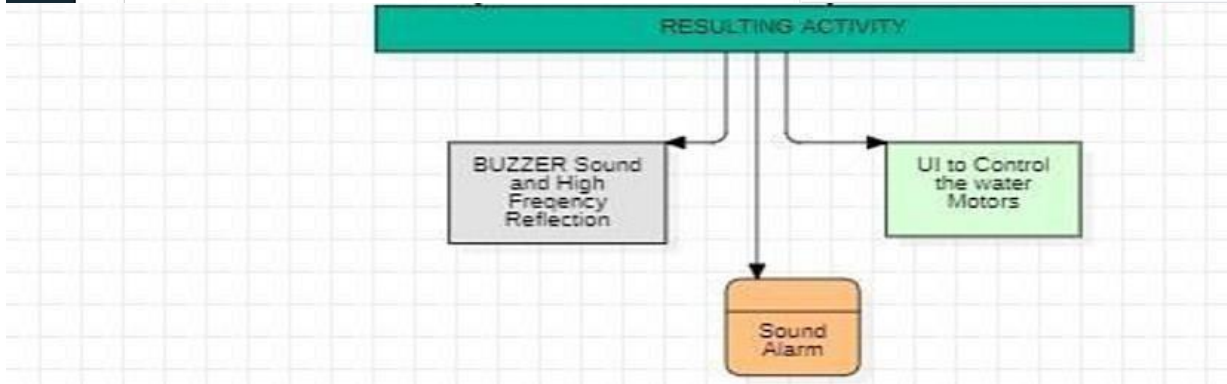


Items per page 50 | 1-1 of 1 item

1 Simulation running



		isn't a critical production application, which means that operations or production don't go down if the IOT solution is down.
6.	Scalability	System must handle expanding load & data retention needs that are based on the upscaling of the solution scope, such as extra manufacturing facilities and extra buildings.

# IOT BASED SMART CROP PROTECTION SYSTEM FOR AGRICULTURE

**Team ID : PNT2022TMID33550**

**Team Members:**

Manojkumar AR

Kavinkumar R

Sachin RM

Logeshwar R

# INTRODUCTION

## **PROJECT OVERVIEW:**

Crops in farms are many times ravaged by local animals like buffaloes, cows, goats, birds etc. this leads to huge losses for the farmers. It is not possible for farmers to barricade entire fields or stay on field 24 hours and guard it. So here we propose an automatic crop protection system from animals. This is a microcontroller based system using PIC family microcontroller. The microcontroller now sounds an alarm to woo the animal away from the field as well as sends SMS to the farmer so that he may be about the issue and come to the spot in case the animal doesn't turn away by the alarm. This ensures complete safety of crop from animals thus protecting farmers' loss.

## **PURPOSE:**

Our main purpose of the project is to develop an intruder alert to the farm, to avoid losses due to animal and fire. These intruder alerts protect the crop that is damaged, which indirectly increases the yield of the crop. The developed system will not be harmful and injurious to animals as well as human beings. The theme of the project is to design an intelligent security system for farm protection by using an embedded system.

## **LITERATURE SURVEY**

### **EXISTING PROBLEM:**

The existing system mainly provide the surveillance functionality. Also these system don't provide protection from wild animals, especially in such an application area. They also need to take actions based on the type of animal that tries to enter the area, as different methods are adopted to prevent different animals from entering restricted areas. The other commonly used method by farmer in order to prevent the crop vandalization by animals include building physical barriers, use of electric fences and manual surveillance and various such exhaustive and dangerous method.

### **REFERENCES:**

- i. Mr.Pranav shitap, Mr.Jayesh redij, Mr.Shikhar Singh, Mr.Durvesh Zagade, Dr. Sharada Chougule. Department of ELECTRONICS AND TELECOMMUNICATION ENGINEERING, Finolex Academy of Management and technology, ratangiri, India.
- ii. N.Penchalaiah, D.Pavithra, B.Bhargavi, D.P.Madhurai, K.EliyasShaik,S.Md.sohaib.Assitant Professor, Department of CSE,AITS, Rajampet,India UG Student, Department of CSE,AITS,Rajampet, India.
- iii. Mr.P.Venkateswara Rao, Mr.Ch Shiva Krishna ,MR M Samba Siva ReddyLBRCE,LBRCE,LBRCE.
- iv. Mohit Korche,Sarthak Tokse, ShubhamShirbhate, Vaibhav Thakre,S. P. Jolhe(HOD). Students , Final Year,Dept.of Electrical engineering,Government

College of engineering,Nagpur head of dept.,Electrical engineering,Government  
College of engineering,Nagpur.

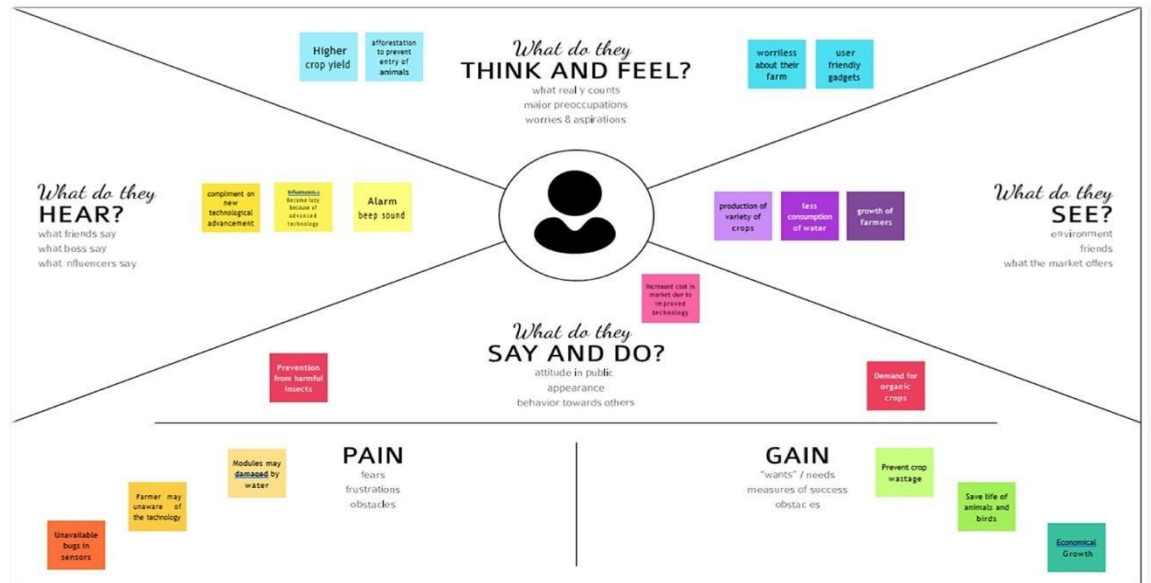
## **PROBLEM STATEMENT   DEFINITION STATEMENT:**

In the world economy of many Country dependent upon the agriculture.

In spite of economic development agriculture is the backbone of the economy. Crops in farms are many times ravaged by local animals like buffaloes, cows, goats, birds and fire etc. this leads to huge loss for the farmers. it is not possible for farmers to blockade to entire fields or stay 24 hours and guard it. Agriculture meets food requirements of the people and produces several raw materials for industries. But because of animal interference and fire in agricultural lands, there will be huge loss of crops. Crops will be totally getting destroyed.

# IDEATION AND PROPOSED SOLUTION

## EMPATHY MAP CANVAS:



a.

# IDEATION AND BRAINSTORMING:

## Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare  
1 hour to collaborate  
2-8 people recommended

### Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

### Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

### Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

- Team gathering**  
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.
- Set the goal**  
Think about the problem you'll be focusing on solving in the brainstorming session.
- Learn how to use the facilitation tools**  
Use the Facilitation Superpowers to run a happy and productive session.  
Open article



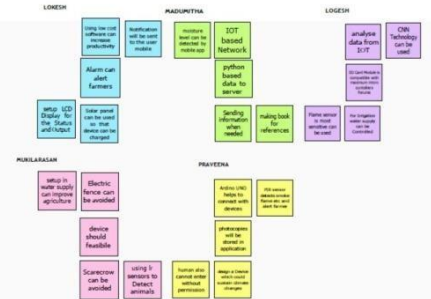
#### Key rules of brainstorming

To run an smooth and productive session

Stay in topic. Encourage wild ideas.

Defer judgment. Listen to others.

Go for volume. If possible, be visual.



### Group Ideas

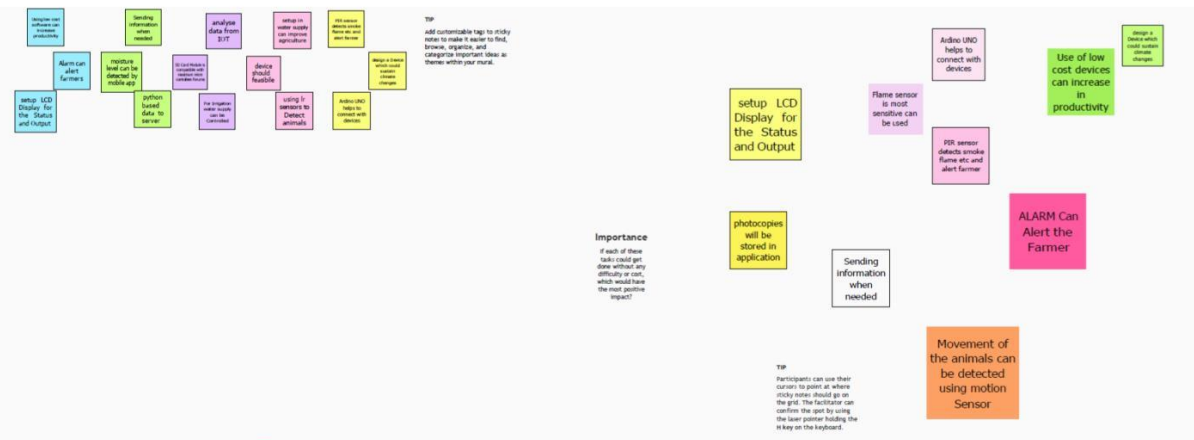
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

### 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.

20 minutes



## **PROPOSED SOLUTION:**

S.NO.	Parameter	Description
1.	Problem Statement. (Problem to be solved)	<ul style="list-style-type: none"><li>✓ Crops are not irrigated properly due to insufficient labour forces.</li><li>✓ Improper maintenance of crops against various environmental factors such as temperature climate, topography and soil quantity which results in crop destruction.</li><li>✓ Requires protecting crops from wild animals attacks birds and pests.</li></ul>
2.	Idea /Solution Description.	<ul style="list-style-type: none"><li>✓ Moisture sensor is interfaced with Arduino Microcontroller to measure the moisture level in soil and relay is used to turn ON &amp; OFF the motor pump for managing the excess water level. It will be updated to authorities through IOT.</li><li>✓ Temperature sensor connected to microcontroller is used to monitor the temperature in the field.</li><li>✓ Image processing techniques with IOT is followed for crop protection against animal attack.</li></ul>
3.	Novelty / Uniqueness.	✓ Automatic crop maintenance and protection using embedded and IOT Technology.
4.	Social Impact / Customer satisfaction.	✓ This proposed system provides many facilities which helps the farmers to maintain the crop field without much loss.
5.	Business Model (Revenue Model).	✓ This prototype can be developed as product with minimum cost with high performance.
6.	Scalability of the solution	✓ This can be developed to a scalable product by using solution sensors and transmitting the data through Wireless Sensor Network and Analysing the data in cloud and operation is performed using robots.

a.



## PROBLEM SOLUTION FIT:

Project Title: IOT Based Smart Crop Protection System for Agriculture		Project Design Phase: I Solution Fit		Team ID: PNT2022TMD20130	
Define CS, fit into CL	<b>1. CUSTOMER SEGMENT(S)</b> <span>CS</span> <ul style="list-style-type: none"> <li>Farmers who trying to protect crops from various problems</li> </ul>	<b>6. CUSTOMER LIMITATIONS</b> <span>CL</span> <small>EG. BUDGET, DEVICES</small> <ul style="list-style-type: none"> <li>Limited supervision.</li> <li>Limited financial constrains.</li> <li>Lack of manpower.</li> </ul>	<b>5. AVAILABLE SOLUTIONS</b> <span>AS</span> <small>PLUSES &amp; MINUSES</small> <ul style="list-style-type: none"> <li>Automation in irrigation.</li> <li>CCTV camera to monitor and supervise the crops.</li> <li>Alarm system to give alert while animals attacks the crops.</li> </ul>	Explore AS, differentiate	
Focus on PR, tap into BE, understand RC	<b>2. PROBLEMS / PAINS</b> <span>PR</span> <small>+ ITS FREQUENCY</small> <ul style="list-style-type: none"> <li>Crops are not irrigated properly.</li> <li>Improper maintenance of crops.</li> <li>Lack of knowledge among farmers in usage of fertilizers and hence crops are affected.</li> <li>Requires protecting crops from Wild animals attacks, birds and pests.</li> </ul>	<b>9. PROBLEM ROOT / CAUSE</b> <span>RC</span> <ul style="list-style-type: none"> <li>Due to in sufficient labour forces. Due to various environmental factors such as temperature climate, topography and soil quality which results in crop destruction.</li> <li>Due to high ammonia, urea, potassium and high PH level fertilizers.</li> </ul>	<b>7. BEHAVIOR</b> <span>BE</span> <small>+ ITS INTENSITY</small> <ul style="list-style-type: none"> <li>Asks suggestions from surrounding peoples and implement there cent technologies.</li> <li>Consumes more time in cropland.</li> <li>Searching for an alternative solution for an existing solution.</li> </ul>	Focus on PR, tap into BE, understand RC	
Identify strong TR & EM	<b>3. TRIGGERS TO ACT</b> <span>TR</span> <ul style="list-style-type: none"> <li>By seeing surrounding cropland with installing machineries.</li> <li>Hearing about innovative technologies and effective solutions.</li> </ul>	<b>10. YOUR SOLUTION</b> <span>SL</span> <ul style="list-style-type: none"> <li>Moisture sensor interfaced with Arduino Microcontroller to measure the moisture level in soil and relay issued tatum ON and OFF the motor pump for managing the excess water level. It will be updated to authorities through IOT.</li> <li>Temperature sensor connected to microcontroller issued to monitor the temperature in the field. The optimum temperature required for crop cultivation is maintained using IOT based fertilizing methods are followed, to minimize the negative effects on growth of crops while using fertilizers.</li> <li>Image processing techniques with IOT is followed for crop protection against animal attacks.</li> </ul>	<b>8. CHANNELS of BEHAVIOR</b> <span>CH</span> <div>ONLINE</div> <ul style="list-style-type: none"> <li>Using different platforms/social media to describe the working and uses of smart crop protection device.</li> </ul> <div>OFFLINE</div> <p>Giving awareness among farmers about the application of the device.</p>	Extract online & offline CH of BE	
	<b>4. EMOTIONS</b> <span>EM</span> <small>BEFORE / AFTER</small> <ul style="list-style-type: none"> <li>Mental frustrations due to insufficient production of crops.</li> <li>Felt smart enough to follow the available technologies with minimum cost.</li> </ul>				

## **REQUIREMENT ANALYSIS**

### **FUNCTIONAL REQUIREMENT:**

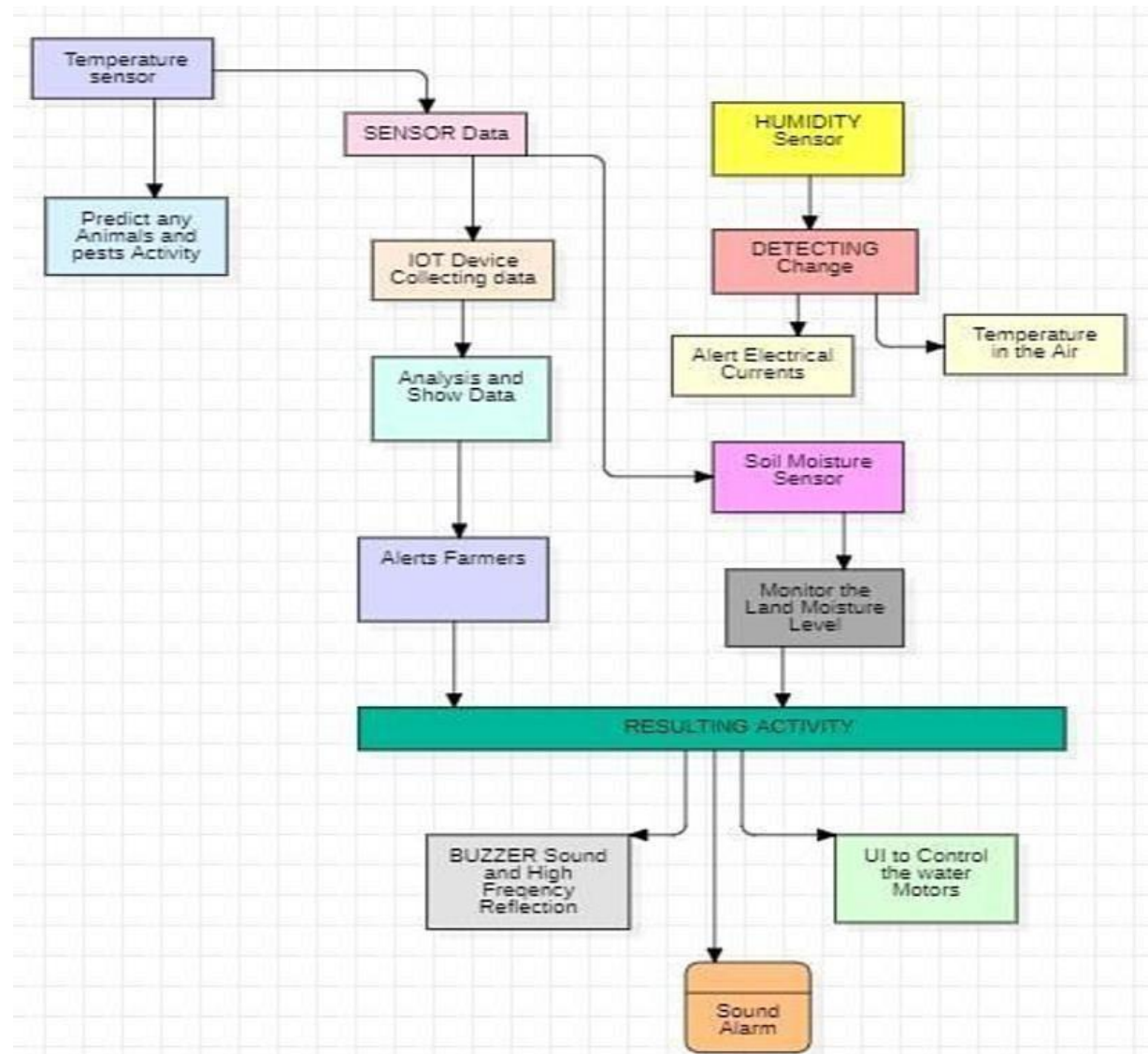
S.NO.	Functional Requirement.	Sub Requirement.
1.	User Visibility	Sense animals nearing the crop field & sounds alarm to woo them away as well as sends SMS to farmer using cloud service.
2.	User Reception	The Data like values of Temperature, Humidity, Soil moisture Sensors are received via SMS.
3.	User Understanding	Based on the sensor data value to get the information about the present of farming land.
4.	User Action	The User needs take action like destruction of crop residues, deep plowing, crop rotation, fertilizers, strip cropping, scheduled planting operations.

## **NON FUNCTIONAL REOUIREMENT:**

S.NO.	Non-Functional Requirement.	Description.
1.	Usability	Mobile Support Users must be able to interact in the same roles & tasks on computers & mobile devices where practical, given mobile capabilities.
2.	Security	Data requires secure access to must register and communicate securely on devices and authorized users of the system who exchange information must be able to do.
3.	Reliability	It has a capacity to recognize the disturbance near the field and doesn't give a false caution signal.
4.	Performance	Must provide acceptable response times to users regardless of the volume of data that is stored and the analytics that occurs in background. Bidirectional, near real-time communications must be supported. This requirement is related to the requirement to support industrial and device protocols at the edge.
5.	Availability	IOT Solutions and domains demand highly available systems for 24 x 7 operations. Isn't a critical production application, which means that operations or production don't go down if the IOT solution is down.
6.	Scalability	System must handle expanding load & data retention needs that are based on the upscaling of the solution scope, such as extra manufacturing facilities and extra buildings.

## PROJECT DESIGN

### DATA FLOW DIAGRAM:



## **SOLUTION AND TECHNICAL ARCHITECTURE:**

a.

## **USER STORIES:**

## PROJECT PLANNING AND SCHEDULING

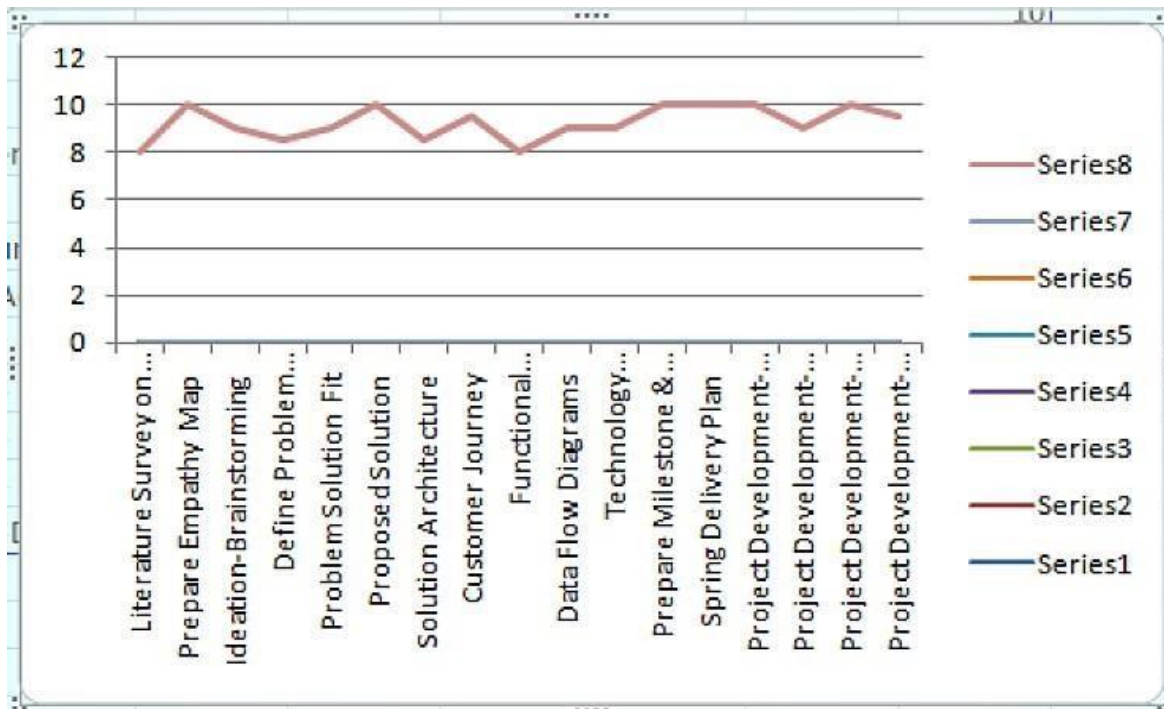
### SPRINT PLANNING AND ESTIMATION:

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	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

#### Velocity:

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

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



# CODING AND SOLUTIONING

## FEATURE-1

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

#BM 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(%) is high, so motor is ON to take water out "
%water_level }, qos=0)
    sleep(1)
if success:
    print('Published alert6 : ' , "water level(%) 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()

```

IBM Watson IoT Platform

Browse Action Device Types Interfaces

Identity Device Information Recent Events State Logs

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

- Rated Voltage : 6V DC
- Operating Voltage : 4 to 8V DC

- Rated Current\*:  $\leq 30\text{mA}$
- SoundOutput at 10cm\* :  $\geq 85\text{dB}$
- Resonant Frequency :  $2300 \pm 300\text{Hz}$
- 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 vehicles such as ambulances, police cars and fire trucks. There are two general types, pneumatic and electronic.

## **FEATURE-2:**

- i. Good sensitivity to Combustible gas in wide range .
- ii. High sensitivity to LPG, Propane and Hydrogen .
- iii. Long life and low cost.
- iv. Simple drive circuit.

# TESTING

## TEST CASES:

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

# User Acceptance Testing:



## Downloads

Latest LTS Version: 18.12.1 (includes npm 8.19.2)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
 Windows Installer <small>node-v18.12.1-x64.msi</small>	 macOS Installer <small>node-v18.12.1.pkg</small>	 Source Code <small>node-v18.12.1.tar.gz</small>

Windows Installer (.msi)

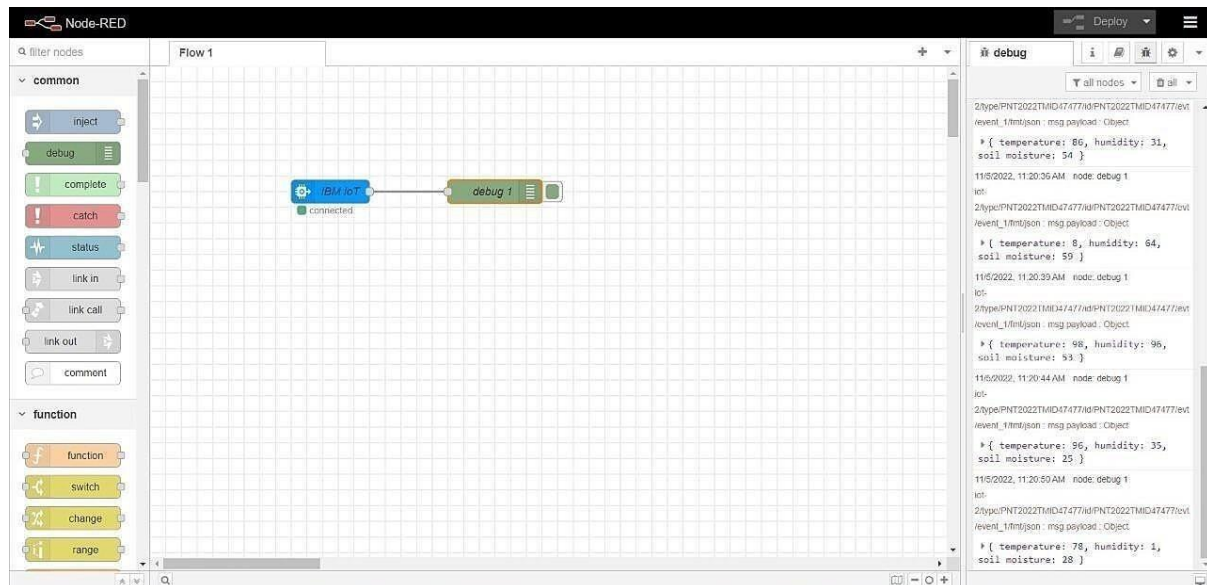
Windows Binary (.zip)

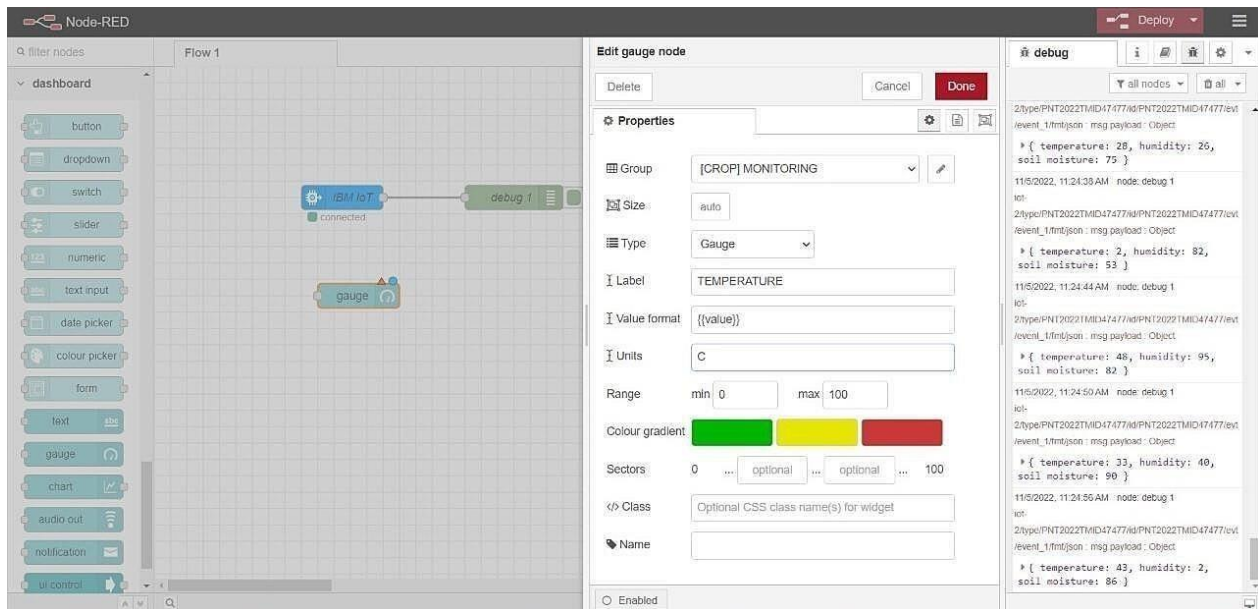
macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x64)

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	





```
node-red
4 Nov 18:48:05 - [info] Node-RED version: v3.0.2
4 Nov 18:48:05 - [info] Node.js version: v18.12.0
4 Nov 18:48:05 - [info] Windows_NT 10.0.19044 x64 LE
4 Nov 18:48:26 - [info] Loading palette nodes
4 Nov 18:48:44 - [info] Settings file : C:\Users\ELCOT\.node-red\settings.js
4 Nov 18:48:45 - [info] Context store : 'default' [module=memory]
4 Nov 18:48:45 - [info] User directory : \Users\ELCOT\.node-red
4 Nov 18:48:45 - [warn] Projects disabled : editorTheme.projects.enabled=false
4 Nov 18:48:45 - [info] Flows file : \Users\ELCOT\.node-red\flows.json
4 Nov 18:48:45 - [info] Creating new flow file
4 Nov 18:48:45 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
4 Nov 18:48:45 - [warn] Encrypted credentials not found
4 Nov 18:48:45 - [info] Starting flows
4 Nov 18:48:46 - [info] Started flows
4 Nov 18:48:46 - [info] Server now running at http://127.0.0.1:1880/
```

## **RESULTS**

The problem of crop vandalization by wild animals and fire has become a major social problem in current time.

It requires urgent attention as no effective solution exists till date for this problem. Thus this project carries a great social relevance as it aims to address this problem. This project will help farmers in protecting their orchards and fields and save them from significant financial losses and will save them from the unproductive efforts that they endure for the protection their fields. This will also help them in achieving better crop yields thus leading to their economic wellbeing.

## **ADVANTAGES AND DISADVANTAGES**

### **Advantage:**

Controllable food supply. you might have droughts or floods, but if you are growing the crops and breeding them to be hardier, you have a better chance of not starving. It allows farmers to maximize yields using minimum resources such as water, fertilizers.

### **Disadvantage:**

The main disadvantage is the time it can take to process the information. In order to keep feeding people as the population grows you have to radically change the environment of the planet



## **CONCLUSION:**

A IoT Web Application is built for smart agricultural system using Watson IoT platform, Watson simulator, IBM cloud and Node-RED

## **FUTURE SCOPE**

In the future, there will be very large scope, this project can be made based on Image processing in which wild animal and fire can be detected by cameras and if it comes towards farm then system will be directly activated through wireless networks. Wild animals can also be detected by using wireless networks such as laser wireless sensors and by sensing this laser or sensor's security system will be activated.

# APPENDIX

## SOURCE CODE

```
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

    cm.data['command'] == 'motoron':

    print("MOTOR ON IS RECEIVED")

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

    if cmd.command == "setInterval":

    else:

    if 'interval' not in cmd.data:

        print("Error - command is missing required information: 'interval'")

    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, "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:

```
    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 = "LWVpQPpVQ166HWN48f" # 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(0,1
00)
    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()
```

## 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":"registere
d", "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,
  "appld": "",
  "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;\n\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":"",

  "l
bs
":[
],
  "x
":
48
0,
  "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", "#e66000", "#ca3838"], "seg1": "",
  "seg2": "",
  "className": "x", "x": 86,
  "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": false,
  "initialize": "",
  "finalize": "",
  "labeled": true,
  "inputs": [
    {
      "name": "temp",
      "type": "text",
      "value": ""
    }
  ],
  "outputs": [
    {
      "name": "temp",
      "type": "text",
      "value": ""
    }
  ]
}
```

```
"x
":
49
0,
"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",
"ctype": "gauge",

"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": "ge
t",
  "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":"","
"li
bs
":[
],
"x
":
63
0,
"y":500, "wires":[["5c7996d53a445412"]]
},
{
"id":"5c7996d53a445412
",
"type":"httpresponse",
"z":"03acb6ae05a0c712
", "name":"","
"statusCode":"","

"header
s":{},
"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":f  
alse,  
"className  
":"  
"},  
{  
"id":"1f4cb829.2fdee8",  
"type":"ui_tab",  
"name":"Home",  
"icon":"dashboard  
", "order":3,  
"disabled":false,  
"hidden":false }
```

## GitHub & Project Demo Link

<https://github.com/IBM-EPBL/IBM-Project-16401-1659613469>

