SMART FARMER - IOT ENABLED SMART FARMING APPLICATION

NALAIYA THIRAN PROJECT

Submitted by

NAVEEN PM
NAVEEN T
NIRANJAN N
PRASANTH M

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

PSNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution; Affiliated to Anna University, Dindigul)

December 2022

TABLE OF CONTENTS

PROJECT REPORT

CHAPTER 1 - INTRODUCTION

1.1 Project Overview

Agriculture has always been the backbone of any economic development. To promote further growth of agriculture, it must be integrated with modern practices and technologies. With the wide spread acceptance of technology, it can be used in farming to make farmers perform their activity with ease. Electronics and IoT has found its application in many of the personal assistant devices. This can be extended to many vital fields like agriculture where their assistants can help solve many issues faced. Electronics can help devices get physically connected with their operational environment and analyze and collect data. IoT can help analyze and transfer the data to the user. The combination of these gives rise to an all-in-one device capable of carrying out a task.

1.2 Purpose

In recent times, the erratic weather and climatic changes have caused issues for farmers in predicting the perfect conditions to initiate farming. Though on a superficial scale it seems unpredictable, it can be determined with certain parameters with which crop planning can be done. Maintenance of farm fields during and after cultivation are also important. These can be performed by measuring soil moisture, humidity and temperature.
Measurement of these parameters are performed using physical sensors. This system is in turn connected to IoT system which can provide a easy to access interface for farmers to read, analyze and take action based on

the presented condition. Taking it a step ahead, the system can also gain access to motors and other electrical equipment used in farming and automate their operation. This can help with unsupervised operation ensuring accuracy and lesser response time.

CHAPTER 2 - LITERATURE SURVEY

2.1 Existing problem

There has been several attempts and solution to help farmers adopt technological practices. Few solutions restricted their performance with just suggestions and alerts. While few employed IoT independent electronics. Few of the cases of previous attempts and researches are described below.

i. "IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing & solar technology". This work was performed using Cloud computing platform (Things Speak) for data acquisition. The circuit was designed using Arduino and DHT 11 sensors.

ii. "Smart Farming using IoT, a solution for optimally monitoring farming conditions". This work used ESP-32 based IoT platform and Blynk mobile application.

iii. "Smart farming using IoT". The automation and interface part made use of water pump and HTTP protocol for parameters monitoring using website.

The above stated prior works lacked one or two features, which when included could have enhanced the performance. In the first work, including a Raspberry Pi based controller in place of Arduino can help reduce the design area while also providing microcontroller with additional UI and IoT interfaces. In the second stated work, going with MIT app inventor instead of Blynk application can improve the possibility of feature expansion. Farmers or developers won't need to go for a paid version of the app to include new features. In the third work, control of water pump can be enhanced with the use of servo-based water valves to direct and control the flow of water rather than using a bi-stated logic.

2.2 References

The following were the source of references:

i. https://www.researchgate.net/publication/313804002_Smart_farming_IoT_based_smart_sensors_agriculture_stick_for_live_temperature_and_moisture_monitoring_using_Arduino_cloud_computing_solar_technology.

ii. https://www.sciencedirect.com/science/article/pii/S1877050919317168

iii. *"Smart farming: IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing & solar technology",* Anand Nayyar Assistant Professor, Department of Computer Applications & IT KCL Institute of Management and Technology, Jalandhar, Punjab Er. Vikram Puri M.Tech(ECE) Student, G.N.D.U Regional Center, Ladewali Campus, Jalandhar

iv. *"Smart Farming using IoT, a solution for optimally monitoring farming conditions",* Jash Doshi; Tirth kumar ; Patel Santosh kumar Bharati.

v. *"Smart Farming Using IOT"*, CH Nishanthi; Dekonda Naveen, Chiramdasu Sai Ram , Kommineni Divya , Rachuri Ajay Kumar; ECE Dept., Teegala Krishna Reddy Engineering College, Hyderabad, India 2,3,4,5student, ECE Dept., Teegala Krishna Reddy Engineering College, Hyderabad, India.

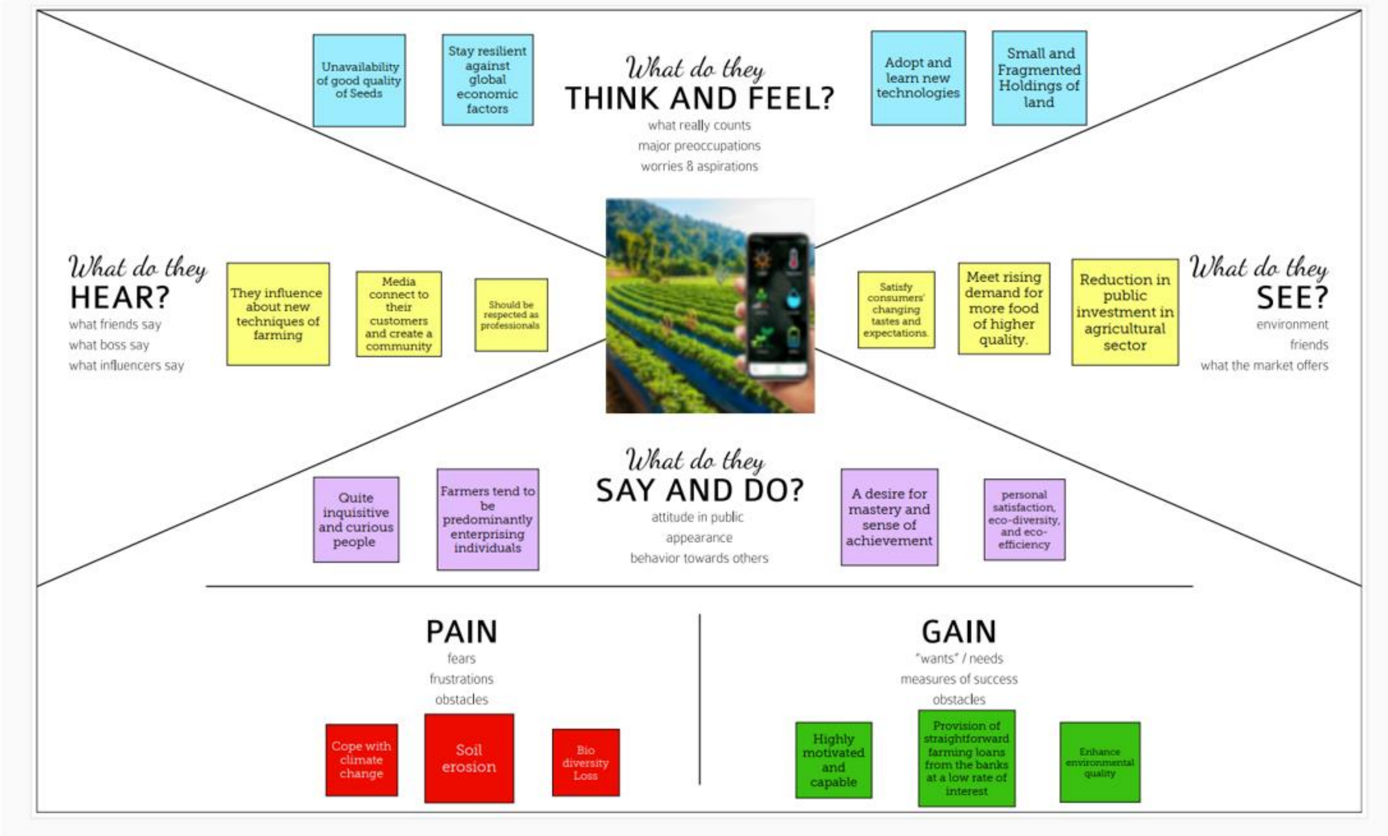
2.3 Problem Statement Definition

The problem statement in a nutshell covers all the possible technical aspects that can be included by farmer to convert farming in to smart and efficient farming. IoT enabled smart farming, on a wider perspective, concentrates on connecting all the independently operating sub-systems in farming automation into a single entity. IoT-based agriculture system helps the farmer in monitoring different parameters of his field like soil moisture, temperature, and humidity using some sensors.

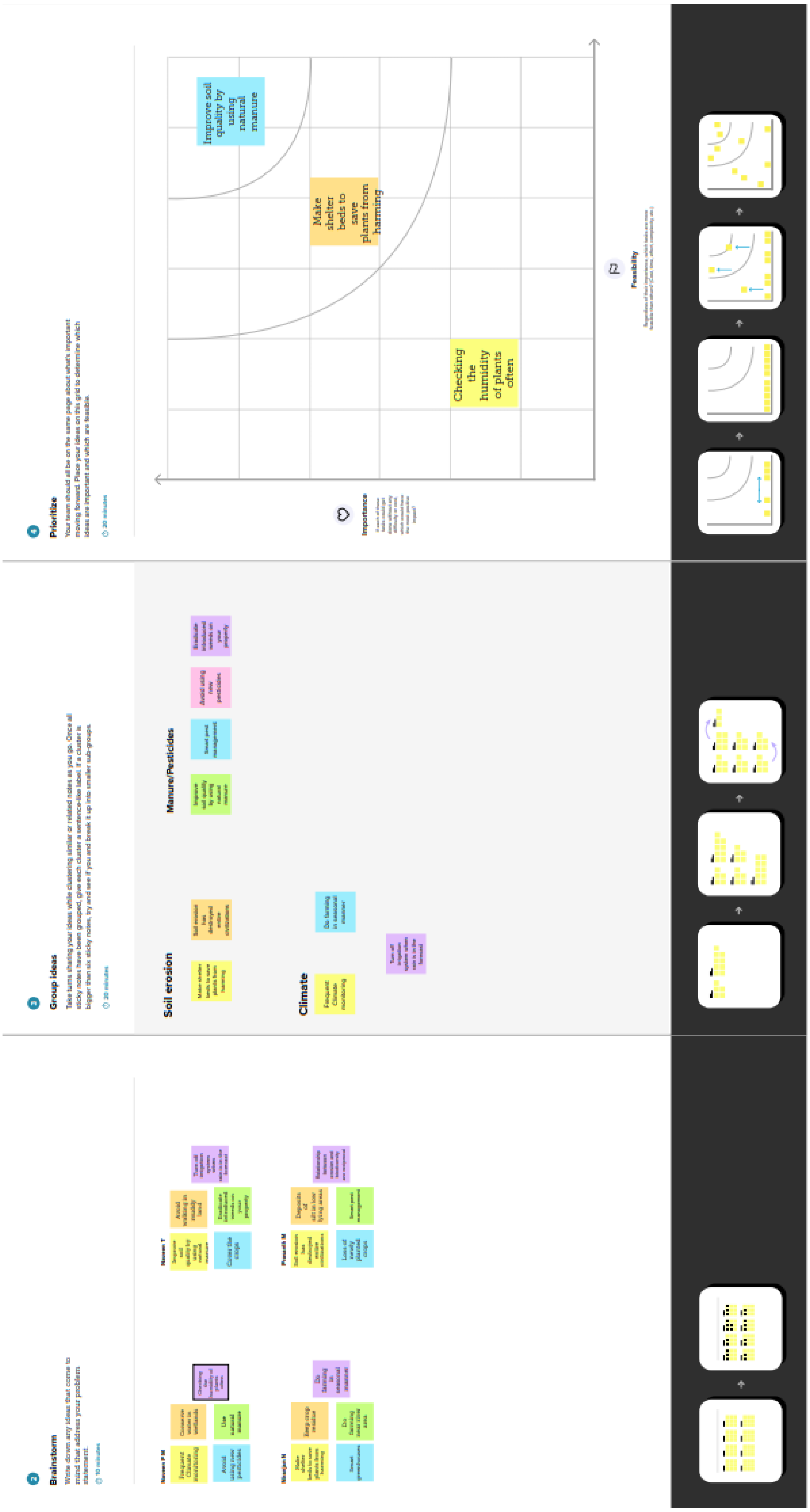
The idea of IoT is further extended with the help of mobile and web application where farmers can monitor all the sensor parameters even if the farmer is not near his field. Watering the crop is one of the important tasks for the farmers. They can make the decision whether to water the crop or postpone it by monitoring the sensor parameters and controlling the motor pumps from the mobile application itself.

CHAPTER 3 - IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas

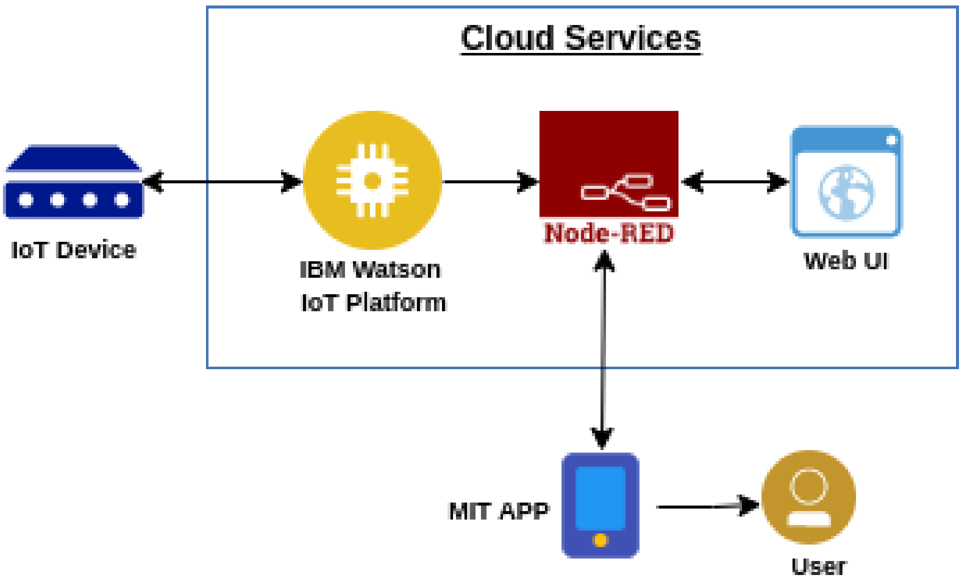


3.2 Ideation & Brainstorming


Brainstorm
Write down any ideas that come to mind that address your problem statement.
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.
Soil erosion
Manure/Pesticides
Climate
Prioritize
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.
Improve soil quality by using natural manure
Make shelter beds to save plants from harming
Checking the humidity of plants often
Importance
Feasibility

3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To deal with humidity, climate change and soil erosion. To satisfy the agricultural needs and expectations.To solve the Fear of investing in farm productivity.
2.	Idea / Solution description	By using Internet of thing,we can estimate the humidity and conditions. IoT in agriculture can be helpful in tracking soil temperature, soil moisture, and soil nutrients to enhance crop productivity.
3.	Novelty / Uniqueness	The IoT should increase the control over productivity and enable management of a greater number of resources through remote sensing. The smart farming should be much more efficient than our traditional farming.
4.	Social Impact / Customer Satisfaction	Smart farming makes it possible to increase the quality and minimize the environmental effect. It should support livelihoods through food, habitat, and jobs and providing raw materials for food and other products.
5.	Business Model (Revenue Model)	The smart farming devices designed in such a way that should be profitable compared to traditional farming methods and the device should be reusable. The cost of the devices should be less compared to cost required for traditional farming. Hence the product must be profitable it does not make losses in any cases.
6.	Scalability of the Solution	The ability of the devices to increase or decrease in performance and cost in response to changes in application. The property of a device to handle a growing amount of works by adding resource to system.



3.4 Problem Solution fit

Define CS, fit into CC

1. CUSTOMER SEGMENT(S)

The customer who are going to use this project includes

Large Scale Farmers

Small Scale Farmers

6. CUSTOMER CONSTRAINTS

Lack of proper irrigation facilities, production machinery, and access to institutional credit, difficulties procuring inputs and storing products, and negative impacts of climate were identified as the major constraints to agricultural productivity.

5. AVAILABLE SOLUTIONS

Precision Agriculture, Crop Monitoring, Irrigation Management, Fertilizer Management Weather Forecasting are best solutions for provided for the farmers.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS

Iot devices connects and interacts with each other,and the internet which means they can work together to send alert or automate other things such as sprinkler in an orchard

9. PROBLEM ROOT CAUSE RC

By adopting Iot in the agricultural sector we get numerous benefits,but still, there are challenges faced by IoT in agricultural sectors.

7. BEHAVIOUR

The customer wants to make the revolutionary propagation in the rating of the irrigation through the reliability of amount of water availability on the land.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS

Smart farming reduces the ecological footprint of farming

4. EMOTIONS: BEFORE / AFTER

Turning the face of conventional agriculture methods by not only making it optimal but also making it cost efficient for farmers and reducing crop wastage

10. YOUR SOLUTION

Our solution for this project is the smart irrigation facilities using IoT based on moisture and temperature

8. CHANNELS of BEHAVIOR

The channels of behavior recombine the ratio of the following

Online

Offline

Identify strong TR & EM

CHAPTER 4 - REQUIREMENT ANALYSIS

4.1 Functional Requirements:

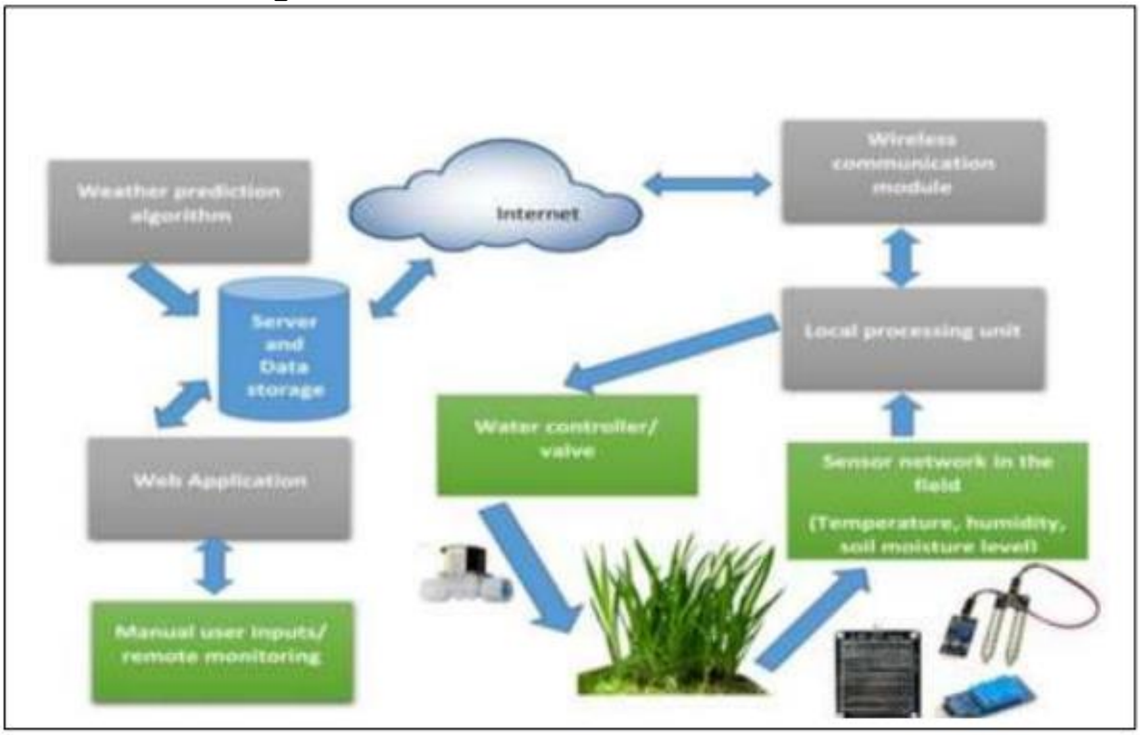
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
1	User Registration	Registration through Form
2	User Confirmation	Confirmation via Email Confirmation via OTP
3	User Login	Login with Email Id and Password
4	Forgot Password	Login with Email Confirmation Of OTP
5	Query Form	Make a note of the problems and issues faced by user when using the application
6	Weather	To find the climate information of a particular area
7	Agro Note	To list of agriculture related information like how to plant, how much litres of water that plant need in a day etc
8	Sensors	To show various data from different sensors like temperature, humidity, soil moisture
9	Database Management	To show various agriculture related data are stored
10	Exit	After user checked every information, user can exit the application

4.2 Non-functional Requirements:

FR No.	Non-Functional Requirement	Description
1	**Usability**	Effective and Easy to Use
2	**Security**	The process of protecting data from Unauthorized Access
3	**Reliability**	Consistency and Accuracy and the shared protection achieves a better trade-off between costs and reliability
4	**Performance**	Measured and estimate the performance of the Productivity
5	**Availability**	24/7 services
6	**Scalability**	Scalability is main concern for IoT platforms. It supports third party sensors. It can be easily scalable for large farming

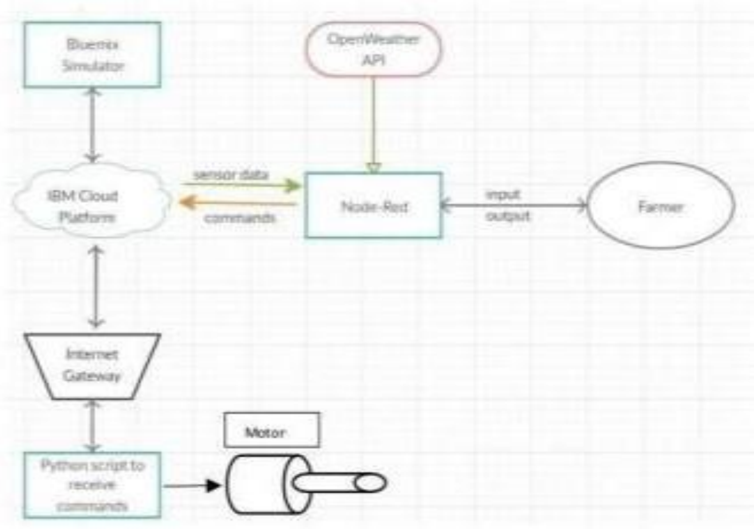
CHAPTER 5 - PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution and Technical Architecture

The technical architecture diagram is as follows:



• Using various sensors, the various soil parameters, including temperature, moisture content and humidity are measured. The results are then stored in the IBM cloud.

• The Arduino UNO is utilised as a processing unit to process the data from the sensors and weather API.

• To write the hardware, software, and APIs. NODE-RED is employed as a programming tool. In order to communicate, the MQTT protocol is used.

• A mobile application created with MIT App Inventor makes all the collected data available to the user. Depending on the sensor results, the user might decide whether or not to irrigate the crop using an app. They can control the motor switch remotely by utilising the app.

5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	Can register for the application by entering my email, password, and confirming my password	I can access my account / dashboard	High	Sprint-1
		USN-2	Will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	Can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	Can Register for the application through Gmail		Medium	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
	Login	USN-5	Can Log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Web user)						
Customer Care Executive						
Administrator						

CHAPTER 6 - PROJECT PLANNING AND SCHEDULING

6.1 Sprint Planning and Estimation

6.2 Sprint Delivery and Schedule

The Sprint schedule is as follows:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Simulation creation	USN-1	Connect Sensors and Arduino with python code	2	High	Prasanth M, Naveen P M
Sprint-2	Software	USN-2	Creating device in the IBM Watson IoT platform, workflow for IoT scenarios usingNode-Red	2	High	Prasanth M, Naveen T, Niranjan N
Sprint-3	MIT App Inventor	USN-3	Develop an application for the Smart farmerproject using MIT App Inventor	2	High	Prasanth M
Sprint-3	Dashboard	USN-3	Design the Modules and test the app	2	High	Niranjan N
Sprint-4	Web UI	USN-4	To make the user to interact with software.	2	High	Niranjan N, Prasanth M, Naveen T, Naveen P M

Project Tracker, Burndown chart:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date(Actual)
Sprint-1	20	7 Days	30 Oct 2022	06 Nov 2022	20	29 Oct 2022
Sprint-2	20	9 Days	31 Oct 2022	09 Nov 2022		05 Oct 2022
Sprint-3	20	6 Days	06 Nov 2022	13 Nov 2022		12 Oct 2022
Sprint-4	20	6 Days	11 Nov 2022	17 Nov 2022		15 Oct 2022

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let us calculate the team's average velocity (AV) per iteration unit (storage points per day).

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

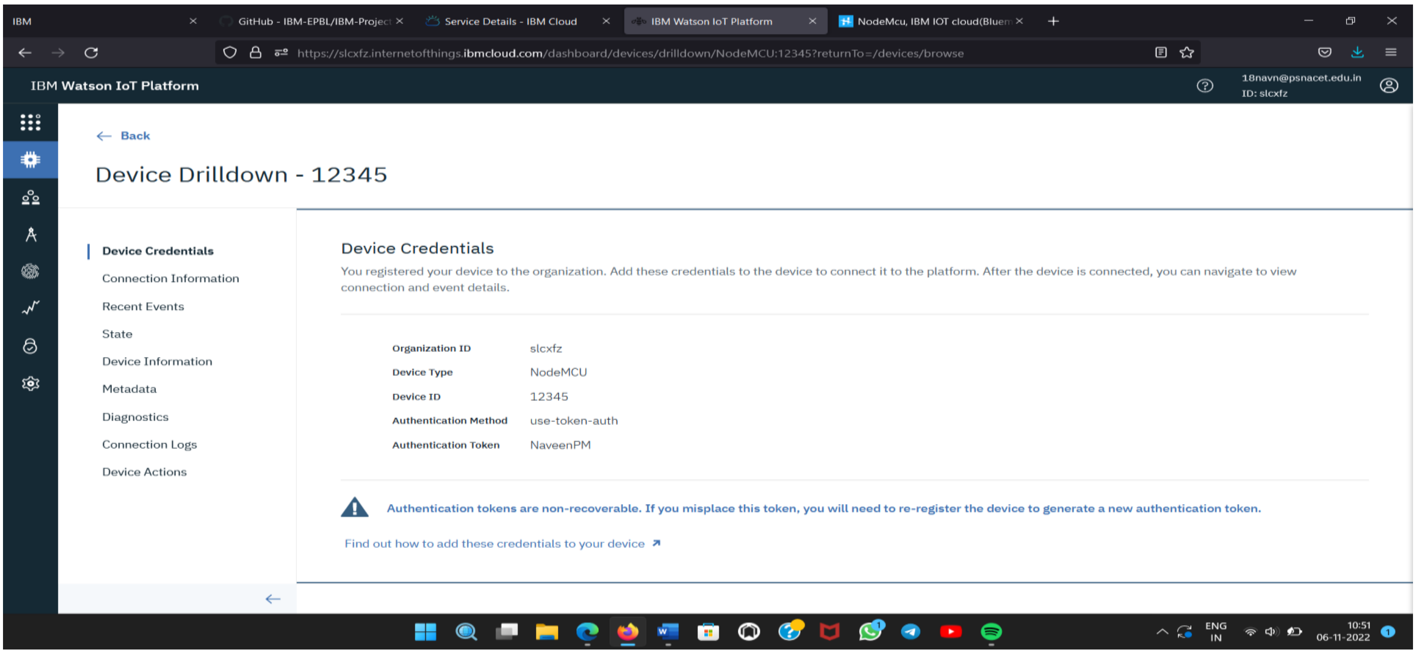
Burndown Chart:

A burndown chart is the graphical representation of work left to be done versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

CHAPTER 7 - CODING AND SOLUTIONING

- **Configuration of the IBM Watson IOT Platform and a device:**

  In the IBM Watson IOT Platform, under the catalog list, under the Internet of Things platform, a device has been created. From that the device credentials such as Device ID, Device Type, Organization ID,Authentication token were obtained.



- **Development of Python Script to publish data to IBM Watson IOT platform:**

  <u>Code</u>:

```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
#Provide your IBM Watson Device Credentials
organization = "nckdv7"
deviceType = "NodeMCU"
```

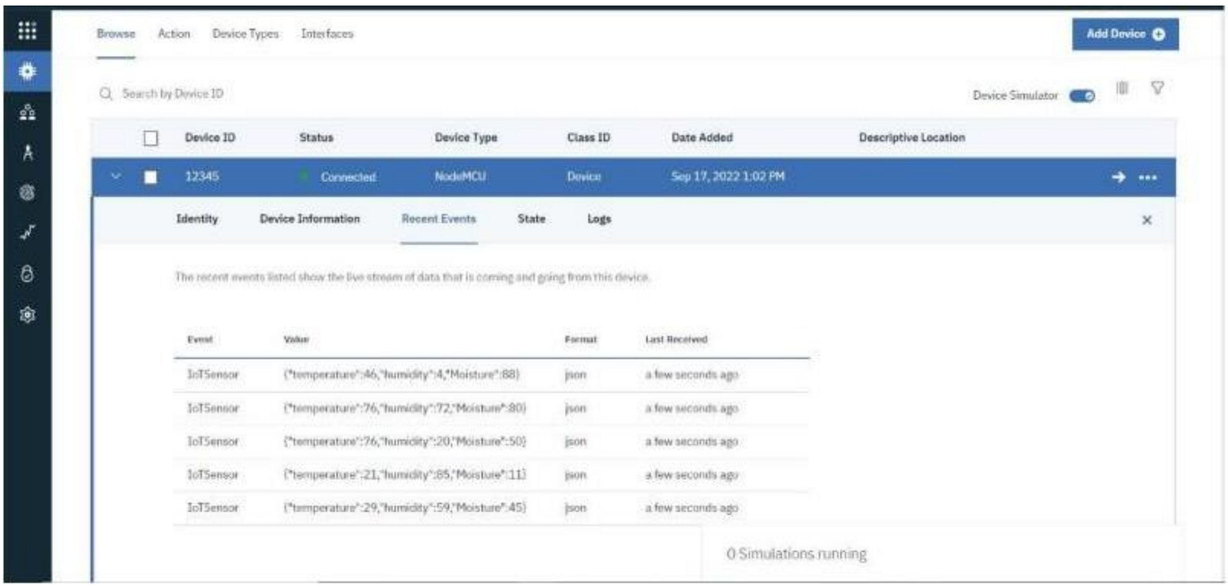
```
deviceId = "12345"
authMethod = "token"
authToken = "12345678" # Initialize GPIO
try:
    deviceOptions = {"org": organization, "type": deviceType, "id":
    deviceId, "auth-method": authMethod, "auth-token": authToken}
    deviceCli = ibmiotf.device.Client(deviceOptions)
    #..............................................
except Exception as e:
    print("Caught exception connecting device: %s" % str(e))
    sys.exit()
# Connect and send a datapoint "hello" with value "world" into the cloud
as # an event of type "greeting" 10 times
deviceCli.connect()
while True:  #Get Sensor Data from DHT11
    temp=random.randint(0,100)
    pulse=random.randint(0,100)
    moisture= random.randint(0,100)
    humidity=random.randint(0,100);
    lat = 17
    lon = 18
    data = { 'temperature' : temp, 'humidity' : humidity, 'Moisture' :
    moisture}
    #print data

def myOnPublishCallback():
    print ("Published Temperature = %s C" % temp, "Humidity = %s
    %%" % humidity, "Soil Moisture = %s %%" % moisture,"to IBM
    Watson")
```

```
success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0, on_publish=myOnPublishCallback)
if not success:
        print("Not connected to IoTF")
time.sleep(1)
deviceCli.commandCallback = myCommandCallback
# Disconnect the device and application from the cloud
deviceCli.disconnect()
```

Python Code Output:

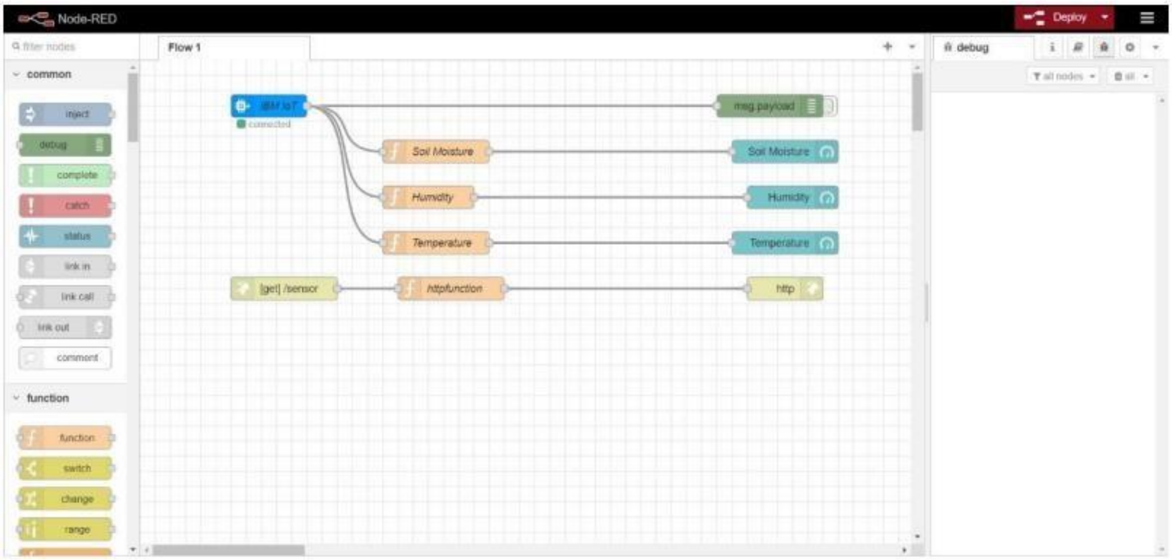
```
*Python 3.7.0 Shell*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\manoj-pt5890\Documents\python\project.py =========
2022-11-11 17:28:32,248   ibmiotf.device.Client      INFO    Connected successfully: d:nckdv7:NodeMCU:12345
Published Temperature = 89 C Humidity = 70 % Soil Moisture = 3 % to IBM Watson
Published Temperature = 78 C Humidity = 5 % Soil Moisture = 2 % to IBM Watson
Published Temperature = 85 C Humidity = 61 % Soil Moisture = 34 % to IBM Watson
Published Temperature = 75 C Humidity = 83 % Soil Moisture = 23 % to IBM Watson
Published Temperature = 72 C Humidity = 34 % Soil Moisture = 70 % to IBM Watson
Published Temperature = 38 C Humidity = 36 % Soil Moisture = 48 % to IBM Watson
Published Temperature = 62 C Humidity = 36 % Soil Moisture = 35 % to IBM Watson
Published Temperature = 34 C Humidity = 64 % Soil Moisture = 29 % to IBM Watson
Published Temperature = 95 C Humidity = 40 % Soil Moisture = 100 % to IBM Watson
Published Temperature = 47 C Humidity = 95 % Soil Moisture = 58 % to IBM Watson
```

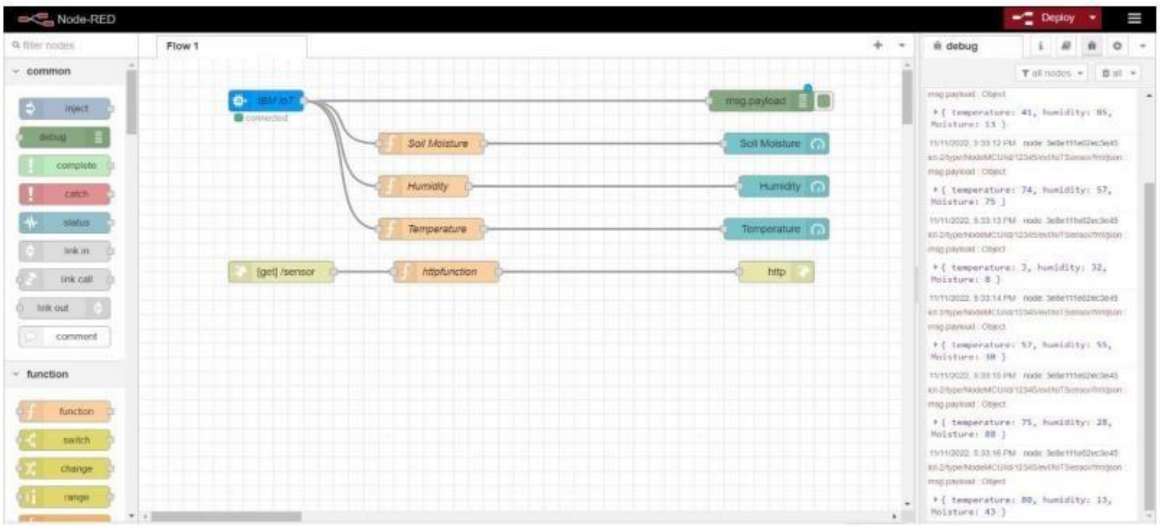
IBM Cloud after publishing data:



- **Creation of Node Red Service for device events:**

  In the IBM Watson IOT platform, under the catalog, under the Node Red app service, an application is deployed using cloud foundry. In the cloud foundry, a group has been created and using the ci pipeline, the app url is obtained. Using the URL, the Node red is launched. The IBM Watson IOT platform is connected to Node red using the IBM IoT palette. Using appropriate palettes, the data published in the IBM IoT platform is printed in the debug window of Node red.





Code block for the function palette:

1) **<u>Soil moisture</u>:**

   Soil = msg.payload.Moisture

```
msg.payload = "Soil Moisture : "
global.set('m',Soil)
msg.payload = Math.round(Soil)
return msg;
```

2) **Humidity:**

```
Humidity = msg.payload.humidity
msg.payload = "Humidity : "
global.set('h',Humidity)
msg.payload = Math.round(Humidity)
return msg;
```

3) **Temperature:**

```
Temperature = msg.payload.temperature
msg.payload = "Temperature : "
global.set('t',Temperature)
msg.payload =Math.round(Temperature)
return msg;
```

4) **HTTP Function:**

```
msg.payload = {"Temperature:": global.get('t'),"Humidity:":
global.get('h'),"Soil Moisture:": global.get('m')}
return msg;
```

- **Creation of Website dashboard:**

  A website dashboard has been created using the gauge palette. It can be accessed by adding “/ui” in the main url of Node red. This dashboard displays the gauge representation of the data published in the IBM IOT platform.

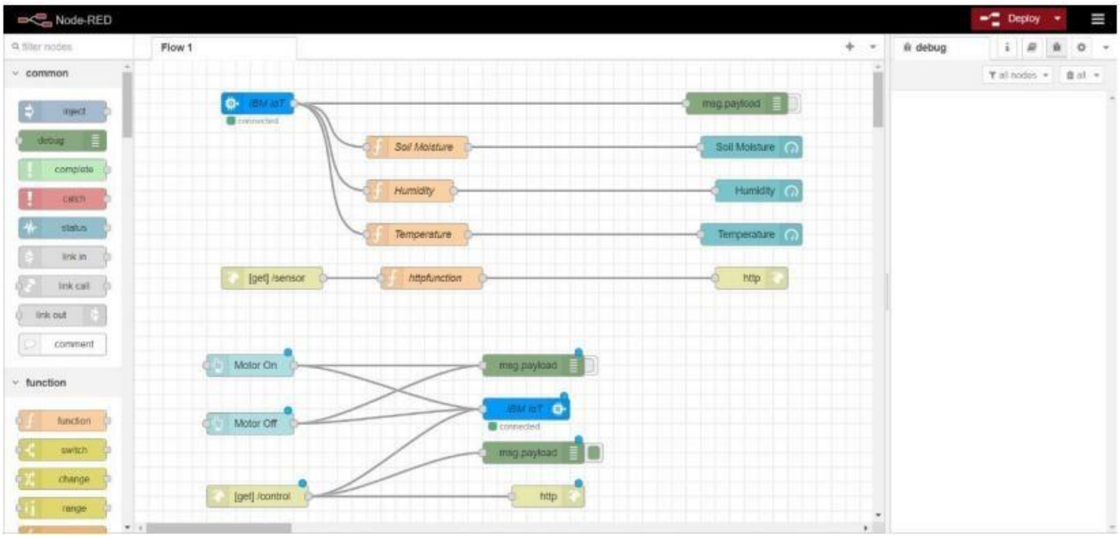
Python code used:

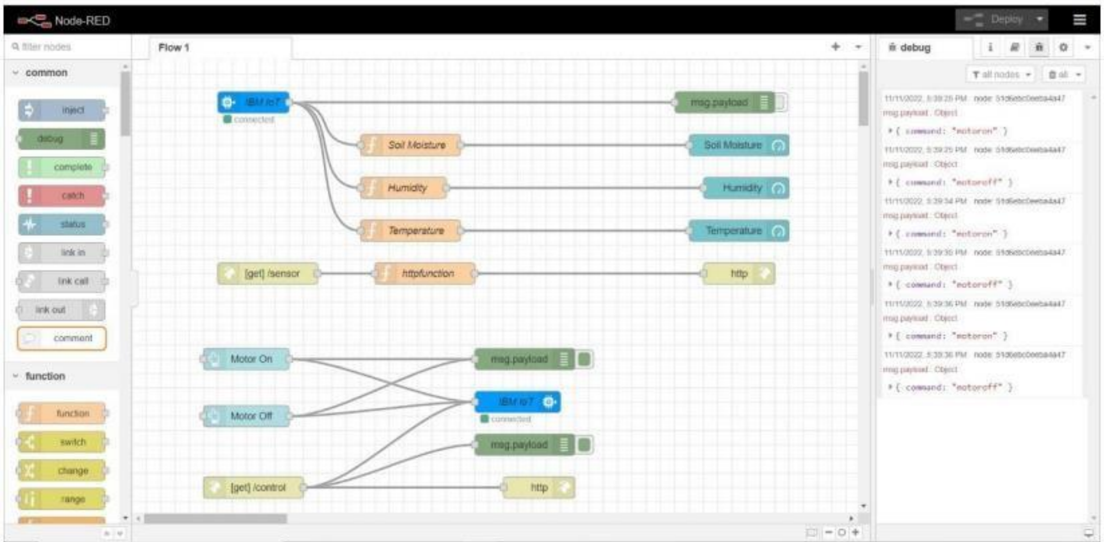
```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
#Provide your IBM Watson Device Credentials
organization = "nckdv7"
deviceType = "NodeMCU"
deviceId = "12345"
authMethod = "token"
authToken = "12345678" # Initialize GPIO
try:
        deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-token": authToken}
        deviceCli = ibmiotf.device.Client(deviceOptions)
        #..............................................
except Exception as e:
```

```
        print("Caught exception connecting device: %s" % str(e))
        sys.exit()
    # Connect and send a datapoint "hello" with value "world" into the
    cloud as an event of type "greeting" 10 times
    deviceCli.connect()
    while True: #Get Sensor Data from DHT11
        temp=random.randint(0,100)
        pulse=random.randint(0,100)
        moisture= random.randint(0,100)
        humidity=random.randint(0,100);
        lat = 17 lon = 18 data = { 'temperature' : temp, 'humidity' :
        humidity, 'Moisture' : moisture}
        #print data
    def myOnPublishCallback():
        print ("Published Temperature = %s C" % temp, "Humidity
        = %s %%" % humidity, "Soil Moisture = %s %%" %
        moisture,"to IBM Watson")
    success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0,
    on_publish=myOnPublishCallback)
    if not success:
        print("Not connected to IoTF")
    time.sleep(1)
    deviceCli.commandCallback = myCommandCallback
    # Disconnect the device and application from the cloud
    deviceCli.disconnect()
```

- **Creation of Node red service for device commands:**

  In addition to the palettes used in the Sprint-2, additional palettes such as buttons have been included to control devices by giving commands and the output is printed in the debug whenever a specific command is given.





Development of Python script to subscribe command from the IBM IOT platform:

Code:

```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
#Provide your IBM Watson Device Credentials
organization = "nckdv7"
deviceType = "NodeMCU"
deviceId = "12345"
authMethod = "token"
authToken = "12345678"
# Initialize GPIO
def myCommandCallback(cmd):
        print("Command received: %s" % cmd.data['command'])
        status=cmd.data['command']
        if status=="motoron":
                print("Motor is ON")
        else:
                print("Motor is OFF")
                #print(cmd)
try:

        deviceOptions = {"org": organization, "type": deviceType, "id":
        deviceId, "authmethod": authMethod, "auth-token": authToken}
        deviceCli = ibmiotf.device.Client(deviceOptions)
        #..............................................
except Exception as e:
        print("Caught exception connecting device: %s" % str(e))
        sys.exit()
```

```
# Connect and send a datapoint "hello" with value "world" into the cloud
as # an event of type "greeting" 10 times
deviceCli.connect()
while True:
        #Get Sensor Data from DHT11
        temp=random.randint(0,100)
        pulse=random.randint(0,100)
        moisture= random.randint(0,100)
        humidity=random.randint(0,100);
        lat = 17
        lon = 18
        data = { 'temperature' : temp, 'humidity' : humidity, 'Moisture' :
        moisture}
        #print data


def myOnPublishCallback():
        print ("Published Temperature = %s C" % temp, "Humidity = %s
        %%" % humidity, "Soil Moisture = %s %%" % moisture,"to IBM
        Watson")
 success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0,
 on_publish=myOnPublishCallback)
 if not success:
        print("Not connected to IoTF")
 time.sleep(1)
 deviceCli.commandCallback = myCommandCallback
# Disconnect the device and application from the cloud
deviceCli.disconnect()
```

Output:

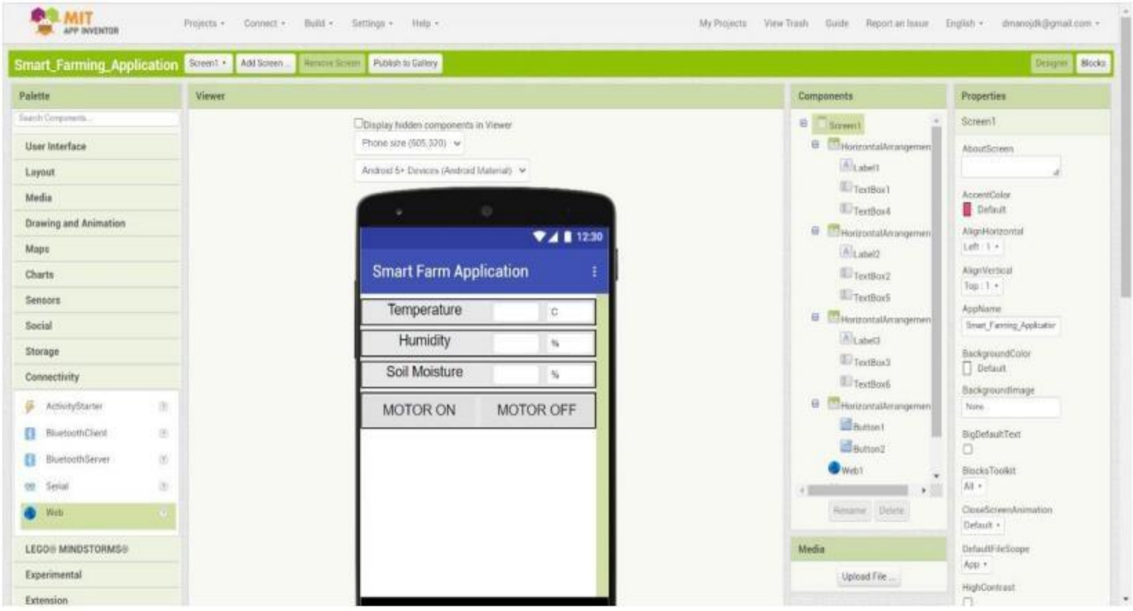
```
*Python 3.7.0 Shell*
File Edit Shell Debug Options Window Help
Published Temperature = 88 C Humidity = 66 % Soil Moisture = 3 % to IBM Watson
Published Temperature = 50 C Humidity = 97 % Soil Moisture = 63 % to IBM Watson
Published Temperature = 24 C Humidity = 33 % Soil Moisture = 50 % to IBM Watson
Published Temperature = 73 C Humidity = 29 % Soil Moisture = 56 % to IBM Watson
Published Temperature = 23 C Humidity = 1 % Soil Moisture = 90 % to IBM Watson
Published Temperature = 31 C Humidity = 12 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 91 C Humidity = 62 % Soil Moisture = 58 % to IBM Watson
Published Temperature = 15 C Humidity = 49 % Soil Moisture = 70 % to IBM Watson
Published Temperature = 51 C Humidity = 81 % Soil Moisture = 84 % to IBM Watson
Published Temperature = 61 C Humidity = 17 % Soil Moisture = 37 % to IBM Watson
Published Temperature = 91 C Humidity = 87 % Soil Moisture = 70 % to IBM Watson
Published Temperature = 35 C Humidity = 6 % Soil Moisture = 95 % to IBM Watson
Published Temperature = 52 C Humidity = 41 % Soil Moisture = 63 % to IBM Watson
Published Temperature = 40 C Humidity = 51 % Soil Moisture = 86 % to IBM Watson
Published Temperature = 33 C Humidity = 21 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 29 C Humidity = 48 % Soil Moisture = 22 % to IBM Watson
Published Temperature = 45 C Humidity = 32 % Soil Moisture = 23 % to IBM Watson
Published Temperature = 98 C Humidity = 38 % Soil Moisture = 8 % to IBM Watson
Published Temperature = 44 C Humidity = 71 % Soil Moisture = 16 % to IBM Watson
Command received: motoron
Motor is ON
Published Temperature = 62 C Humidity = 2 % Soil Moisture = 34 % to IBM Watson
Published Temperature = 21 C Humidity = 14 % Soil Moisture = 82 % to IBM Watson
Published Temperature = 35 C Humidity = 2 % Soil Moisture = 5 % to IBM Watson
Published Temperature = 34 C Humidity = 78 % Soil Moisture = 44 % to IBM Watson
Command received: motoroff
Motor is OFF
```

```
Published Temperature = 93 C Humidity = 81 % Soil Moisture = 87 % to IBM Watson
Command received: motoron
Motor is ON
Command received: motoroff
Motor is OFF
Published Temperature = 54 C Humidity = 36 % Soil Moisture = 81 % to IBM Watson
Published Temperature = 56 C Humidity = 76 % Soil Moisture = 56 % to IBM Watson
Published Temperature = 70 C Humidity = 53 % Soil Moisture = 74 % to IBM Watson
Published Temperature = 58 C Humidity = 22 % Soil Moisture = 68 % to IBM Watson
Command received: motoron
Motor is ON
Published Temperature = 93 C Humidity = 34 % Soil Moisture = 11 % to IBM Watson
Command received: motoroff
Motor is OFF
Published Temperature = 86 C Humidity = 67 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 49 C Humidity = 70 % Soil Moisture = 61 % to IBM Watson
Published Temperature = 94 C Humidity = 48 % Soil Moisture = 77 % to IBM Watson
Published Temperature = 59 C Humidity = 6 % Soil Moisture = 11 % to IBM Watson
Published Temperature = 16 C Humidity = 6 % Soil Moisture = 41 % to IBM Watson
```

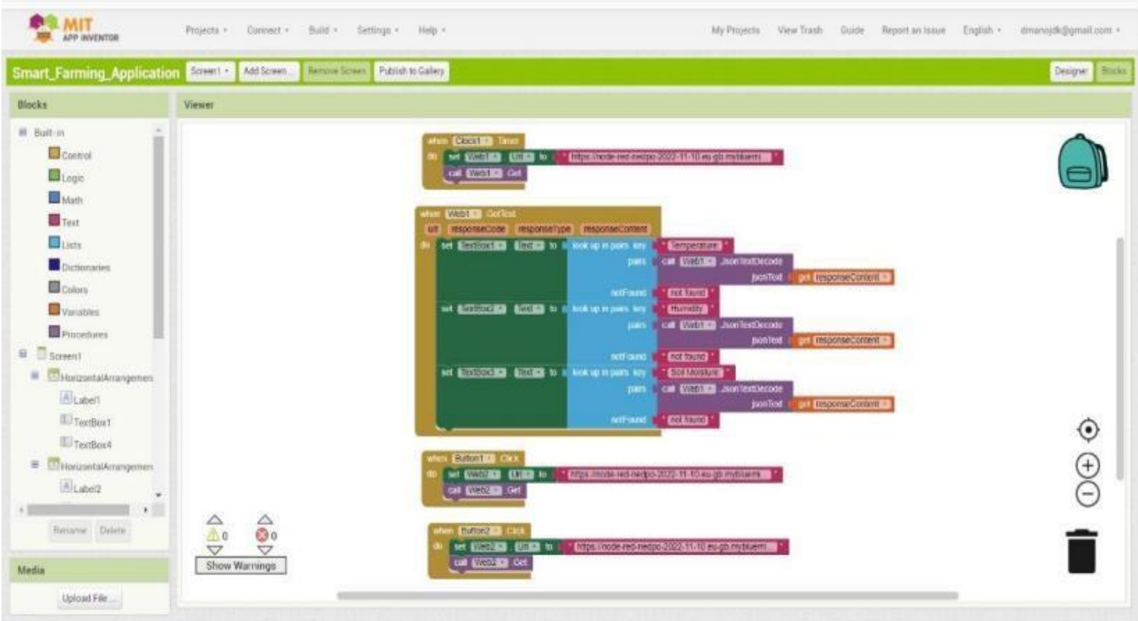
- **Development of Mobile application using MIT App Inventor:**

  In the MIT App Inventor platform, an application is created which monitors the farmland parameters such as temperature, humidity, soil moisture and controls the actuators such as motors.

MIT App Front End:



Backend:



App working:

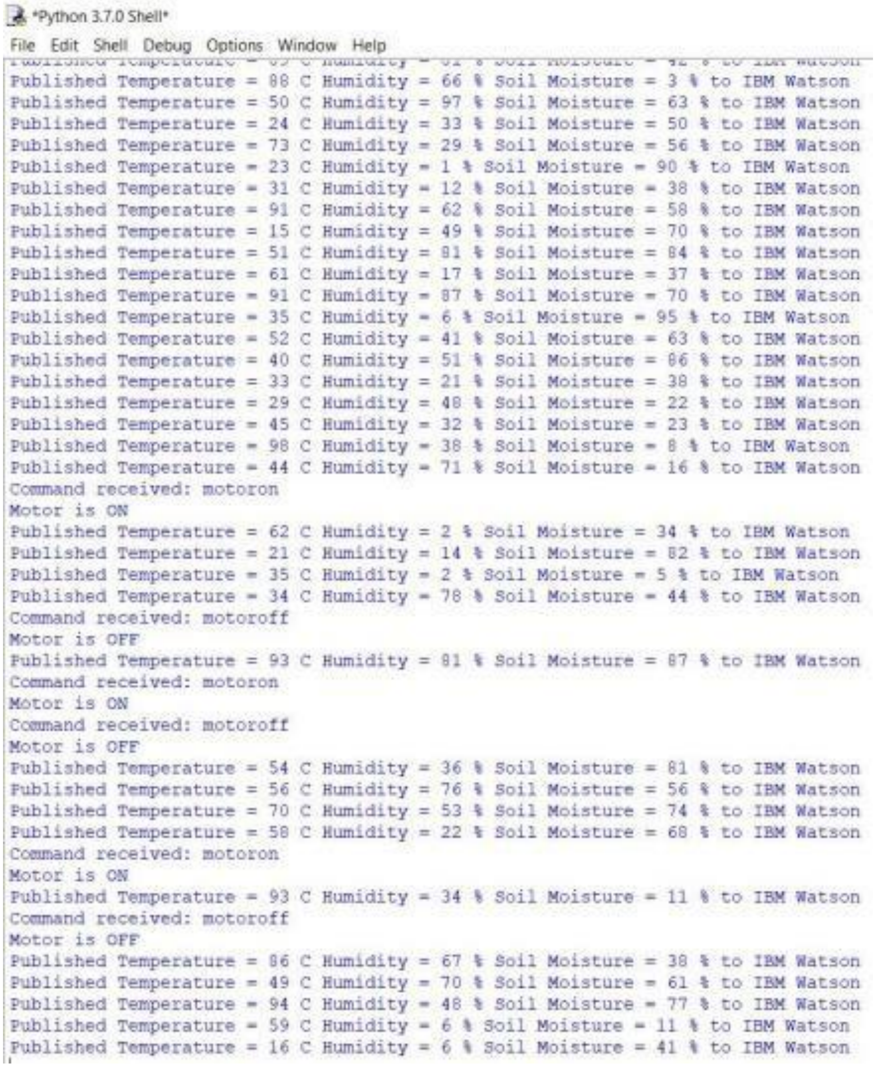
The app works based on HTTP protocol. The app uses HTTP GETmethod to parse the JSON data from the Node red website and displays the value in the UI. Using the HTTP POST method, the app sends command when a specific button is pressed. From where, the python codesubscribes the command data from the cloud thereby notifying the command is received.

Python code:

```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
#Provide your IBM Watson Device Credentials
organization = "nckdv7"
deviceType = "NodeMCU"
deviceId = "12345"
authMethod = "token"
authToken = "12345678"
# Initialize GPIO
def myCommandCallback(cmd):
        print("Command received: %s" % cmd.data['command'])
        status=cmd.data['command']
        if status=="motoron":
                print("Motor is ON")
        else:
                print("Motor is OFF")
                #print(cmd)

try:
        deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-token": authToken}
        deviceCli = ibmiotf.device.Client(deviceOptions)
        #..............................................
except Exception as e:
```

```
        print("Caught exception connecting device: %s" % str(e))
        sys.exit()
# Connect and send a datapoint "hello" with value "world" into the cloud
as # an event of type "greeting" 10 times
deviceCli.connect()
while True:
        #Get Sensor Data from DHT11
        temp=random.randint(0,100)
        pulse=random.randint(0,100)
        moisture= random.randint(0,100)
        humidity=random.randint(0,100);
        lat = 17
        lon = 18
        data = { 'temperature' : temp, 'humidity' : humidity, 'Moisture' :
        moisture}
        #print data
 def myOnPublishCallback():
        print ("Published Temperature = %s C" % temp, "Humidity = %s
         %%" % humidity, "Soil Moisture = %s %%" % moisture,"to IBM
        Watson")
success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0,
on_publish=myOnPublishCallback)
 if not success:
        print("Not connected to IoTF")
 time.sleep(1)
 deviceCli.commandCallback = myCommandCallback
# Disconnect the device and application from the cloud
deviceCli.disconnect()
```

Output:



```
*Python 3.7.0 Shell*
File  Edit  Shell  Debug  Options  Window  Help
Published Temperature = 88 C Humidity = 66 % Soil Moisture = 3 % to IBM Watson
Published Temperature = 50 C Humidity = 97 % Soil Moisture = 63 % to IBM Watson
Published Temperature = 24 C Humidity = 33 % Soil Moisture = 50 % to IBM Watson
Published Temperature = 73 C Humidity = 29 % Soil Moisture = 56 % to IBM Watson
Published Temperature = 23 C Humidity = 1 % Soil Moisture = 90 % to IBM Watson
Published Temperature = 31 C Humidity = 12 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 91 C Humidity = 62 % Soil Moisture = 58 % to IBM Watson
Published Temperature = 15 C Humidity = 49 % Soil Moisture = 70 % to IBM Watson
Published Temperature = 51 C Humidity = 81 % Soil Moisture = 84 % to IBM Watson
Published Temperature = 61 C Humidity = 17 % Soil Moisture = 37 % to IBM Watson
Published Temperature = 91 C Humidity = 87 % Soil Moisture = 70 % to IBM Watson
Published Temperature = 35 C Humidity = 6 % Soil Moisture = 95 % to IBM Watson
Published Temperature = 52 C Humidity = 41 % Soil Moisture = 63 % to IBM Watson
Published Temperature = 40 C Humidity = 51 % Soil Moisture = 86 % to IBM Watson
Published Temperature = 33 C Humidity = 21 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 29 C Humidity = 48 % Soil Moisture = 22 % to IBM Watson
Published Temperature = 45 C Humidity = 32 % Soil Moisture = 23 % to IBM Watson
Published Temperature = 98 C Humidity = 38 % Soil Moisture = 8 % to IBM Watson
Published Temperature = 44 C Humidity = 71 % Soil Moisture = 16 % to IBM Watson
Command received: motoron
Motor is ON
Published Temperature = 62 C Humidity = 2 % Soil Moisture = 34 % to IBM Watson
Published Temperature = 21 C Humidity = 14 % Soil Moisture = 82 % to IBM Watson
Published Temperature = 35 C Humidity = 2 % Soil Moisture = 5 % to IBM Watson
Published Temperature = 34 C Humidity = 78 % Soil Moisture = 44 % to IBM Watson
Command received: motoroff
Motor is OFF
Published Temperature = 93 C Humidity = 81 % Soil Moisture = 87 % to IBM Watson
Command received: motoron
Motor is ON
Command received: motoroff
Motor is OFF
Published Temperature = 54 C Humidity = 36 % Soil Moisture = 81 % to IBM Watson
Published Temperature = 56 C Humidity = 76 % Soil Moisture = 56 % to IBM Watson
Published Temperature = 70 C Humidity = 53 % Soil Moisture = 74 % to IBM Watson
Published Temperature = 58 C Humidity = 22 % Soil Moisture = 68 % to IBM Watson
Command received: motoron
Motor is ON
Published Temperature = 93 C Humidity = 34 % Soil Moisture = 11 % to IBM Watson
Command received: motoroff
Motor is OFF
Published Temperature = 86 C Humidity = 67 % Soil Moisture = 38 % to IBM Watson
Published Temperature = 49 C Humidity = 70 % Soil Moisture = 61 % to IBM Watson
Published Temperature = 94 C Humidity = 48 % Soil Moisture = 77 % to IBM Watson
Published Temperature = 59 C Humidity = 6 % Soil Moisture = 11 % to IBM Watson
Published Temperature = 16 C Humidity = 6 % Soil Moisture = 41 % to IBM Watson
```

CHAPTER 8 - PERFORMANCE METRICS

S. No.	Name of the Phase	Tasks Performed	Performance Metrics
1.	Development of Problem Statement	The underlying problem analyzed and a rough idea of the solution was planned	The Problem statement was developed
2.	Ideation Phase	Extracting use and test cases	Empathy map, Ideation and Literature survey were formulated.
3.	Project Design Phase 1	Solution for the problem is formulated and architecture is designed	Problem solution fit was designed and the Proposed solution is finalized with the help of Solution architecture.
4.	Project Design Phase 2	In depth analysis of the solution is performed including requirements, tech stack, etc.	Solution Requirements, Overall Technology stack, Data flow diagrams, User stories were formulated.
5.	Project Planning Phase	Various sprints were designed as individual progressive steps.	Project Milestone and Sprint Plans were developed.

CHAPTER 9 - ADVANTAGES AND DISADVANTAGES

9.1 Advantages:

- By monitoring the soil parameters of the farm, the user can have a complete analysis of the field, in terms of numbers.
- Using the website and the application, an interactive experiencecan be achieved.
- As the data gets pushed to the cloud, one can access the data anywhere from this world.
- Without human intervention, water pump can be controlled through the mobile application and it's flow can be customized using servo motors.
- By using Raspberry Pi MCU, scalability can be increased due to its high processing power and enough availability of GPIO pins

9.2 Disadvantages:

- Data transfer is through the internet. So data fetch and push might delay due to slow internet connection, depending on the location and other physical parameters.
- System can only monitor a certain area of the field. In order to sense and monitor an entire field, sensors should be placed in manyplaces, which may increase the cost.
- Data accuracy may vary according to various physical parameters such as temperature, pressure, rain.
- Cost of the system is high due to usage of Raspberry Pi.
- Rodent and insects may cause damage to the system.

CHAPTER 10 – CONCLUSION

The project thus monitors important parameters present in the field such as temperature, humidity, soil moisture etc., and controls important actuators such as motors etc. It is helpful for farmers to remotely monitor their fields even during adverse weather conditions and help them control farming equipments remotely using cloud.

CHAPTER 11 - FUTURE SCOPE

The project can be further extended by monitoring other parameters such as nutrient contents in the soil, soil texture etc. AI techniques integrated with cloud can be integrated to monitor any pest attacks present in the plant. The application can be made interactive which provides suggestions to farmers to improve their farmlands.

CHAPTER 12 – APPENDIX

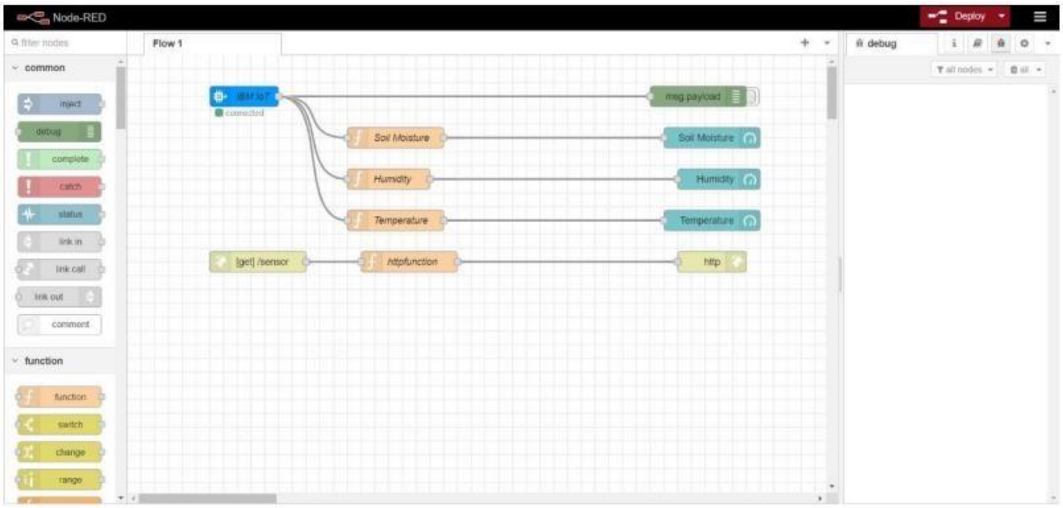
12.1 Source Code:

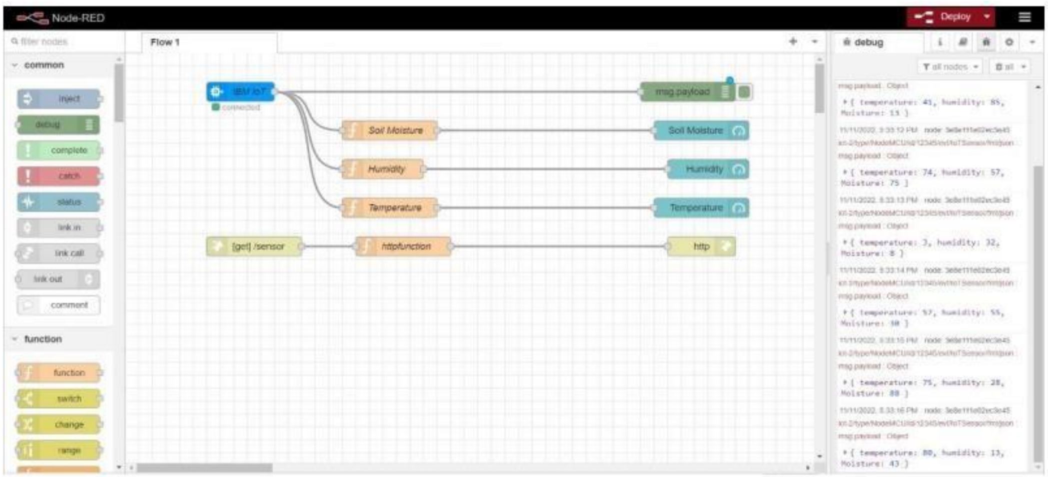
```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random

#Provide your IBM Watson Device Credentials
organization = "nckdv7"
deviceType = "NodeMCU"
deviceId = "12345"
authMethod = "token"
authToken = "12345678"

# Initialize GPIO
def myCommandCallback(cmd):
    print("Command received: %s" % cmd.data['command'])
    status=cmd.data['command']
    if status=="motoron":
        print("Motor is ON")
    else:
        print("Motor is OFF")
    #print(cmd)
try:
        deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-token": authToken}
```

```
        deviceCli = ibmiotf.device.Client(deviceOptions)
        #..............................................

except Exception as e:
        print("Caught exception connecting device: %s" % str(e))
        sys.exit()

# Connect and send a datapoint "hello" with value "world" into the cloud as an event of type "greeting" 10 times
deviceCli.connect()

while True:
      #Get Sensor Data from DHT11
      temp=random.randint(0,100)
      pulse=random.randint(0,100)
      moisture= random.randint(0,100)
      humidity=random.randint(0,100);
      lat = 17
      lon = 18

      data = { 'temp' : temp, 'humidity' : humidity, 'Soil Moisture' : moisture}
      #print data
      def myOnPublishCallback():
         print ("Published Temperature = %s C" % temp, "Humidity = %s %%" % humidity, "Soil Moisture = %s %%" % moisture,"to IBM Watson")
```

```
        success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0, on_publish=myOnPublishCallback)
        if not success:
            print("Not connected to IoTF")
        time.sleep(1)

        deviceCli.commandCallback = myCommandCallback

# Disconnect the device and application from the cloud
deviceCli.disconnect()
```

Node Red Service Creation:





Code block for the function palette:

1) **<u>Soil moisture</u>:**

```
Soil = msg.payload.Moisture
msg.payload = "Soil Moisture : "
global.set('m',Soil)
msg.payload = Math.round(Soil)
return msg;
```

2) **<u>Humidity:</u>**

```
Humidity = msg.payload.humidity
msg.payload = "Humidity : "
global.set('h',Humidity)
msg.payload = Math.round(Humidity )
return msg;
```

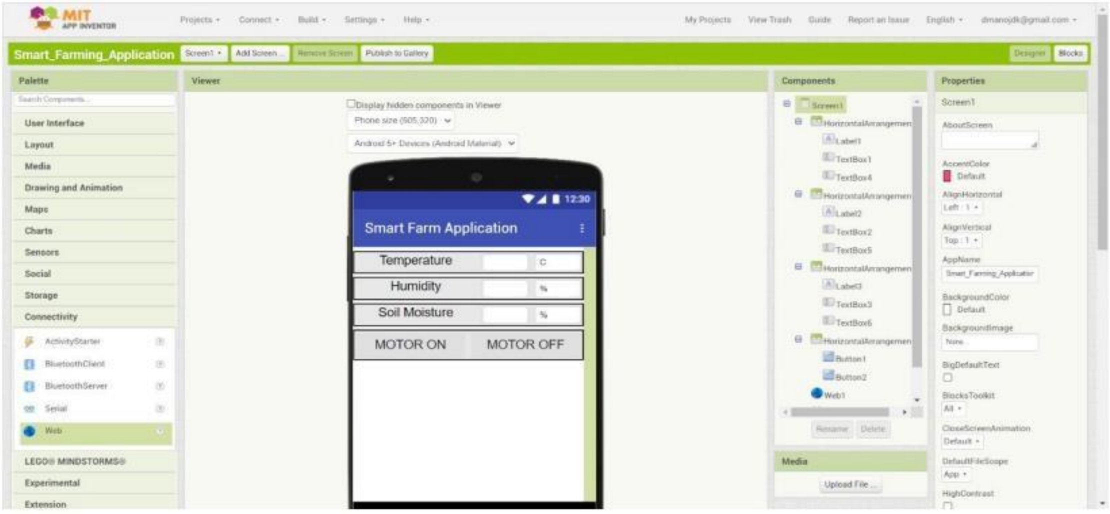
3) **<u>Temperature:</u>**

```
Temperature = msg.payload.temperature
msg.payload = "Temperature : "
global.set('t',Temperature)
msg.payload =Math.round(Temperature)
return msg;
```

4) **<u>HTTP Function:</u>**

```
msg.payload = {"Temperature:": global.get('t'),"Humidity:":
global.get('h'),"Soil Moisture:": global.get('m')}
return msg;
```

MIT App Front End:



Backend:

12.2 GitHub and Project Demo Link:

GitHub:

https://github.com/IBM-EPBL/IBM-Project-53011-1661252734

Project Demo Link:

https://drive.google.com/file/d/1tEm_30mItVxo7ZZouRHgvg0M6n6aLfF3/view?usp=share_link