BETHLAHEM INSTITUTE OF ENGINEERING

(Affiliated to AICTE & ANNA UNIVERSITY)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

REPORT ON

HX 8001 PROFESSIONAL READINESS FOR INNOVATION,

EMPLOYABILITY AND ENTREPENEURSHIP

(Nalaiya thiran program)

PROJECT TITLE

IoT Based Smart Crop Protection System For Agriculture

TEAM ID:PNT2022TMID51293

TEAM MEMBERS

1.JESU SHERLY.S(TEAM LEAD)

2.RUBA.T

3.SAJITHA STARLIN.R

4.SUBISHA.E

MENTOR

MS.HENILIN.J.K

EVALUATOR

MS. FEBIN RANI

INDEX

INTRODUCTION

PROJECT OVREVIEW:

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 tobarricade entire fields or stay on field 24 hours and guard it.so here we propose automatic crop protection system from animals. This is a microcontroller based system using PIC family microcontroller. The microcontroller now sound an alarm to woo the animal away from the field as well as sends SMS to the farmer so that he may about the issue and come to the spot in case the animal don't turn away by the alarm. This ensures complete safetyof crop from animals thus protecting farmers loss.

PURPOSE:

Our main purpose of the project is to develop intruder alert to the farm, to avoid losses due to animal and fire. These intruder alert protect the crop that damaging that indirectly increase yield of the crop. The develop system will not harmful and injurious to animal as well as human beings. Theme of project is to design a intelligent security system for farm protecting by using 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 andmanual surveillance and various such exhaustive and dangerous method.

REFERENCES:

- Ms.Jesu sherly, Ms.Ruba, Ms.Sajitha Starlin and Ms.Subisha of Computer Science and Engineering Department.

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

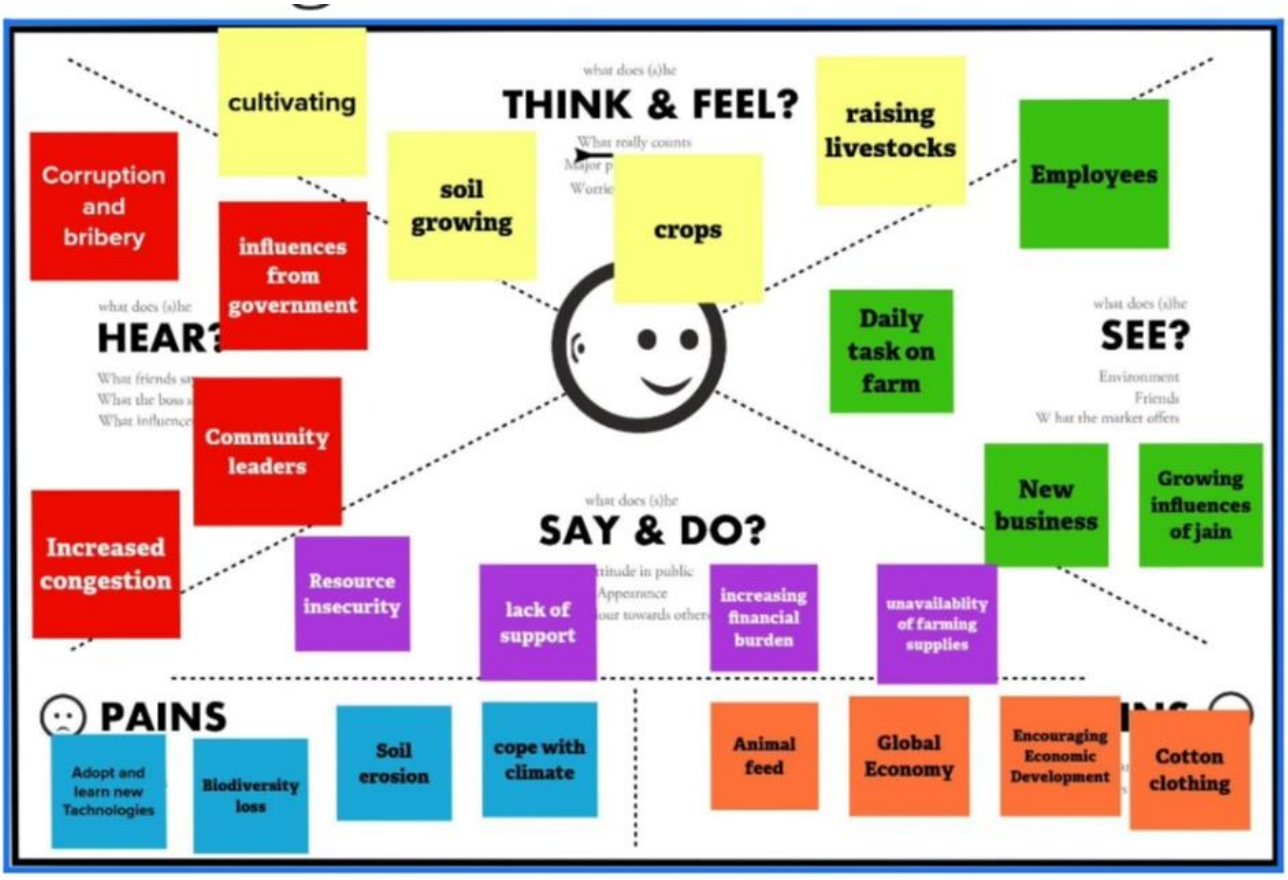
physical barriers, use of electric fences andmanual surveillance and various such exhaustive and dangerous method.

REFERENCES:

- Ms.Jesu sherly, Ms.Ruba, Ms.Sajitha Starlin and Ms.Subisha of Computer Science and Engineering Department.
-
- St.Joseph College Of Engineering ,Sriperumbudur,Chennai
- Mr.P.Venkateswara Rao, Mr.Ch Shiva Krishna ,MR M Samba Siva ReddyLBRCE,LBRCE,LBRCE.
- 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.

IDEATION AND PROPOSED SOLUTION

EMPATHY MAP CANVAS:



IDEATION AND BRAINSTORMING:

Template

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

Share template feedback

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

- **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 →

1 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

PROBLEM

Smart crop production system for Agriculture

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Defer judgment.
- Go for volume.
- Encourage wild ideas.
- Listen to others.
- If possible, be visual.

2 Brainstorm

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

10 minutes

TIP

You can select a sticky note and hit the pencil [switch] to switch to start drawing!

Jesu Sherly

- weed Management
- Organic pesicides
- crop Rotation

Ruba

- Maintaining crop diversity
- Enviroment impacts
- use of non renewable resources

Sajitha Starlin

- Domestic and wild biodiversity
- loosing the potential crop pattern
- Types of pesicides spraying equipment

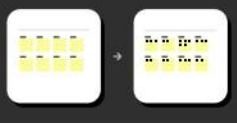
Subisha

- inter cropping
- Physical quality soil
- Soil moisture

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example →



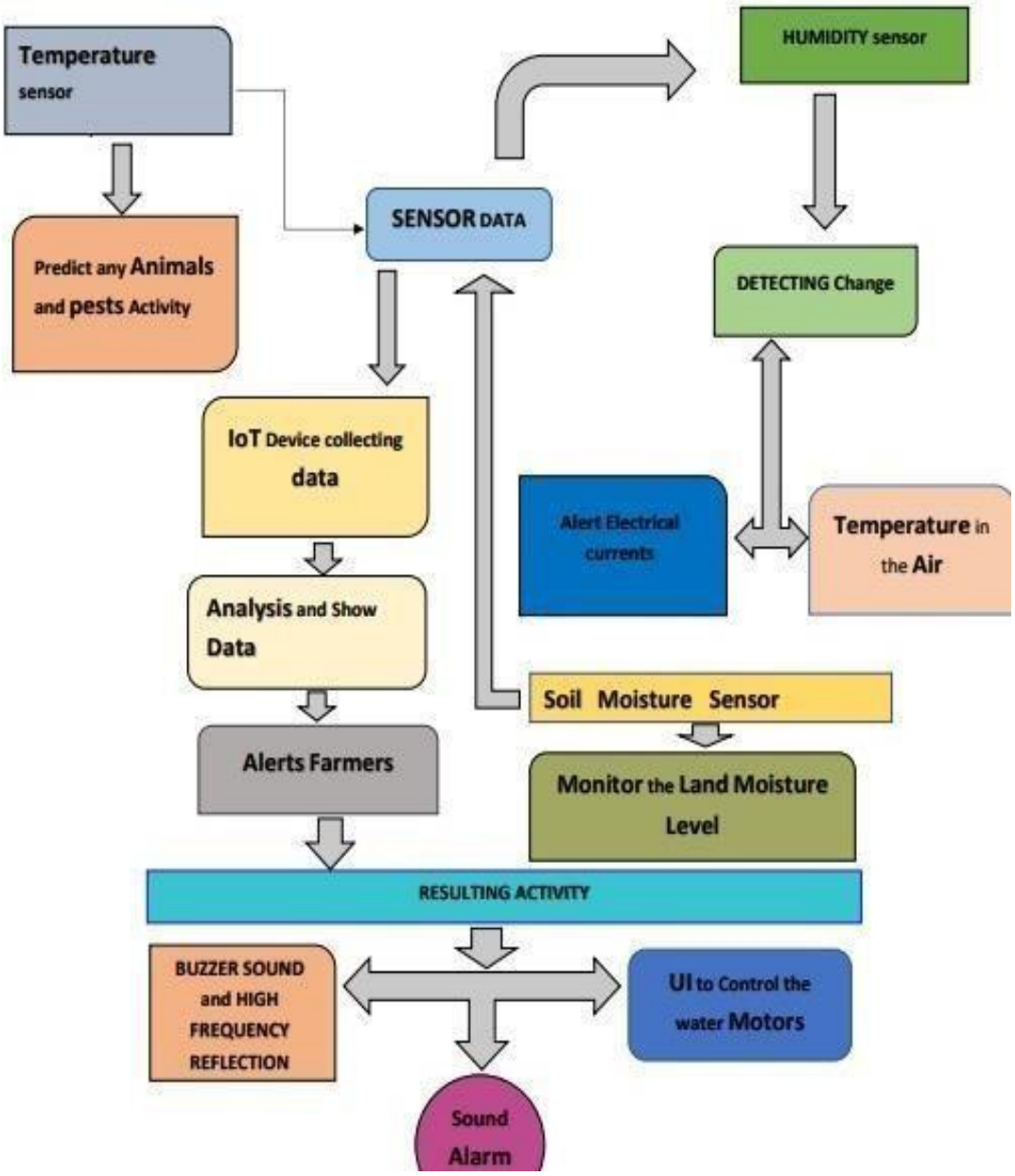
PROPOSED SOLUTION:

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Develop affordable app-based solution for Soil health monitoring and suggest which crop to be sown based on it. (Technology Bucket: IoT, AI, ML etc.)
2.	Idea / Solution description	Create app-based solution to detect soil parameters like moisture content, temperature, relative humidity, nutrient, Ph, CEC, and NPK etc. and provide crop suggestions to be produced based on soil parameters & environment values.
3.	Novelty / Uniqueness	Provide remedies & alerts on soil deficiencies like Watering for low Moisture level, Fertilizers for Nutrient deficiencies etc.
4.	Social Impact / Customer Satisfaction	Farmers can take immediate actions resulting better crop produces and farmers have better income. High Yield and prescriptive guidance.

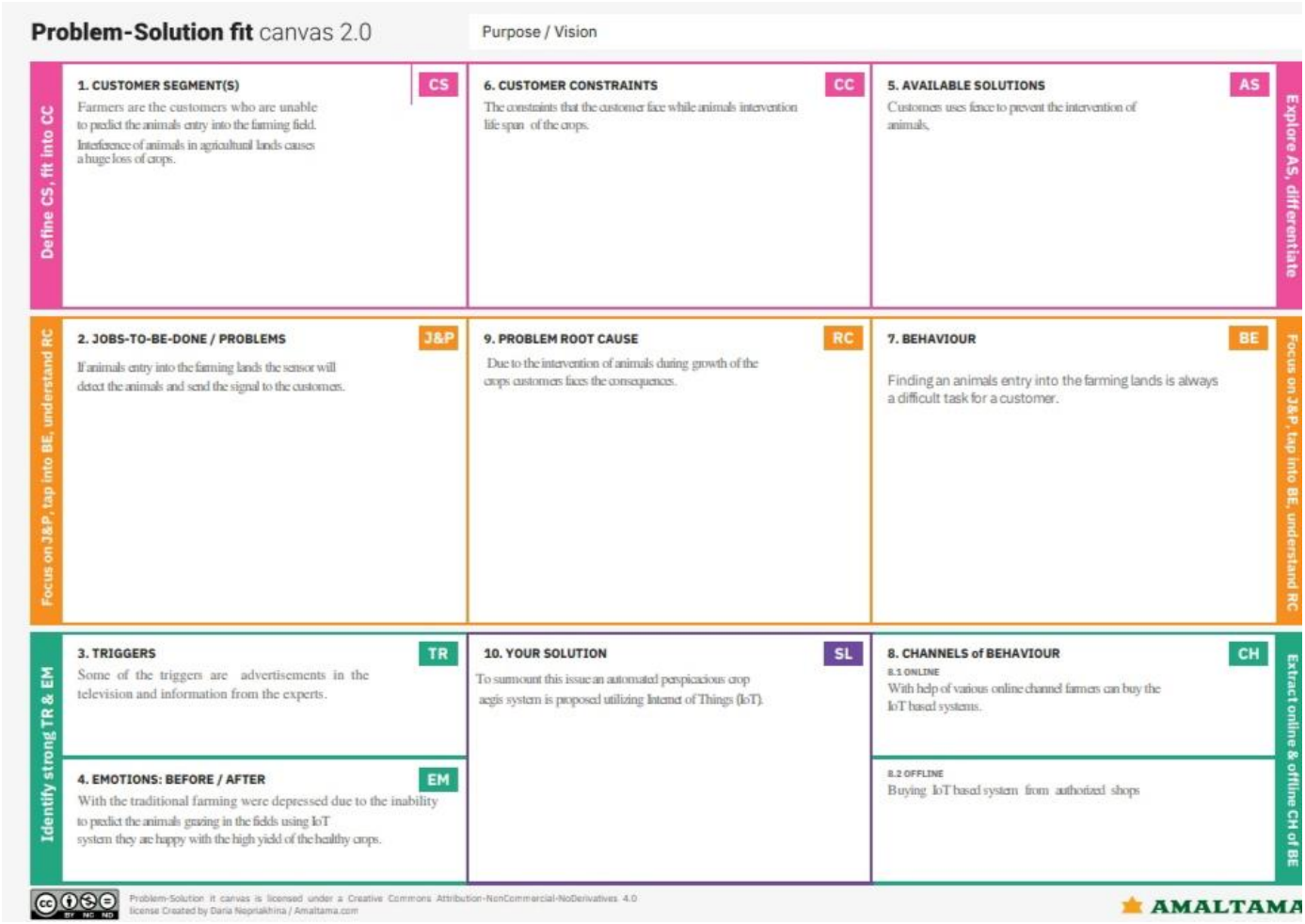
5.	Business Model (Revenue Model)	GSM model
6.	Scalability of the Solution	soilarmor, minimizing soil disturbance, plant diversity, continual live plant/root, and livestock integration.

PROJECT DESIGN

DATA FLOW DIAGRAM:



PROBLEM SOLUTION FIT



REQUIREMENTS ANALYSIS

FUNCTIONAL REQUIREMENTS:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Resource discovery	The specifications define the common services provided by the application service layer in IOT systems, referred to as common service functions. 'Discovery' is one of the defined CSFs which allows IOT entities to send discovery requests to search resources about applications and services.
FR-2	Resource management	The resources considered in Table 1 include batterytime, memory usage, and other data related to application performance to make quality of service reliable. Although some parts of this requirement rely on its implementation of the 'Application and Service Layer Management' and 'Device Management' could probably support these requirements.
FR-3	Data management	The 'Data Management and Repository' is responsible for providing data storage and management converting aggregated data into a specific format and preparing for further analytics such as semantic processing.
FR-4	Event management	The 'Subscription and Notification' can manage subscription to the resources hosted in the platform, and can provide notification containing the changes on the resources to the address where the subscriber wants to receive them. Accordingly, application and services can acquire all the information about the proper events in real-time.
	Code management	The 'Device Management' utilizes the already-existing technologies including broadband forum (BBF) TR-069, OMA-DM, LwM2M for managing device capabilities. Of course, code updating operations for IOT devices could be achieved with the help of management clients, servers, and adapters specifications.

NON FUNCTINAL REQUIREMENT:

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The 'Device Management' allows the application entities registered to an server platform to be easily maintained through existing device management technologies. Also, the Node.js-based implementation enables the middleware components to be updated or replaced accordingly without any high-level of technical expertise.
NFR-2	**Security**	Security is a very critical requirement in IOT solutions and defines its security framework including identification, authorization and authentication. Our middleware platform can be registered to the server (i.e., Mobius) as an application entity. It can attempt to access a list of authorized resources hosted by the server with its server-generated unique identifier and privileges, called access control policy. However, authentication and other security components such as certificates still remain incomplete.
NFR-3	**Reliability**	we have not yet realized capabilities related to Reliability, which allows platform-equipped devices to adapt themselves according to short-term or long-term changes in resource conditions, application scenarios, and surrounding environments, remaining our future work.

NFR-4	**Performance**	This requirement belongs to a part of intelligence for IOT devices, and the proposed IOT device platform provides no analytic tools on data or decision-making procedures depending on resource conditions, for example, recommending the most suitable (or currently available) one among multiple IOT devices offering the same service, which is one area of our future work.
NFR-5	**Availability**	Availability could be achieved by ensuring some level of fault-tolerance. The developed IOT platform does not deal with all fault tolerance issues that mainly occur in hardware interfaces. However, a watchdog function is able to detect the failure of middleware components interacting with hardware interfaces, and restart or reconnect if needed.
NFR-6	**Scalability**	An IOT platform needs to support rapidly growing numbers of IOT devices and keep a certain level of support. Although the scalability of an IOT platform is crucial, it highly depends on implementation and performance in IOT servers rather than connected devices. Accordingly, in support of a well-designed based IOT server we can say that our middleware platform may deliver some level of appropriate for the given environment and applications.

USER STORIES:

User Type	Functional requirement (Epic)	User Story number	User Story/Task	Acceptance criteria	Priority	Release
	Registration	USN-1	User can enter into the web application	I can access my account /dashboard	High	Sprint 1
		USN-2	User can register their credentials like email id and password	I can receive confirmation email & click confirm	High	Sprint 1

Mobile users	Login	USN-3	User can log into the application by entering email & password	I can login to my account	High	Sprint 1
	Dashboard	USN-4	User can view the temperature	I can view the data given by the device	High	Sprint 2
		USN-5	User can view the level of sensor monitoring value	I can view the data given by the device	High	Sprint 2
Web users	Usage	USN-1	User can view the web page and get the information	I can view the data given by the device	High	Sprint 3
Customer	Working	USN-1	User act according to the alert given by the device	I can get the data work according to it	High	Sprint 3
		USN-2	User turns ON the water motors/Buzzer/Sound Alarm when occur the disturbance on field.	I can get the data work according to it		Sprint 4
Customer care Executive	Action	USN-1	User solve the problem when some faces any usage issues	I can solve the issues when some one fails to understanding the procedure	High	Sprint 4
Administration	Administration	USN-1	User store every information	I can store the gained information	High	Sprint 4

PROJECT PLANNING AND SCHEDULING

SPRINT PLANNINGAND 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)

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

```
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.conn
ect()  while
True:
#Getting values from sensors.
 temp_sensor = round(
 random.uniform(0,80),2)
```

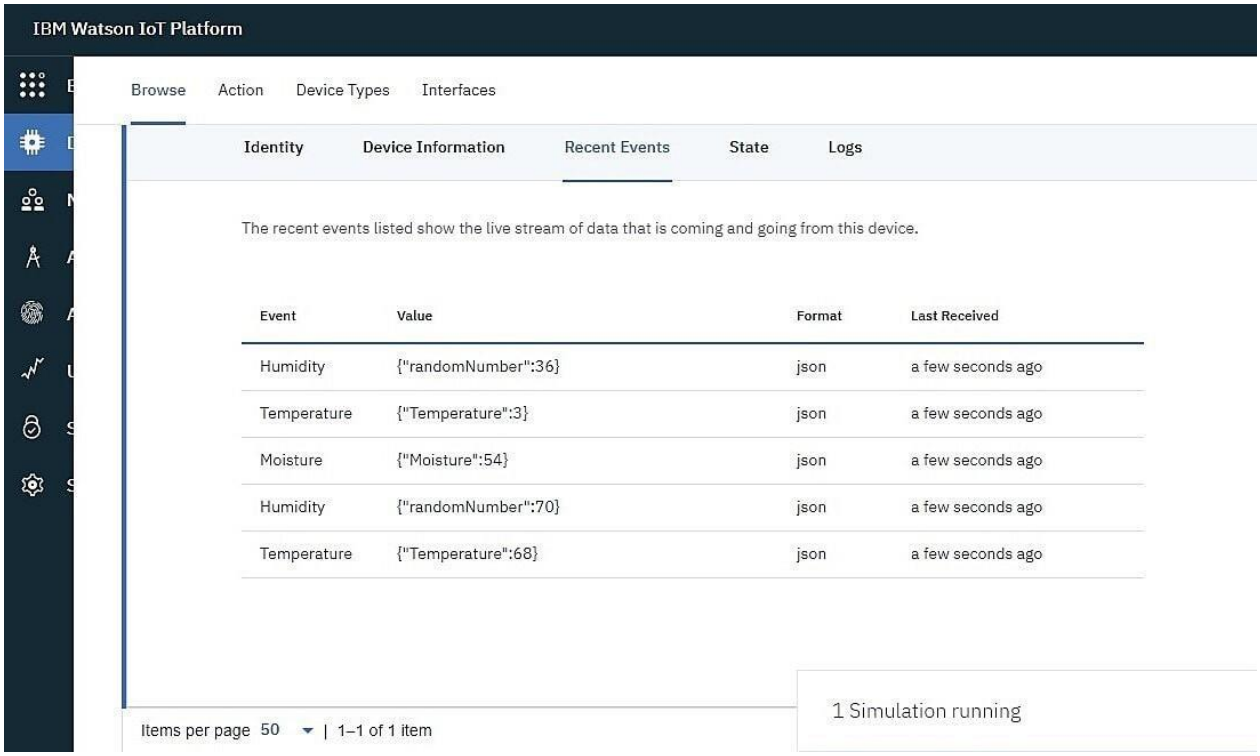
```
PH_sensor =
round(random.uniform(1,14),3)
camera = ["Detected","Not Detected","Not Detected","Not Detected","Not Detected","Not Detected",]
camera_reading = random.choice(camera)
flame = ["Detected","Not Detected","Not Detected","Not
Detected","Not Detected","Not Detected",]  flame_reading =
random.choice(flame)
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
device disconnect()
```

OUTPUT:


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

- 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.

FEATURE-2:

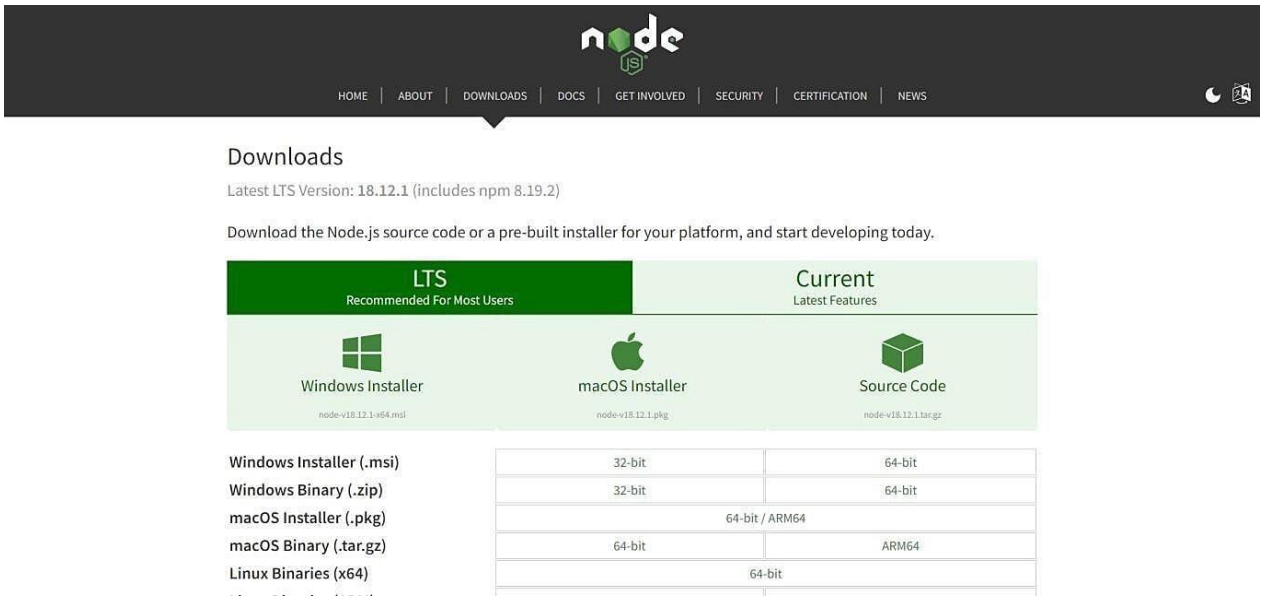
i. Goodsensitivity to Combustible gas in wide range .
ii. Highsensitivity to LPG, Propane and Hydrogen .
iii. Longlife and low cost.
iv. Simpledrive circuit.

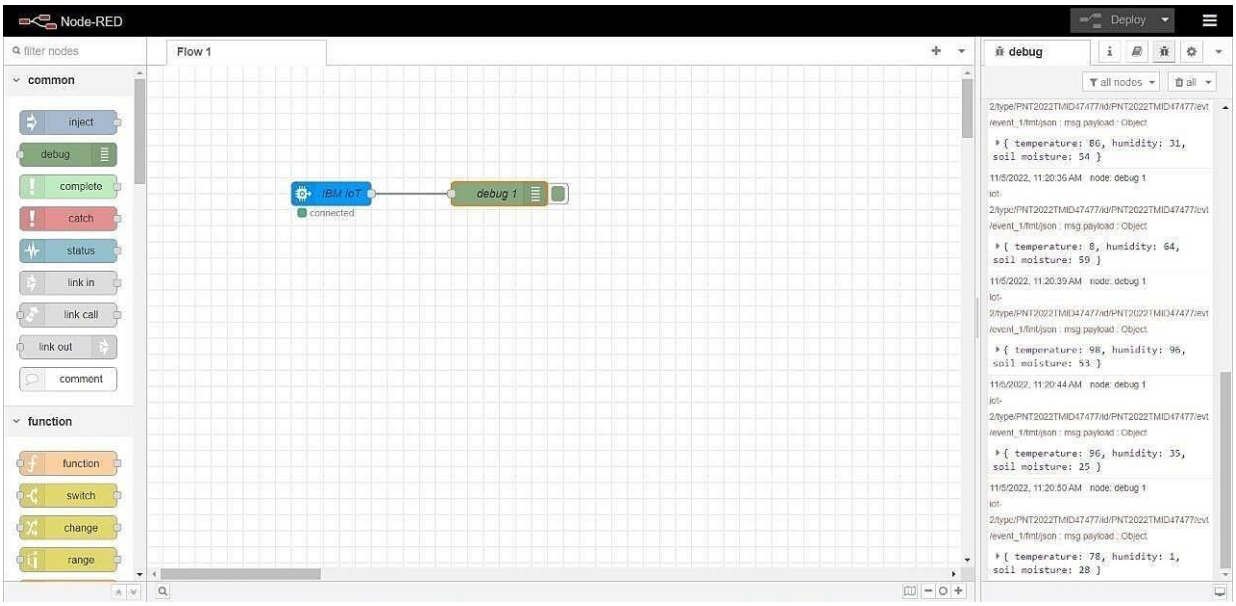
TESTING

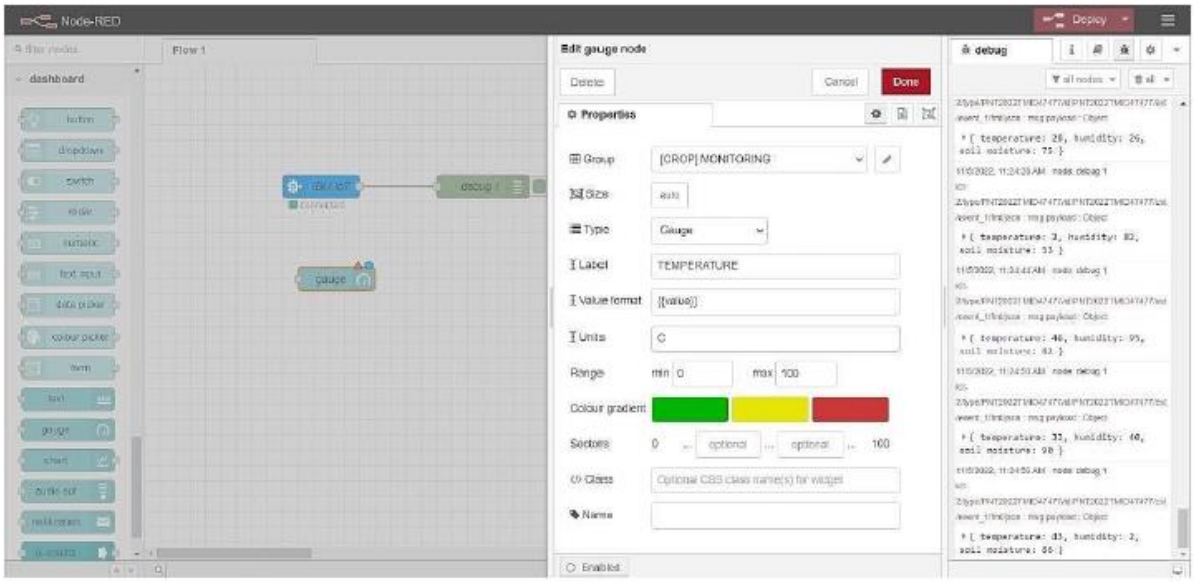
TEST CASES:

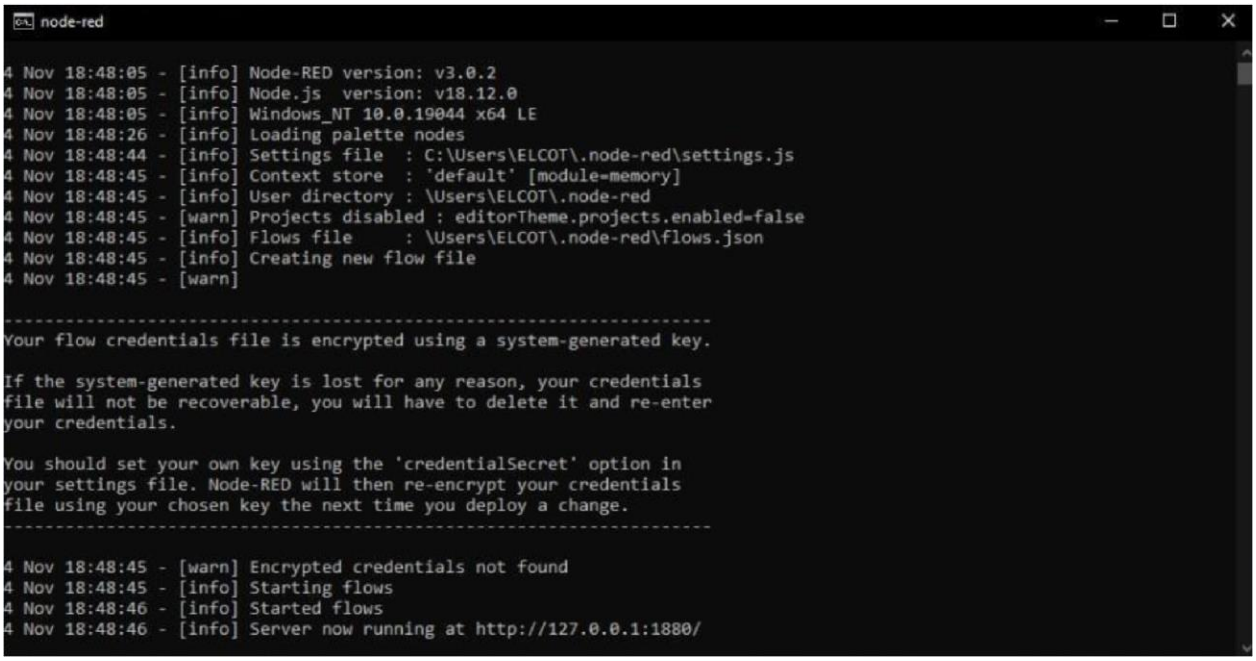
sno	Parameter	Values	Screenshot
1	Model summary	-	
2	Accuracy	Training accuracy-95% Validation accuracy-72%	
3	Confidence score	Class detected-80% Confidence score-80%	

User Acceptance Testing:






Node-RED
Deploy
Flow 1
dashboard
button
dropdown
switch
slider
numeric
text input
date picker
colour picker
form
text
gauge
chart
notification
ui control
debug
gauge
Edit gauge node
Delete
Cancel
Done
Properties
Group
[CROP]MONITORING
Size
auto
Type
Gauge
Label
TEMPERATURE
Value format
{{value}}
Units
C
Range
min 0
max 100
Colour gradient
Sectors
0
optional
optional
100
Class
Optional CSS class name(s) for widget
Name
Enabled
debug
all nodes
{ temperature: 28, humidity: 26, soil moisture: 75 }
{ temperature: 3, humidity: 83, soil moisture: 33 }
{ temperature: 46, humidity: 95, soil moisture: 83 }
{ temperature: 31, humidity: 60, soil moisture: 98 }
{ temperature: 43, humidity: 2, soil moisture: 86 }


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 willhelp 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 achievingbetter crop yields thus leading to theireconomic 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 chanceof not straving. 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 theenvironment of the planet

CONCLUSION

A IoT Web Application is built for smart agricultural system using Watson IoT platform, Watsonsimulator, 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 animaland fire can be detected by cameras and if it comes towards farmthen 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 beactivated.

APPENDIX
SOURCE CODE

```
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-tok
 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 O
     , qos=0)
      sleep(1)
      if success:
         print( 'Published alert1 : ', "Temperature(%s) is high, sprinkerlers are turned ON" %temp_sensor,"to IBM Wats
      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 fertil
     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()
```

```
{
"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":"s
tr","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":"MotorOF
F",
"tooltip":"",
"color":"",
"bgcolor":"",
"className":"",
"icon":"",
"payload":"{\"command\":\"motoroff\"}",
"payloadType":"str",
"topic":"motoroff",
"topicType":"s
tr","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":fal
```

```
se},
{"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":"",
"li
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",
```

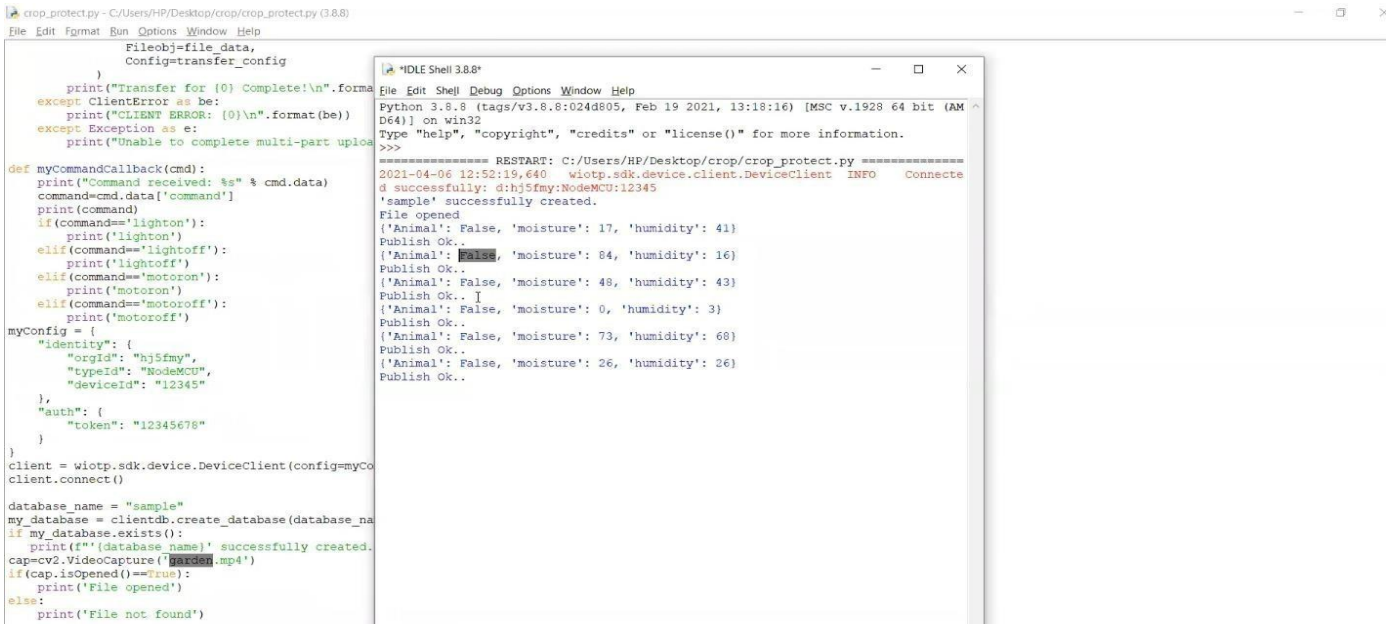
"targetType":"msg",
"statusVal":"",
"statusType":"auto",
"x":780,
"y":180,
"wires":[]
},
{
"id":"70a5b076eeb80b70",
"type":"ui_gauge",
"z":"03acb6ae05a0c712",
"name":"",
"group":"f4cb8513b95c98a4",
"order":6,
"width":"0",
"height":"0",
"gtype":"gage",
"title":"Humidity",
"label":"Percentage(%)",
"format":"{{value}}
","min":0,
"max":"100",
"colors":["#00b500","#e6e600","#ca3838"], "seg1":"",
"seg2":"",
"className
":"","x":86
0,
"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":"",
"li
bs
":[
],
"x

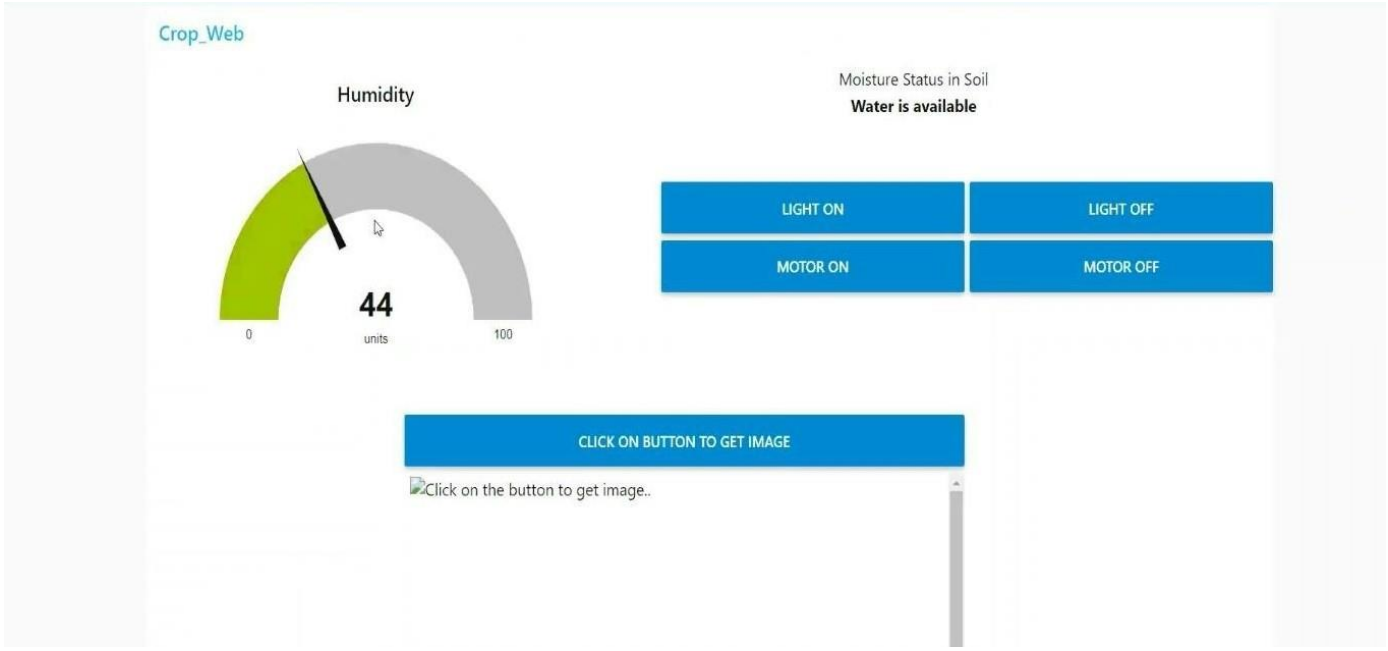
":
49
0,
"y":360,
"wires":[["8e8b63b110c5ec2d","a949797028158f3f"]]
},
{
"id":"8e8b63b110c5ec2d",
"type":"ui_gauge",
"z":"03acb6ae05a0c712",
"name":"",
"group":"f4cb8513b95c98a4",
"order":11,
"width":"0",
"height":"0",
"gtype":"gage",
"title":"Temperature",
"label":"DegreeCelcius",
"format":"{{value}}",
"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",
"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":"ge
t",
"upload":fals
e,
"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 }

OUTPUT


crop_protect.py - C:/Users/HP/Desktop/crop/crop_protect.py (3.8.8)
File Edit Format Run Options Window Help
Fileobj=file_data,
Config=transfer_config
)
print("Transfer for {0} Complete!\n".forma
except ClientError as be:
print("CLIENT ERROR: {0}\n".format(be))
except Exception as e:
print("Unable to complete multi-part uploa
def myCommandCallback(cmd):
print("Command received: %s" % cmd.data)
command=cmd.data['command']
print(command)
if(command=='lighton'):
print('lighton')
elif(command=='lightoff'):
print('lightoff')
elif(command=='motoron'):
print('motoron')
elif(command=='motoroff'):
print('motoroff')
myConfig = {
"identity": {
"orgId": "hj5fmy",
"typeId": "NodeMCU",
"deviceId": "12345"
},
"auth": {
"token": "12345678"
}
}
client = wiotp.sdk.device.DeviceClient(config=myCo
client.connect()
database_name = "sample"
my_database = clientdb.create_database(database_na
if my_database.exists():
print(f"'{database_name}' successfully created.
cap=cv2.VideoCapture('garden.mp4')
if(cap.isOpened()==True):
print('File opened')
else:
print('File not found')
IDLE Shell 3.8.8
File Edit Shell Debug Options Window Help
Python 3.8.8 (tags/v3.8.8:024d805, Feb 19 2021, 13:18:16) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/HP/Desktop/crop/crop_protect.py ==========
2021-04-06 12:52:19,640 wiotp.sdk.device.client.DeviceClient INFO Connecte
d successfully: d:hj5fmy:NodeMCU:12345
'sample' successfully created.
File opened
{'Animal': False, 'moisture': 17, 'humidity': 41}
Publish Ok..
{'Animal': False, 'moisture': 84, 'humidity': 16}
Publish Ok..
{'Animal': False, 'moisture': 48, 'humidity': 43}
Publish Ok..
{'Animal': False, 'moisture': 0, 'humidity': 3}
Publish Ok..
{'Animal': False, 'moisture': 73, 'humidity': 68}
Publish Ok..
{'Animal': False, 'moisture': 26, 'humidity': 26}
Publish Ok..


Crop_Web
Humidity
44
units
0
100
Moisture Status in Soil
Water is available
LIGHT ON
LIGHT OFF
MOTOR ON
MOTOR OFF
CLICK ON BUTTON TO GET IMAGE
Click on the button to get image..

GITHUB LINK:

https://github.com/IBM-EPBL/IBM-Project-40089-1660623266

PROJECT DEMO LINK: