on patterns learned from the historical data.

A Univariate time series, as the name suggests, is a series with a single time-dependent variable. For example, if you are tracking hourly temperature values for a given region and want to forecast the future temperature using historical temperatures, this is univariate time series forecasting.

A Multivariate time series has more than one time-dependent variable. Each variable depends not only on its past values but also has some dependency on other variables. This dependency is used for forecasting future values.

We have used Statistical model - ARIMA for prediction of multivariate time series. ARIMA is one of the most popular classical methods for time series forecasting. It stands for autoregressive integrated moving average and is a type of model that forecasts given time series based on its own past values, that is, its own lags and the lagged forecast errors. ARIMA consists of three components: The "AR" part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior observed) values. The "MA" part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The "I" (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

We have used this multivariate time series prediction model to predict the time

at which the level of soil moisture becomes less than 10%. The prediction can be visualised as shown here.

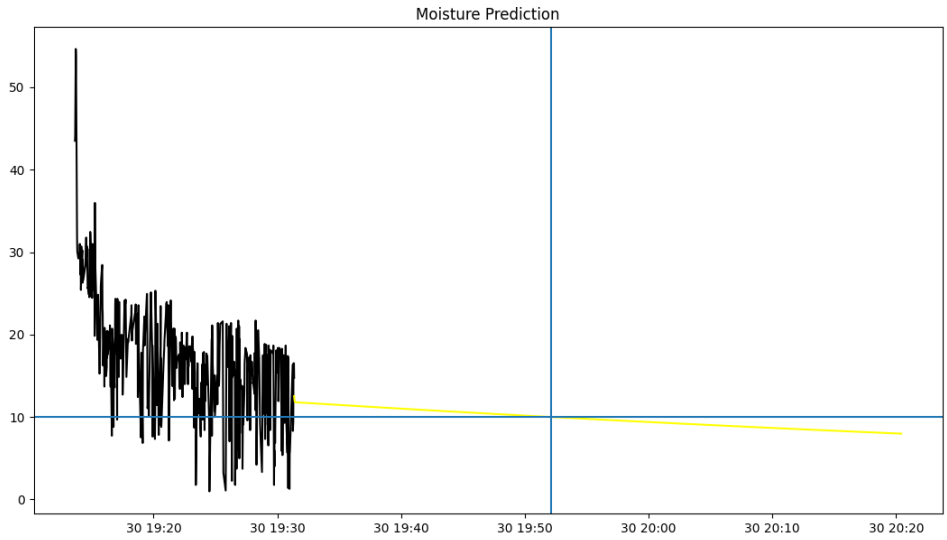

Figure 3.7: Visualization of prediction

3.3 Application Development

The application is built upon Flutter which is an open-source UI software development kit (SDK) created by Google. It is widely used to develop cross-platform applications for Android, iOS, websites etc. By coding in Dart, an open-source programming language for developing the Flutter app, the app can be built upon any operating system and create a consistent user app experience for various mobile devices.

To connect to AWS, we have used the mqtt_client package. It is a server and browser based MQTT client for Dart. The client is designed to take as much MQTT protocol work off the user as possible, connection protocol is handled automatically as are the message exchanges needed to support the different QOS levels and the keep alive mechanism. This allows the user to concentrate on publishing/subscribing and not the details of MQTT itself.

The functions which we have implemented are:

- Home Page : Created the landing page for our application.
- Local Page: Alerts the user by showing them notifications on watering the plant based on their date and time of the mobile application.
- NotfiScreen Page: Created a function to make the UI of the notification to be shown to the user.
- PlantScreen Page: To display the sensor readings and display a graph depicting the health of the plant. Also showing them the date and time to water the plant.

Following are the packages and dependencies used in building the application:
flutter:
sdk: flutter
cupertino_icons: ^1.0.5 fl_chart: ^0.55.1
syncfusion_flutter_charts: ^20.2.49
get: ^4.6.5
mqtt_client: ^9.7.2
http: ^0.13.5
flutter_local_notifications: ^9.6.0
timezone: ^0.8.0
rxdart: ^0.27.4

3.4 Implementation Code and Snapshots

Code to get the predictions for watering the plant:

```
Future<String> getTimestamp() async {
    final response = await http
        .get(Uri.parse('https://plant-analytics.onrender.com/predict'));
    if (response.statusCode == 200) {
      return response.body;
```

```
    } else {
      return "null";
    }
  }

  late Future<String> futureString;
  late final LocalNotificationService service;
  @override
  void initState() {
    service = LocalNotificationService();
    service.initialize();
    super.initState();
    futureString = getTimestamp();
}
```

Code to get the values from AWS using MQTT Server client

```
final client =
    MqttServerClient('a1n0zq7k1z3sqk-ats.iot.ap-south-1.amazonaws.com', '');
isConnected = mqttConnect('ninad');
final mqttReceivedMessages = snapshot.data as
List<MqttReceivedMessage<MqttMessage?>>?;

final recMess = mqttReceivedMessages![0].payload as MqttPublishMessage;

final pt =
    MqttPublishPayload.bytesToStringAsString(recMess.payload.message);

final result = jsonDecode(pt);
var arr = ["High", "Moderate", "Low"];
double avg = (q.length == 0) ? 0.0 : q.elementAt(q.length - 1);

q.add(result["moisture"] == null ?
    0.0 : approxRollingAverage(avg, result["moisture"]));

if (q.length > 5) {
    q.removeFirst();
}
DateTime now = DateTime.now();
if (result["moisture"] <= 10.0) {
    notification(2);
} else if ((now.hour <= 15 && now.hour >= 7) && result["light"] < 1.2) {
    notification(3);
}
double first = q.elementAt(0);
double second = q.elementAt(1) != null ? q.elementAt(1) : 0.0;
double third = q.elementAt(2) != null ? q.elementAt(2) : 0.0;
double fourth = q.elementAt(3) != null ? q.elementAt(3) : 0.0;
double fifth = q.elementAt(4) != null ? q.elementAt(4) : 0.0;
```

Code to show the soil moisture content graph:

```
SizedBox(
height: 150,
    child: SfCartesianChart(
      primaryXAxis: CategoryAxis(),
        palette: const <Color>[
            Color.fromARGB(255, 116, 214, 74),
       ],
       legend: Legend(isVisible: false),
       series: <LineSeries<SalesData, String>>[
        LineSeries<SalesData, String>(
            dataSource: <SalesData>[
               SalesData('-5', q.elementAt(0)),
               SalesData('-4', q.elementAt(1)),
               SalesData('-3', q.elementAt(2)),
               SalesData('-2', q.elementAt(3)),
               SalesData('now', q.elementAt(4))
                 ],
                xValueMapper: (SalesData sales, _) => sales.year,
                yValueMapper: (SalesData sales, _) => sales.sales,
                dataLabelSettings: const DataLabelSettings(
                    isVisible: false,
                ),
            )
      ],
    ),
),
```

Code to show notifications on the screen:

```
Future<void> notification(int val) async {
    if (val == 1) {
      await service.showNotification(
          uid: 0,
          title: 'Connection Error',
          body: "Your Plant isn't connected!");
    } else if (val == 2) {
      await service.showNotification(
          uid: 0,
          title: 'Dry Plant !',
          body: "Water your Plant it'll dry up soon !");
    } else if (val == 3) {
      await service.showNotification(
          uid: 0,
          title: 'Sunlight Needed!',
          body: "Place you plant in sunlight for a few hours!");
    }
  }
```

```
  Future<void> timedNotif(int seconds) async {
    await service.showScheduledNotification(
        uid: 0,
        title: 'Time to Water !',
        body: "Water your lovely Plant!",
        seconds: seconds);
  }
```

Code to connect the MQTT Server client from the app:

```
Future<bool> mqttConnect(String uniqueID) async {
    setStatus("Connecting MQTT Broker");
    ByteData rootCA = await rootBundle.load('assets/certs/RootCA.pem');
    ByteData deviceCert =
        await rootBundle.load('assets/certs/DeviceCertificate.crt');
    ByteData privateKey = await rootBundle.load('assets/certs/Private.key');

    SecurityContext context = SecurityContext.defaultContext;
    context.setClientAuthoritiesBytes(rootCA.buffer.asUint8List());
    context.useCertificateChainBytes(deviceCert.buffer.asUint8List());
    context.usePrivateKeyBytes(privateKey.buffer.asUint8List());

    client.securityContext = context;
    client.logging(on: true);
    client.keepAlivePeriod = 20;
    client.port = 8883;
    client.secure = true;
    client.onConnected = onConnected;
    client.onDisconnected = onDisconnected;
    client.pongCallback = pong;

    final MqttConnectMessage connMess =
        MqttConnectMessage().withClientIdentifier(uniqueID).startClean();

    client.connectionMessage = connMess;

    await client.connect();

    if (client.connectionStatus!.state == MqttConnectionState.connected) {
      // ignore: avoid_print
      print("Connected to AWS Successfully");
    } else {
      return false;
    }

    const topic = "esp8266/pub";
    client.subscribe(topic, MqttQos.atLeastOnce);

    return true;
```

```
}

void setStatus(String content) {
    setState(() {
      statusText = content;
    });
}

void onConnected() {
    setStatus("Client connection was successful");
}

void onDisconnected() {
    setStatus("Client Disconnected");
}

void pong() {
    setStatus("Ping response client callback invoked");
}
```

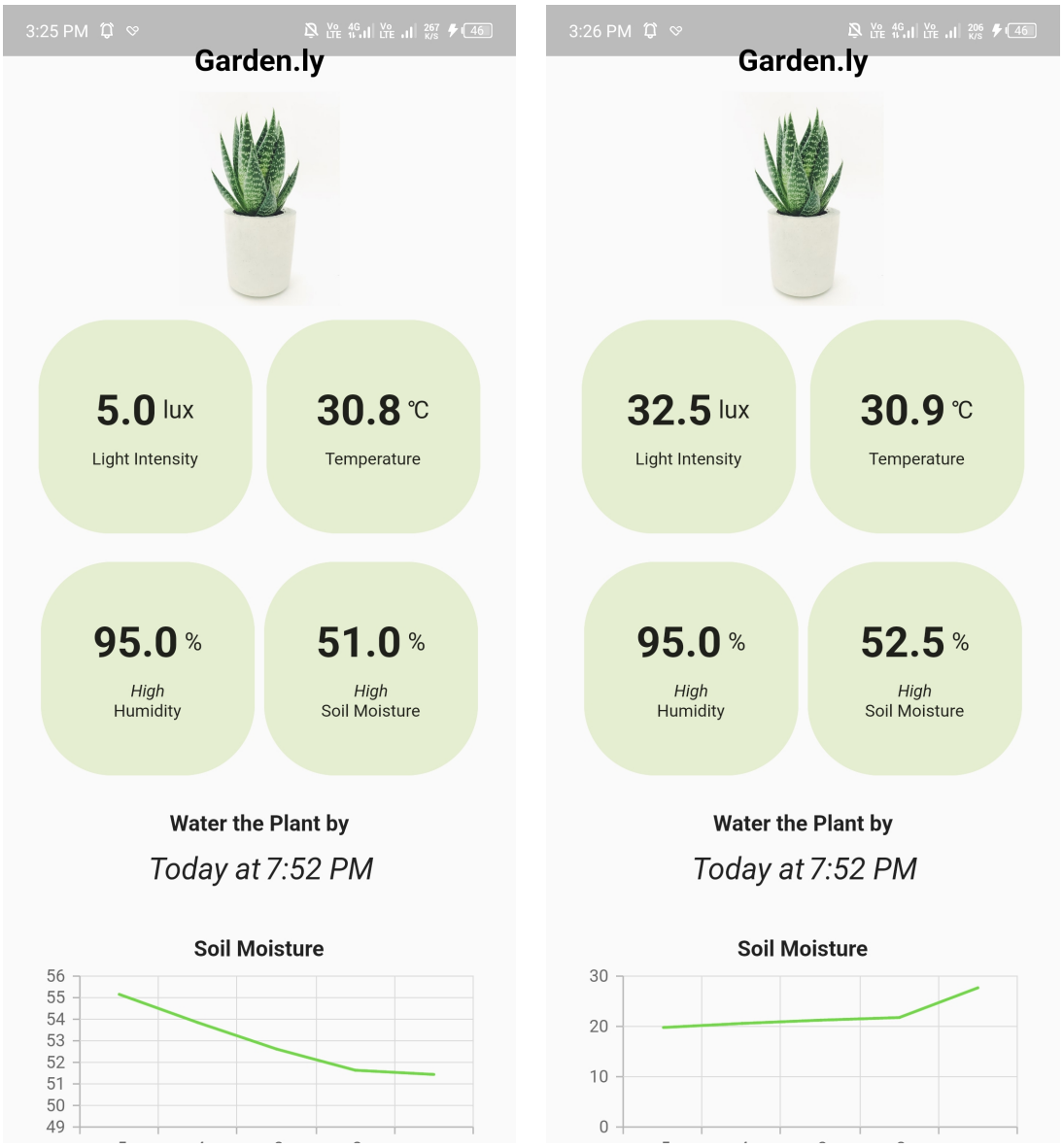

Figure 3.8: Screenshots of the application

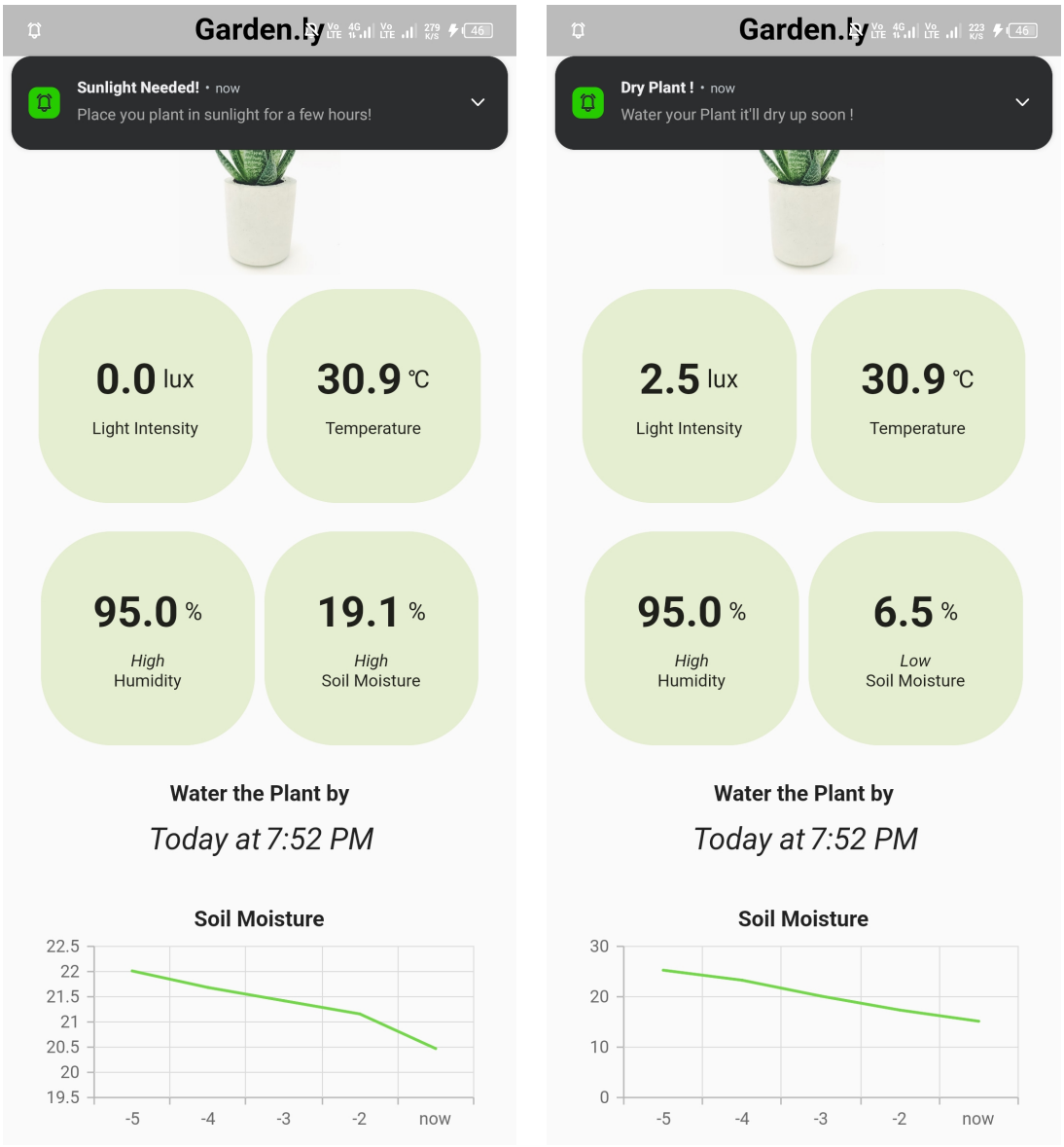

Figure 3.9: Notifications in the application

3.5 Link to Video Demonstration

Video Demonstration of Garden.ly - our Plant Environment Monitoring System https://bit.ly/3F0KcZ6.

Chapter 4

Conclusion and Future Scope

The plant environment monitoring system we have created fulfills the need to inform the plant owner to be aware of the needs of the plant to grow in a healthy way. By notifying the user to water the plant when the moisture is low, predicting the time by which user should water the plant before the moisture level drops drastically low and to move the plant to a more sun-lit place when the light intensity is low allows the user to actively modify the well being environment of the plant.

In the future release of the application, there are many possible improvements that can be added such as:

i) Adding rotation shield with motor for automatic shade in times of intense sunlight

ii) Adding a water pump which automatically waters when the soil moisture is below the threshold.

iii) Inserting a NPK sensor in the soil to monitor the quality of the soil.

iv) Allows user to create their own profile and login from multiple devices/ modes to access the same data

v) Expand the system so that same user can use multiple plant sensors

References

[1] Pawar, P., Gawade, A., Soni, S., Sutar, S., Sonkamble, H. *'IOT Based Smart Plant Monitoring System'* International Journal for Research in Applied Science and Engineering Technology, 10(5), 505–510. (2022, May 31)

[2] Wang, Miss. *'Advanced Multivariate Time Series Forecasting Models'* Journal of Mathematics and Statistics. 14. 253-260. 10.3844/jmssp.2018.253.260. (2020)