PROJECT REPORT

IOT BASED SMART CROP PROTECTION SYSTEM FOR AGRICULTURE

TEAM ID: PNT2022TMID24163

TEAM LEAD: YOKESH R

TEAM MEMBER 1: VINOTH B

TEAM MEMBER 2: SARAN N

TEAM MEMBER 3: AJAYKUMAR K P

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROJECT OVERVIEW

This is a Smart Agriculture System project based on Internet Of Things (IoT), that can measure soil moisture and temperature conditions for agriculture using Watson IoT services. IoT is network that connects physical objects or things embedded with electronics, software and sensors through network connectivity that collects and transfers data using cloud for communication. Data is transferred through internet without human to human or human to computer interaction.

In this project we have not used any hardware. Instead of real soil and temperature conditions, sensors IBM IoT Simulator is used which can transmit soil moisture temperature as required.

- ➢ **Project requirements**: Node-RED, IBM Cloud, IBM Watson IoT, Node.js, IBM Device, IBM IoT Simulator, Python 3.7, Open Weather API platform.
- ➢ **Project Deliverables**: Application for IoT based Smart Agriculture System

1.2 PURPOSE

A vast majority of the people are invariably affected by the production of crops. Farmers, for example, rely on them for their survival. The consumers, on the other hand, depend on the crops as it provides them with a multitude of utilities. It therefore, becomes essential to protect and maintain these crops. The project aims at improving the farmers's situation by preventing them from incurring losses due to the damage of crops. Crop failure also deteriorates the quality of the yield thereby decreasing the quality of living.

LITERATURE SURVEY

SUMMARY

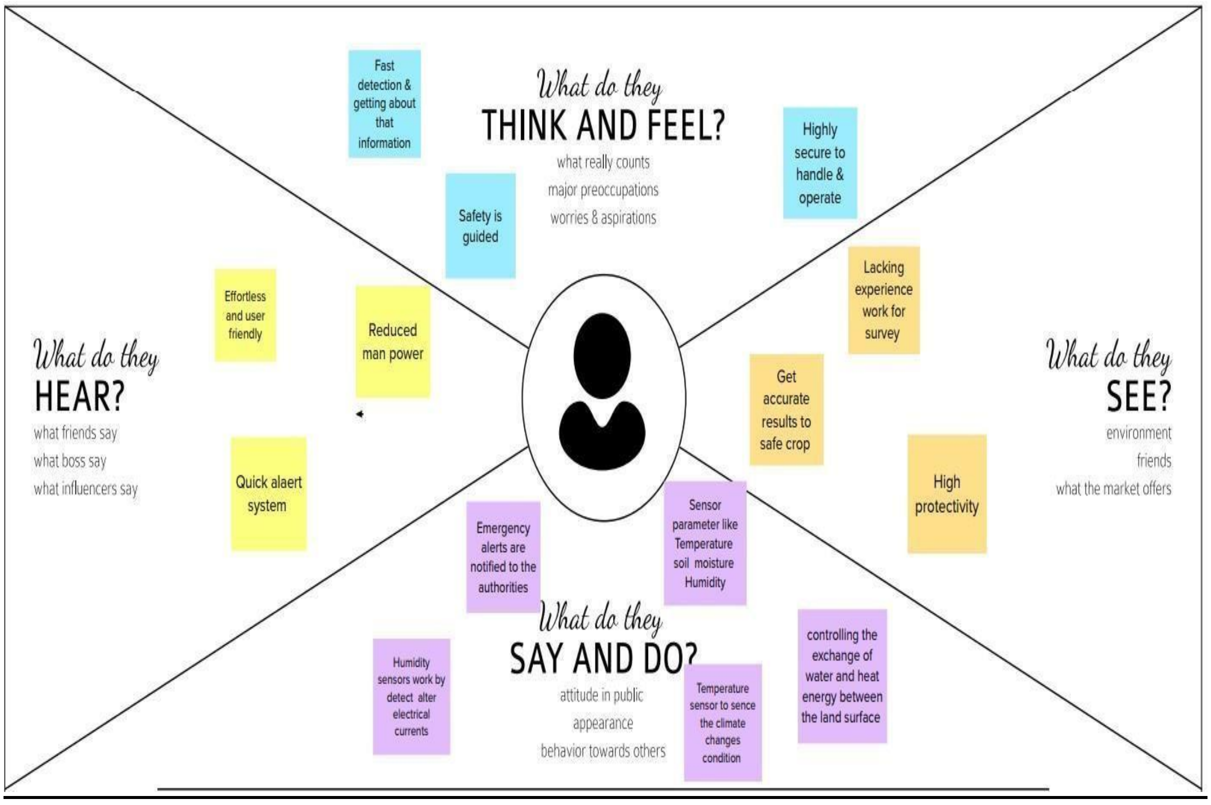
Using several PIR sensors can also prove to be efficient identifying the location of the intrusion and usage of multiple PIR sensors can be used to find the height of the animal and classify the seriousness of intrusion. Moisture control is another aspect where people generally misjudge the effectiveness. So actively, monitoring and automating the process of controlling moisture level will prove helpful. For cloud database in this case the information that needed to be stored in the cloud are in strings and integers for which google sheets database is more than sufficient. Choosing online based weather API proves to be more efficient than that of local sensors for the following since satellite weather data is almost as accurate as local offline sensor outputs.

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative visualization used to express clearly what one knows about a particular type of user. It externalizes knowledge about users in order to create a shared understanding of user needs, and aid in decision making.

Empathy maps are split into 4 quadrants (Says, Thinks, Does, and Feels), with the user in the middle. Empathy maps provide a glance into who a user is as a whole.The ***Says*** quadrant contains what the user says or what he needs. The ***Thinks*** quadrant captures what the user is thinking throughout the experience. The ***Does*** quadrant encloses the actions the user takes. The ***Feels*** quadrant is the user’s emotional state.


Fast detection & getting about that information
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
Highly secure to handle & operate
Safety is guided
Lacking experience work for survey
Effortless and user friendly
Reduced man power
What do they HEAR?
what friends say
what boss say
what influencers say
Get accurate results to safe crop
What do they SEE?
environment
friends
what the market offers
Quick alaert system
High protectivity
Emergency alerts are notified to the authorities
Sensor parameter like Temperature soil moisture Humidity
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
Humidity sensors work by detect alter electrical currents
Temperature sensor to sence the climate changes condition
controlling the exchange of water and heat energy between the land surface

3.2 IDEATION AND BRAINSTORMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity. Both brainstorming and ideation are processes invented to create new valuable ideas, perspectives, concepts and insights, and both are methods for envisioning new frameworks and systemic problem solving.

The Ideation chart for Industry Specific Intelligent Fire Management System is shown in Table Below.

IDEA 1	IDEA 2	IDEA 3
Crop protection from animals using IR motion detectors. The farmland is surrounded by fences and each fence is equipped with multiple IR motion detectors in various heights. Location of each motion detector is surveyed and stored in the database.	A user interface system for farmers to analyze the data. The data to the system are sensor data from Humidity sensor, Temperature sensor, PIR Sensor and they are processed using a microcontroller and stored in a database. This database also gives an overview on crop yields, profit and losses for the farmer,	Crop protection from environmental factors such as UV rays, temperature, humidity, moisture content in soil. Using color sensors to detect NPK values of the soil and determining its fertility this data can be used to determine what type of fertilizers to be used. These factors can play a major role in crop protection and crop yield.

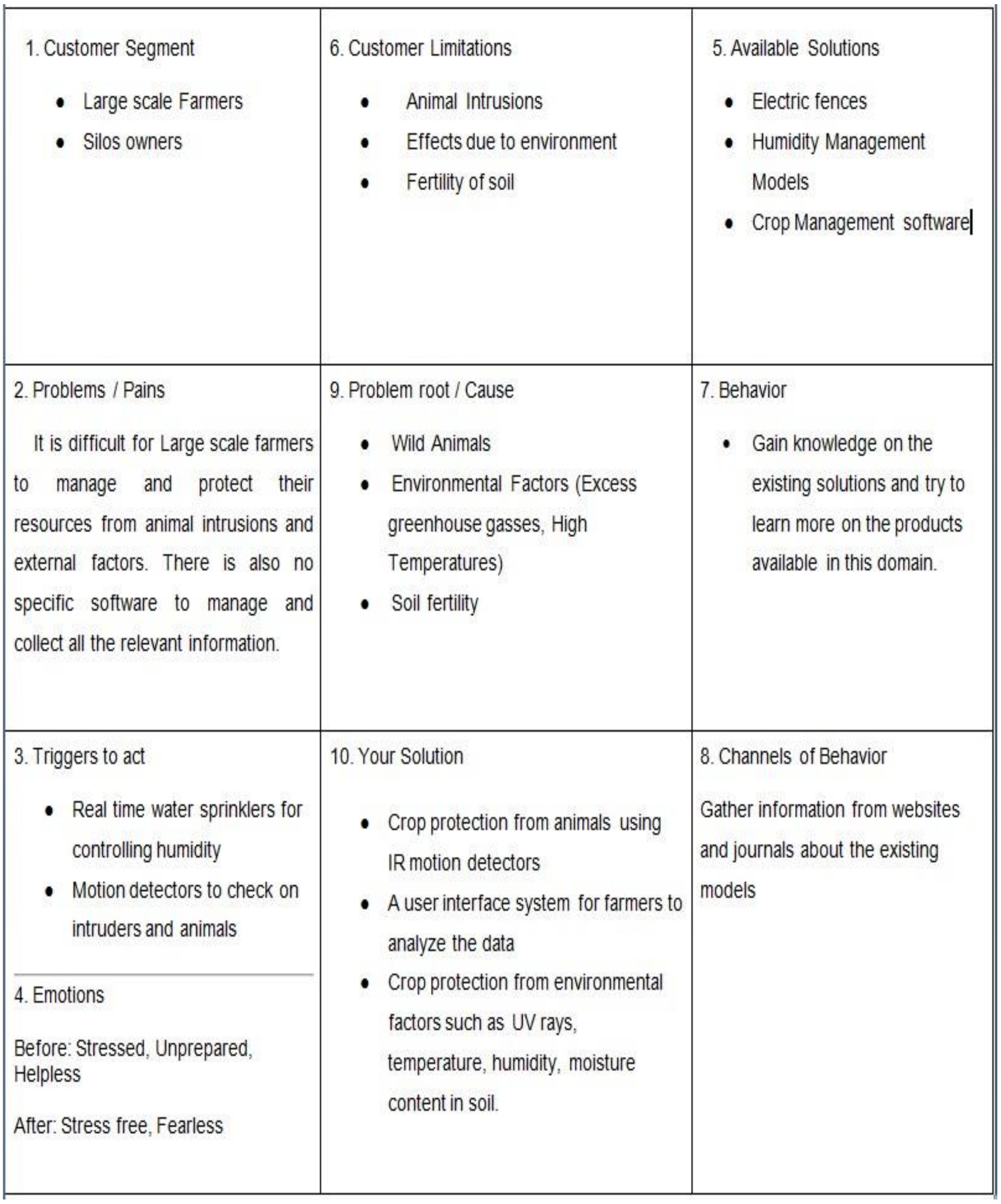
Cameras are placed in suitable locations so that we get a complete view over the farmland. When an animal or the intruder enters the field. The IR detectors which are placed in various heights are used to detect the type of the animal which has entered the field and the size of the animal. Alarms can be used to alert when large animals enter the field. And the camera is activated when the IR sensor detects motion. Then the picture is sent to the farmer.	what crop has been sowed and Expenses. This database can be used in the future for analyzing a pattern for best yields, to minimize the expenses and help the farmer take decisions financially.	So having control over these will help to improve the yield. Sensors for UV concentration, Moisture content, temperature are measured and water sprinklers are used to control the parameters accordingly.

3.3 PROPOSED SOLUTION

The proposed solution for IOT Based Smart crop protection system for Agriculture is shown Below.

S.No.	Parameter	Description
1.	Problem Statement	Develop an efficient system & an application that can monitor and alert the users(farmers)
2.	Idea/Solution description	➢ This product helps the field in monitoring the animals other disturbance ➢ In several areas, the temperature sensors will be integrated to monitor the temperature & humidity ➢ If in any area feel dry is detected by admins, will be notified along with the location in the web application
3.	Novelty/Uniqueness	➢ Fastest alerts to the farmers ➢ The increasing demand for quality food ➢ User friendly
4.	Social Impact/Customer Satisfaction	➢ Easy installation and provide efficient results ➢ Can work with irrespective of fear
5.	Business Model(Revenue Model)	➢ As the product usage can be understood by everyone, it is easy for them to use it properly for their safest organization ➢ The product is advertised all over the platforms. Since it is economical, even helps small scale farming land from disasters.
6.	Scalability of the Solution	➢ Even when the interruption is more, the product sense the accurate location and alerts the farmers effectively

3.4 PROBLEM SOLUTION FIT



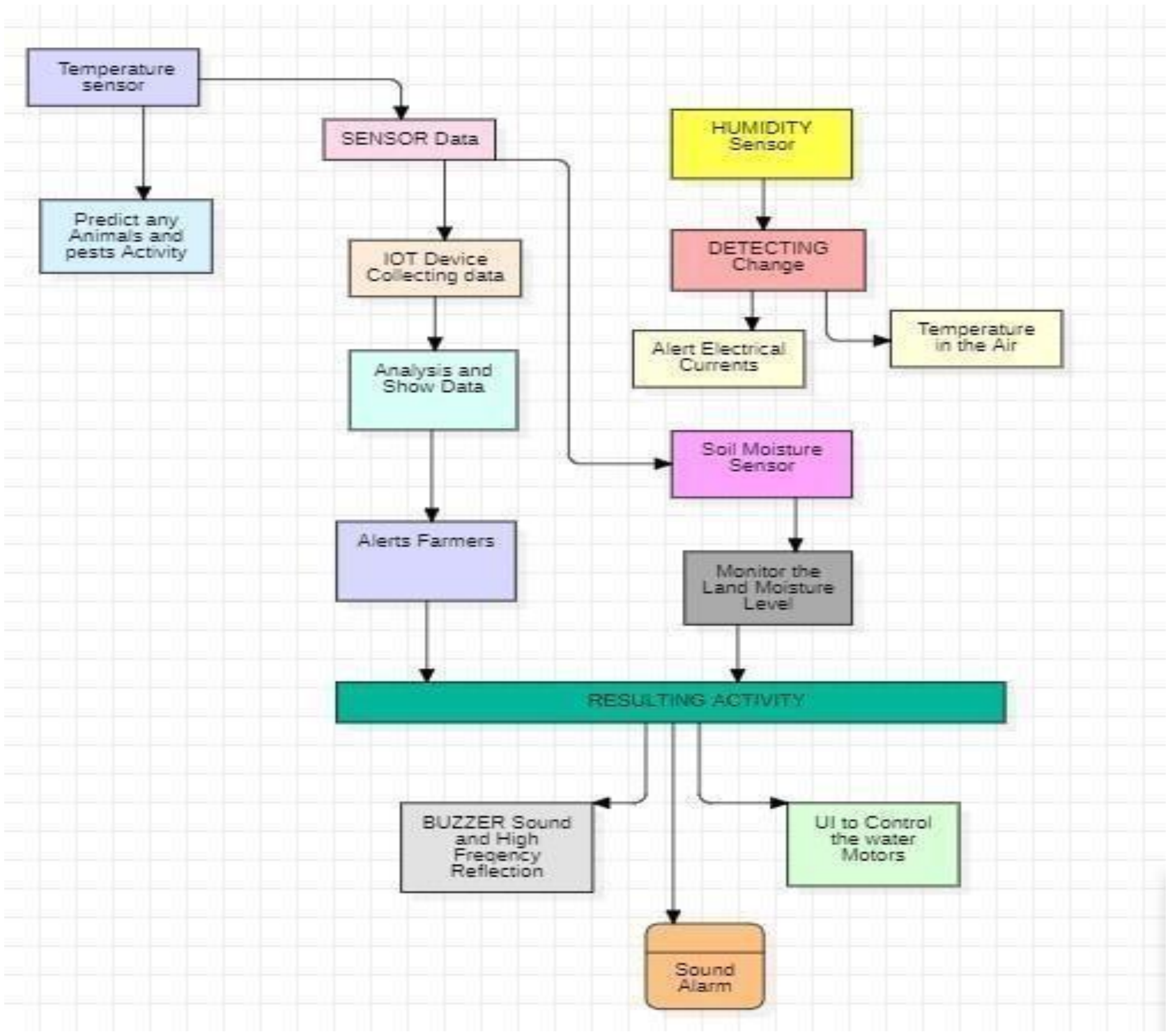
1. Customer Segment • Large scale Farmers • Silos owners	6. Customer Limitations • Animal Intrusions • Effects due to environment • Fertility of soil	5. Available Solutions • Electric fences • Humidity Management Models • Crop Management software
2. Problems / Pains It is difficult for Large scale farmers to manage and protect their resources from animal intrusions and external factors. There is also no specific software to manage and collect all the relevant information.	9. Problem root / Cause • Wild Animals • Environmental Factors (Excess greenhouse gasses, High Temperatures) • Soil fertility	7. Behavior • Gain knowledge on the existing solutions and try to learn more on the products available in this domain.
3. Triggers to act • Real time water sprinklers for controlling humidity • Motion detectors to check on intruders and animals 4. Emotions Before: Stressed, Unprepared, Helpless After: Stress free, Fearless	10. Your Solution • Crop protection from animals using IR motion detectors • A user interface system for farmers to analyze the data • Crop protection from environmental factors such as UV rays, temperature, humidity, moisture content in soil.	8. Channels of Behavior Gather information from websites and journals about the existing models

REQUIREMENT ANALYSIS

PROJECT DESIGN

5.1 DATA FLOW DIAGRAM

The data flow diagram for IOT based smart crop protection system using for agriculture is shown below.

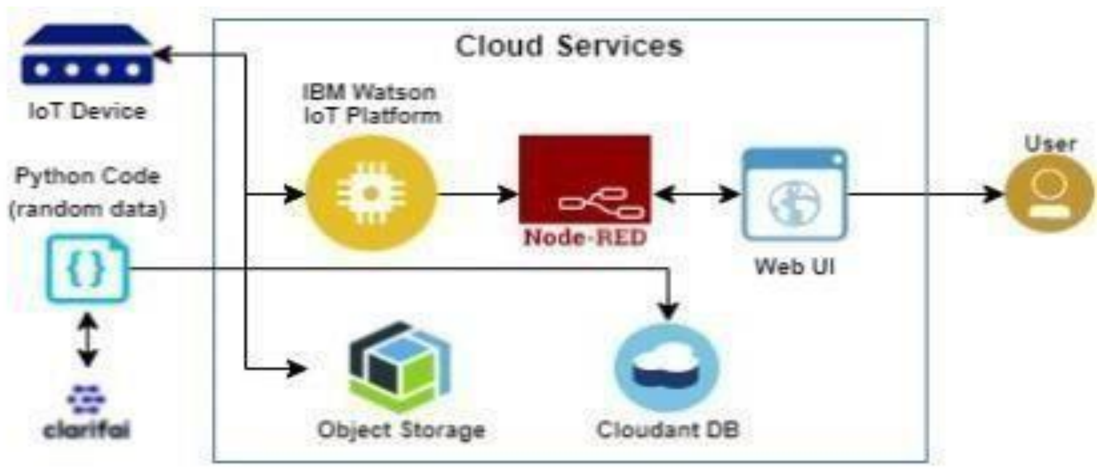

Temperature sensor
SENSOR Data
HUMIDITY Sensor
Predict any Animals and pests Activity
IOT Device Collecting data
DETECTING Change
Alert Electrical Currents
Temperature in the Air
Analysis and Show Data
Soil Moisture Sensor
Alerts Farmers
Monitor the Land Moisture Level
RESULTING ACTIVITY
BUZZER Sound and High Freqency Reflection
UI to Control the water Motors
Sound Alarm

5.2 SOLUTION AND TECHNICAL ARCHITECHTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

• Find the best tech solution to solve existing business problems.

• Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
• Define features, development phases, and solution requirements.
• Provide specifications according to which the solution is defined, managed, and delivered. The below figure shows the solution architecture of IoT based smart crop protection system.



PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

The below Table shows the sprint planning and estimation of IoT Based Smart Crop Protection System.

Sprint	Functional Requirement (Epic)	User Number Story	User Story/Task	Story Points	Priority
Sprint-1	Registration	USN-1	I can create account in IBM cloud and the data are collected.	20	High
Sprint-2	Analyze	USN-2	All the data that are collected is cleaned and uploaded in the database or IBM cloud.	20	Medium
Sprint-3	Dashboard	USN-3	I can use my account in my dashboard for uploading dataset.	10	Medium
Sprint-3	Visualization	USN-4	I can prepare data for Visualization.	10	High
Sprint-4	Visualization	USN-5	I can present data in my dashboard.	10	High
Sprint-4	Prediction	USN-6	We can Protect the crops from the animals.	10	High

6.2.SPRINT DELIVERY SCHEDULE

The sprint delivery plan is scheduled accordingly as shown in the below table 6.2 which consists of the sprints with respective to their duration, sprint start and end date and the releasing data.

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

CODING AND SOLUTIONING FEATURE 1:

```
import random import
ibmiotf.application
import  ibmiotf.device
from time import sleep
import sys
#IBM Watson Device Credentials.  organization =
"op701j" deviceType = "Lokesh" deviceId =
"Lokesh89" authMethod = "token" authToken =
"1223334444"      def
            myCommandCallback(cmd):
print("Command  received: %s"          %
cmd.data['command'])
status=cmd.data['command'] if
status=="sprinkler_on":
  print ("sprinkler is ON")
else :  print ("sprinkler
  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()
#Connecting to IBM
watson.
deviceCli.connect() while
True:
#Getting values from sensors.
temp_sensor = round( random.uniform(0,80),2) PH_sensor =
round(random.uniform(1,14),3) camera = ["Detected","Not Detected","Not
Detected","Not Detected","Not Detected","Not Detected",] camera_reading =
random.choice(camera) flame = ["Detected","Not Detected","Not Detected","Not
Detected","Not Detected","Not Detected",] flame_reading = random.choice(flame)
moist_level = round(random.uniform(0,100),2) water_level =
round(random.uniform(0,30),2)  #storing the sensor data to send in json format to cloud.
temp_data = { 'Temperature' : temp_sensor }  PH_data = { 'PH Level' : PH_sensor }
camera_data = { 'Animal attack' : camera_reading} flame_data = { 'Flame' : flame_reading
} moist_data = { 'Moisture Level' : moist_level} water_data = {
'Water Level' : water_level}

# publishing Sensor data to IBM Watson for every 5-10 seconds.
success = deviceCli.publishEvent("Temperature sensor", "json", temp_data,
qos=0) sleep(1) if success:
  print (" ...........................publish ok ....................................... ")
print ("Published Temperature = %s C" % temp_sensor, "to IBM Watson")
```

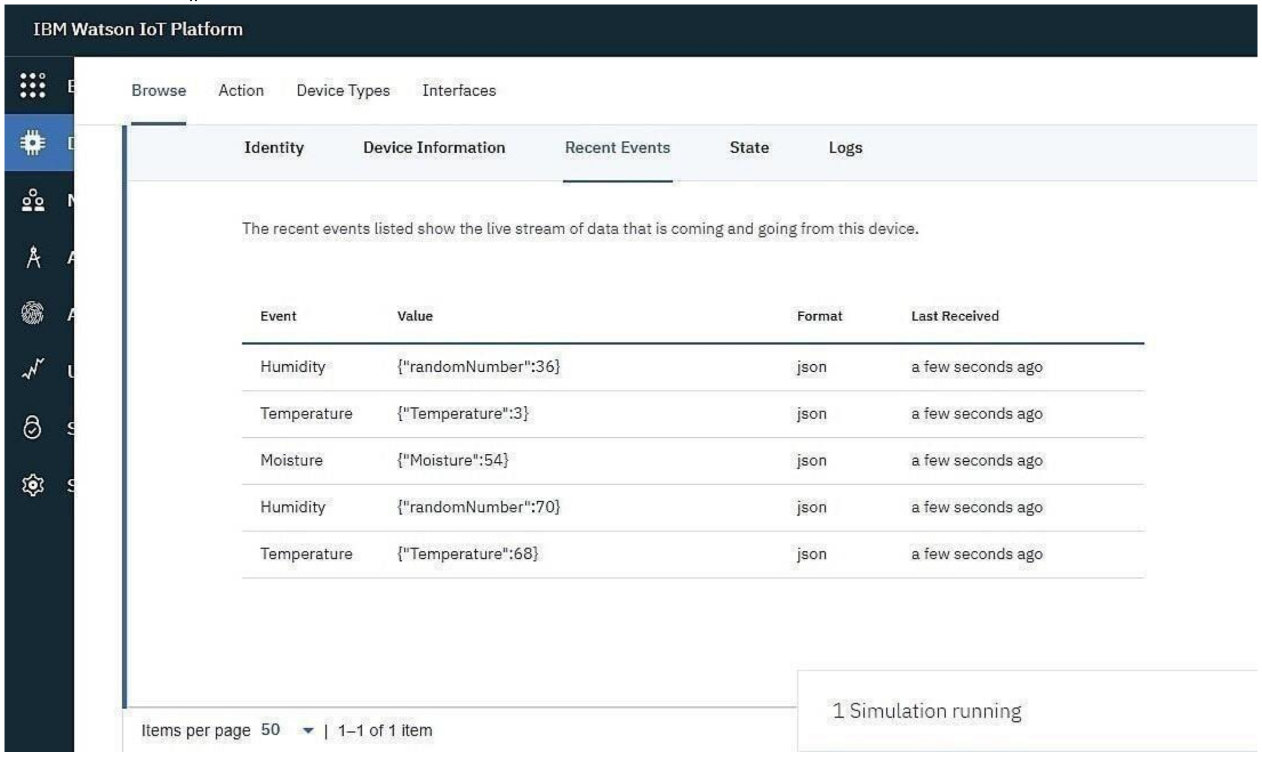
```
success = deviceCli.publishEvent("PH sensor", "json", PH_data,
qos=0) sleep(1) if success:
  print ("Published PH Level = %s" % PH_sensor, "to IBM Watson")

success = deviceCli.publishEvent("camera", "json", camera_data, qos=0)
sleep(1) if success:
  print ("Published Animal attack %s " % camera_reading, "to IBM
Watson") success = deviceCli.publishEvent("Flame sensor", "json",
flame_data, qos=0) sleep(1) if success:
  print ("Published Flame %s " % flame_reading, "to IBM Watson")

success = deviceCli.publishEvent("Moisture sensor", "json", moist_data,
qos=0) sleep(1) if success:  print ("Published Moisture Level = %s "
   % moist_level, "to IBM Watson")
success = deviceCli.publishEvent("Water sensor", "json",
water_data, qos=0) sleep(1) if success:  print ("Published Water
Level = %s cm" % water_level, "to IBM Watson")
print ("") #Automation to control sprinklers by present temperature an to send alert
message to IBM Watson.

if (temp_sensor > 35):
  print("sprinkler-1 is ON")
success = deviceCli.publishEvent("Alert1", "json",{ 'alert1' : "Temperature(%s) is high, sprinkerlers are turned
ON" %temp_sensor }
,

qos=0)
sleep(1)
if
success:
  print( 'Published alert1 : ', "Temperature(%s) is high, sprinkerlers are turned ON" %temp_sensor,"to IBM
  Watson")
print("")
else: print("sprinkler-1 is
OFF") print("")

#To send alert message if farmer uses the unsafe fertilizer to crops. if

(PH_sensor > 7.5 or PH_sensor < 5.5): success = deviceCli.publishEvent("Alert2", "json",{ 'alert2' : "Fertilizer PH
  level(%s) is not safe,use other fertilizer" %PH_sensor } ,
qos=0)
sleep(1) if
success:
  print('Published alert2 : ' , "Fertilizer PH level(%s) is not safe,use other fertilizer" %PH_sensor,"to IBM
  Watson")
print("")

#To send alert message to farmer that animal attack on

crops.  if (camera_reading == "Detected"):

  success = deviceCli.publishEvent("Alert3", "json", { 'alert3' : "Animal attack on crops detected" }, qos=0)
sleep(1) if
```

success: print('Published alert3 : ' , "Animal attack on crops detected","to IBM
  Watson","to IBM Watson")

print("") #To send alert message if flame detected on crop land and turn ON the splinkers to take
immediate action.

if (flame_reading == "Detected"):
  print("sprinkler-2 is ON")
success = deviceCli.publishEvent("Alert4", "json", { 'alert4' : "Flame is detected crops are in danger,sprinklers turned ON" },
qos=0) sleep(1) if success: print('Published alert4 : ' , "Flame is detected crops are in
  danger,sprinklers turned ON","to IBM Watson")

#To send alert message if Moisture level is LOW and to Turn ON Motor-1 for
irrigation. if (moist_level < 20):
  print("Motor-1 is ON")
success = deviceCli.publishEvent("Alert5", "json", { 'alert5' : "Moisture level(%s) is low, Irrigation started" %moist_level },
qos=0) sleep(1) if success:
  print('Published alert5 : ' , "Moisture level(%s) is low, Irrigation started" %moist_level,"to IBM Watson")
print("")
#To send alert message if Water level is HIGH and to Turn ON Motor-2 to take
water out. if (water_level > 20): print("Motor-2 is ON")
success = deviceCli.publishEvent("Alert6", "json", { 'alert6' : "Water level(%s) is high, so motor is ON to take water out " %water_level }, qos=0) sleep(1) if success: print('Published alert6 : ' , "water level(%s) is high, so motor is ON to take water out " %water_level,"to IBM Watson") print("")
#command recived by farmer
deviceCli.commandCallback =
myCommandCallback # Disconnect the device
and application from the cloud
deviceCli.disconnect()



The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
Humidity	{"randomNumber":36}	json	a few seconds ago
Temperature	{"Temperature":3}	json	a few seconds ago
Moisture	{"Moisture":54}	json	a few seconds ago
Humidity	{"randomNumber":70}	json	a few seconds ago
Temperature	{"Temperature":68}	json	a few seconds ago

Items per page 50 | 1–1 of 1 item

1 Simulation running

Features

Output: Digital pulse high (3V) when triggered (motion detected) digital low when idle (no motion detected). Pulse lengths are determined by resistors and capacitors on the PCB and differ from sensor to sensor. Power supply: 5V-12V input voltage for most modules (they have a 3.3V regulator),but 5V is ideal in case the regulator has different specs.

BUZZER

Specifications

- RatedVoltage : 6V DC
- Operating Voltage : 4 to 8V DC
- Rated Current*: ≤30mA
- SoundOutput at 10cm* : ≥85dB
- Resonant Frequency : 2300 ±300Hz
- Tone: Continuous A buzzer is a loud noise maker.

Most modern ones are civil defense or air- raid sirens, tornado sirens, or the sirens on emergency service vehiclessuch as ambulances, police cars and fire trucks. There are two general types, pneumatic and electronic.

OTHER FEATURES:

i. Goodsensitivity to Combustible gas in wide range .

ii. Highsensitivity to LPG, Propane and Hydrogen .

iii. Longlife and low cost.

iv. Simpledrive circuit.

TESTING AND RESULTS

8.1 TEST CASES

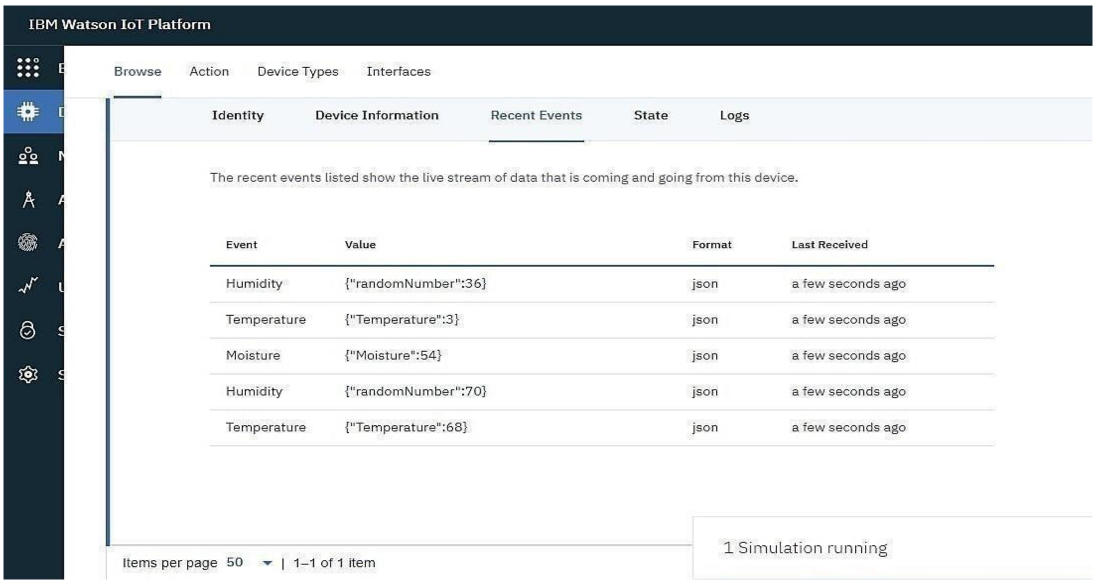
if the temperature is high which means the message should be sent to the farmer.. Typical time taken for the message to reach the user after the detection of intrusion is 30 to 40 seconds, so the response time is average around 35 seconds.

sno	parameter	Values
1	Model summary	-
2	accuracy	Training accuracy- 95% Validation accuracy- 72%
3	Confidence score	Class detected- 80% Confidence score-80%

PERFORMANCE RESULTS:

9.1 PERFORMANCE METRICES

It updates the values of Humidity, Temperature, Moisture, Humidity and these values will be automatically updated for certain period of times.



The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
Humidity	{"randomNumber":36}	json	a few seconds ago
Temperature	{"Temperature":3}	json	a few seconds ago
Moisture	{"Moisture":54}	json	a few seconds ago
Humidity	{"randomNumber":70}	json	a few seconds ago
Temperature	{"Temperature":68}	json	a few seconds ago

RESPONSE TIME

Typical time taken for the message to reach the user after the detection of intrusion is 30 to 40 seconds .So the response time is average around 35 seconds.

10.ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- All the data like climatic conditions and changes in them, soil or crop conditions everything can be easily monitored.
- Risk of crop damage can be lowered to a greater extent.

- Many difficult challenges can be avoided making the process automated and the quality of crops can be maintained.
- The process included in farming can be controlled using the web applications from anywhere, anytime.
- Live monitoring can be done of all the processes and the conditions on the agricultural field

DISADVANTAGES

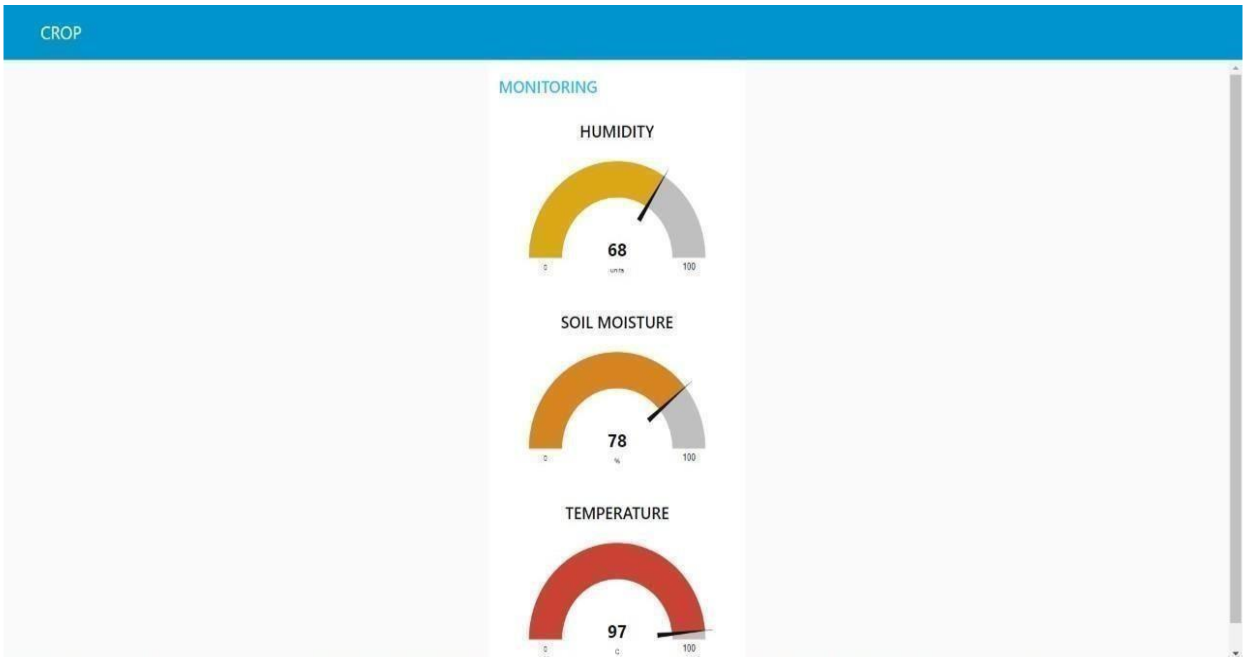
- Smart Agriculture requires internet connectivity continuously, but rural parts cannot fulfil this requirement.
- Any faults in the sensors can cause great loss in the agriculture, due to wrong records and the actions of automated processes.

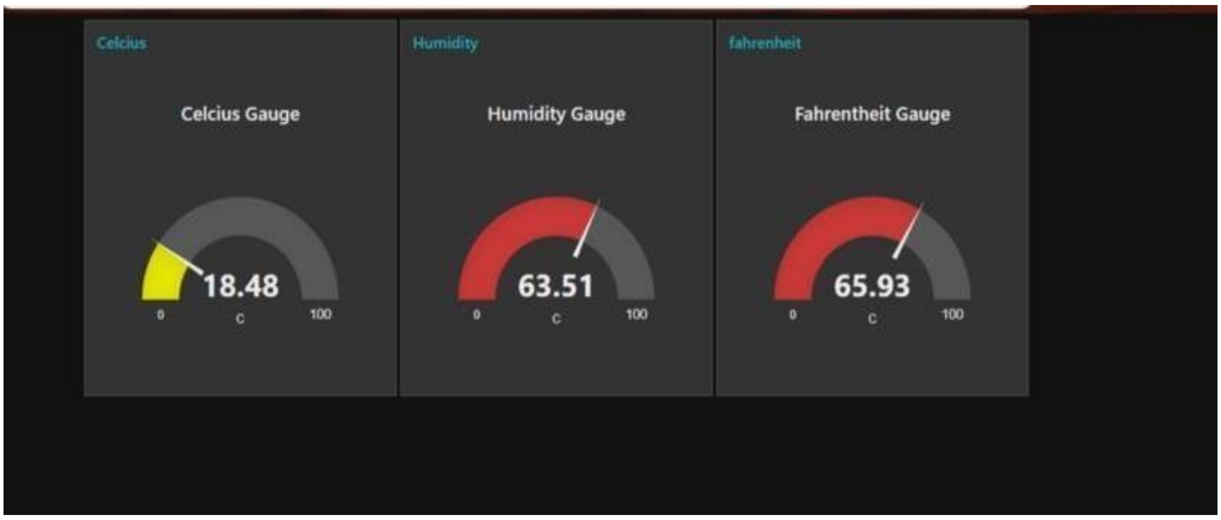
13.UAT Execution & Report Submission

Date	18 November 2022
Team ID	PNT2022TMID24163
Project Name	Project – IOT based smart crop protection using
Maximum Marks	4 Marks

The obtained output get stimulated and got the result with the random values.

Display the image and pre-process the level of the Node-RED web UI and display the temperature, humidity, and soil moisture levels. Integrate the buttons in the UI to control the Motors





11.CONCLUSION

An IoT Based Web Application is built for smart agricultural system using Watson IoT get alerted and to get an field situation when he/she is far from the cultivating field . It will notify the climate conditions at the current state and get alerted and also protect the fields and crops from the animals and birds by get alerting the farmer by the same application .

This was an integrated application system for the farmers that they will get very much easier, eco friendly and also consume very low cost which can be affordable by the farmers

RESULTS :

We have successfully completed the project works that the integrated systems of crop protection from the animals and measuring the climate conditions of the field and to alert the farmer using the web application.

12. FUTURE SCOPE

The proposed work system is a successful working prototype that fulfils to protect crops from the intrusion of animals and birds.

This system will helps the users to monitor the temperature and to notify the weather conditions.

This system assuredly assists the users to know about the soil moisture level. And the IoT based smart crop protection system implemented here brings a naval approach crop protection system from animals.

This assures the early detection and prevention of incurring losses due to the damage of crops.

SOURCE CODE PYTHON SCRIPT

A. MOTOR.PY

```
import time import sys import ibmiotf.application # to install
pip install ibmiotf import ibmiotf.device

# Provide your IBM Watson Device Credentials organization =
"8gyz7t" # replace the ORG ID deviceType = "weather_monitor" #
replace the Device type deviceId = "b827ebd607b5" # replace Device
ID authMethod = "token"
authToken = "LWVpQPaVQ166HWN48f" # Replace the authtoken


 def myCommandCallback(cmd): # function for Callback if

     cmd.data['command'] == 'motoron': print("MOTOR
         ON IS RECEIVED")

     elif cmd.data['command'] == 'motoroff': print("MOTOR
         OFF IS RECEIVED")

     if cmd.command == "setInterval": if

         'interval' not in cmd.data: print("Error - command is missing required
             information: 'interval'")
         else:
             interval = cmd.data['interval'] elif
     cmd.command == "print":
         if 'message' not in cmd.data:
             print("Error - command is missing required information: 'message'")
         else:
             output = cmd.data['message'] print(output)
```

```
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:
    deviceCli.commandCallback = myCommandCallback

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

SENSOR.PY

```
import time import sys
import ibmiotf.application
import ibmiotf.device import
random
# Provide your IBM Watson Device Credentials organization =
"8gyz7t" # replace the ORG ID deviceType = "weather_monitor" #
replace the Device type deviceId = "b827ebd607b5" # replace Device
ID authMethod = "token" authToken = "LWVpQPaVQ166HWN48f"
# Replace the authtoken



def myCommandCallback(cmd):
      print("Command received: %s" % cmd.data['command']) 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:
       temp=random.randint(0,100)
       pulse=random.randint(0,100)
       soil=random.randint(0,100)
 data = { 'temp' : temp, 'pulse': pulse ,'soil':soil}
       #print        data        def
       myOnPublishCallback():
         print ("Published Temperature = %s C" % temp, "Humidity = %s %%" % pulse,"Soil
Moisture = %s %%" % soil,"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()
```

B. Node-RED FLOW :

```
[
{ "id":"625574ead9839b34",
"type":"ibmiotout", "z":"630c8601c5ac3295",
"authentication":"apiKey",
"apiKey":"ef745d48e395ccc0",
"outputType":"cmd",
"deviceId":"b827ebd607b5",
"deviceType":"weather_monitor",
"eventCommandType":"data",
"format":"json",
"data":"data", "qos":0,
"name":"IBM IoT",
"service":"registered",
"x":680,
"y":220,
"wires":[]
},
{
"id":"4cff18c3274cccc4", "type":"ui_button",
"z":"630c8601c5ac3295",
"name":"", "group":"716e956.00eed6c",
"order":2,
"width":"0",
"height":"0", "passthru":false,
"label":"MotorON",
"tooltip":"",
"color":"",
"bgcolor":"",
"className":"",
"icon":"", "payload":"{\"command\":\"motoron\"}",
"payloadType":"str",
"topic":"motoron",
```

"topicType":"str",
"x":360,
"y":160, "wires":[["625574ead9839b34"]]},
{
"id":"659589baceb4e0b0", "type":"ui_button",
"z":"630c8601c5ac3295",
"name":"", "group":"716e956.00eed6c",
"order":3,
"width":"0",
"height":"0", "passthru":true,
"label":"MotorOFF",
"tooltip":"",
"color":"",
"bgcolor":"",
"className":"",
"icon":"", "payload":"{\"command\":\"motoroff\"}",
"payloadType":"str",
"topic":"motoroff",
"topicType":"str",
"x":350,
"y":220, "wires":[["625574ead9839b34"]]},
{"id":"ef745d48e395ccc0",
"type":"ibmiot",
"name":"weather_monitor",
"keepalive":"60",
"serverName":"",
"cleansession":true, "appId":"",
"shared":false},
{"id":"716e956.00eed6c",
"type":"ui_group",
"name":"Form",
"tab":"7e62365e.b7e6b8",
"order":1,
"disp":true,
"width":"6",
"collapse":false},

{"id":"7e62365e.b7e6b8",
"type":"ui_tab",
"name":"contorl",
"icon":"dashboard",
"order":1, "disabled":false,
"hidden":false}
]

[
{
"id":"b42b5519fee73ee2",
"type":"ibmiotin",
"z":"03acb6ae05a0c712",
"authentication":"apiKey",
"apiKey":"ef745d48e395ccc0",
"inputType":"evt",
"logicalInterface":"", "ruleId":"",
"deviceId":"b827ebd607b5",
"applicationId":"",
"deviceType":"weather_monitor",
"eventType":"+",
"commandType":"",
"format":"json",
"name":"IBMIoT",
"service":"registered",
"allDevices":"",
"allApplications":"",
"allDeviceTypes":"",
"allLogicalInterfaces":"",
"allEvents":true,
"allCommands":"",
"allFormats":"",
"qos":0,
"x":270,
"y":180,
"wires":[["50b13e02170d73fc","d7da6c2f5302ffaf","a949797028158f3f","a71f164bc3 78bcf1"]]

},
{ "id":"50b13e02170d73fc",
"type":"function", "z":"03acb6ae05a0c712",
"name":"Soil Moisture",
"func":"msg.payload = msg.payload.soil;\nglobal.set('s',msg.payload);\nreturn msg;", "outputs":1,
"noerr":0,
"initialize":"",
"finalize":"",
"libs":[],
"x":490,
"y":120,
"wires":[["a949797028158f3f","ba98e701f55f04fe"]]
},
{
"id":"d7da6c2f5302ffaf", "type":"function",
"z":"03acb6ae05a0c712", "name":"Humidity",
"func":"msg.payload = msg.payload.pulse;\nglobal.set('p',msg.payload)\nreturn msg;", "outputs":1,
"noerr":0,
"initialize":"",
"finalize":"",
"libs":[],
"x":480,
"y":260, "wires":[["a949797028158f3f","70a5b076eeb80b70"]]
},
{ "id":"a949797028158f3f",
"type":"debug",
"z":"03acb6ae05a0c712",
"name":"IBMo/p",
"active":true, "tosidebar":true,
"console":false, "tostatus":false,
"complete":"payload",
"targetType":"msg",
"statusVal":"",
"statusType":"auto", "x":780,
"y":180,
"wires":[]

},
{
"id":"70a5b076eeb80b70",
"type":"ui_gauge",
"z":"03acb6ae05a0c712", "name":"",
"group":"f4cb8513b95c98a4",
"order":6,
"width":"0",
"height":"0",
"gtype":"gage",
"title":"Humidity",
"label":"Percentage(%)",
"format":"{{value}}",
"min":0, "max":"100",
"colors":["#00b500","#e6e600","#ca3838"], "seg1":"",
"seg2":"",
"className":"",
"x":860,
"y":260,
"wires":[]
},
{
"id":"a71f164bc378bcf1", "type":"function",
"z":"03acb6ae05a0c712",
"name":"Temperature",
"func":"msg.payload=msg.payload.temp;\nglobal.set('t',msg.payload);\nreturn msg;", "outputs":1,
"noerr":0,
"initialize":"",
"finalize":"",
"libs":[],
"x":490,
"y":360,
"wires":[["8e8b63b110c5ec2d","a949797028158f3f"]]
},
{
"id":"8e8b63b110c5ec2d",

"type":"ui_gauge",
"z":"03acb6ae05a0c712", "name":"",
"group":"f4cb8513b95c98a4",
"order":11,
"width":"0",
"height":"0",
"gtype":"gage", "title":"Temperature",
"label":"DegreeCelcius",
"format":"{{value}}",
"min":0, "max":"100",
"colors":["#00b500","#e6e600","#ca3838"], "seg1":"",
"seg2":"",
"className":"",
"x":790,
"y":360,
"wires":[]
},
{
"id":"ba98e701f55f04fe",
"type":"ui_gauge",
"z":"03acb6ae05a0c712", "name":"",
"group":"f4cb8513b95c98a4",
"order":1,
"width":"0",
"height":"0",
"gtype":"gage", "title":"Soil
Moisture",
"label":"Percentage(%)",
"format":"{{value}}",
"min":0, "max":"100",
"colors":["#00b500","#e6e600","#ca3838"], "seg1":"",
"seg2":"",
"className":"",
"x":790,
"y":120,
"wires":[]

},
{
"id":"a259673baf5f0f98",
"type":"httpin",
"z":"03acb6ae05a0c712",
"name":"",
"url":"/sensor",
"method":"get",
"upload":false,
"swaggerDoc":"",
"x":370,
"y":500,
"wires":[["18a8cdbf7943d27a"]]
},
{
"id":"18a8cdbf7943d27a", "type":"function",
"z":"03acb6ae05a0c712",
"name":"httpfunction",
"func":"msg.payload{\"pulse\":global.get('p'),\"temp\":global.get('t'),\"soil\":global.get(
's')};\nreturn msg;", "outputs":1,
"noerr":0,
"initialize":"",
"finalize":"",
"libs":[],
"x":630,
"y":500, "wires":[["5c7996d53a445412"]]
},
{ "id":"5c7996d53a445412",
"type":"httpresponse",
"z":"03acb6ae05a0c712",
"name":"",

"statusCode":"",
"headers":{
}, "x":870,
"y":500,
"wires":[]
},
{
"id":"ef745d48e395ccc0",
"type":"ibmiot",
"name":"weather_monitor",
"keepalive":"60",
"serverName":"",
"cleansession":true,
"appId":"",
"shared":false},
{
"id":"f4cb8513b95c98a4",
"type":"ui_group", "name":"monitor",
"tab":"1f4cb829.2fdee8",
"order":2,
"disp":true,
"width":"6",
"collapse":fals
e,
"className":"
" },
{
"id":"1f4cb829.2fdee8",
"type":"ui_tab",
"name":"Home",
"icon":"dashboard",
"order":3,
"disabled":false,
"hidden":false }

GITHUB LINK: https://github.com/IBM-EPBL/IBM-Project-51498-1660979972

DEMO LINK: https://drive.google.com/drive/u/3/folders/1CVSotWBGOsK2Ogv6j8N5e7_2MJNoItWl