IOT BASED SMART CROP PROTECTION SYSTEM FOR AGRICULTURE

NAALAIYATHIRAN PROJECT REPORT

TEAM ID:PNT2022TMID22829

SUBMITTED BY

TEAM LEAD: KOWSIKA M	732919ECR076
TEAM MEMBERS:	
KANNAN S A	732919ECR059
KANNIKA V	732919ECR060
KISHORE A B	732919ECR074
KEERTHANA V N	732919ECR072

IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY

(AUTONOMOUS)

ERODE

TABLE OF CONTENT

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 to barricade 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 safety of 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 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. iii. Mr.P.Venkateswara Rao, Mr.Ch Shiva Krishna ,MR M Samba Siva Reddy LBRCE,LBRCE,LBRCE.

iv. Mohit Korche, Sarthak Tokse, Shubham Shirbhate, 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.

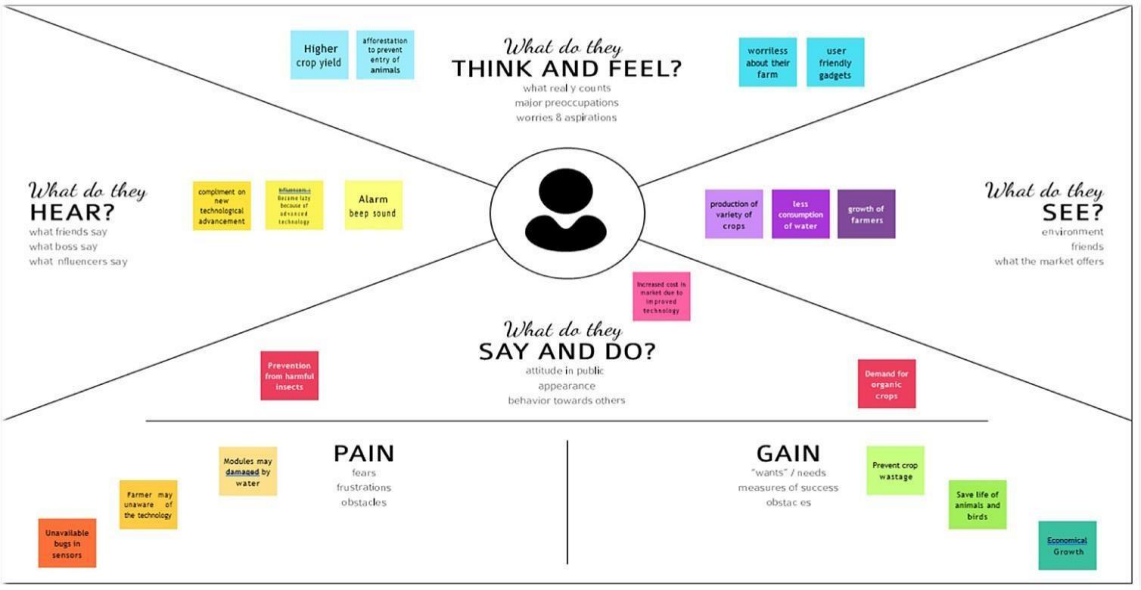
PROBLEM STATEMENT DEFINITION STATEMENT:

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 forms 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.

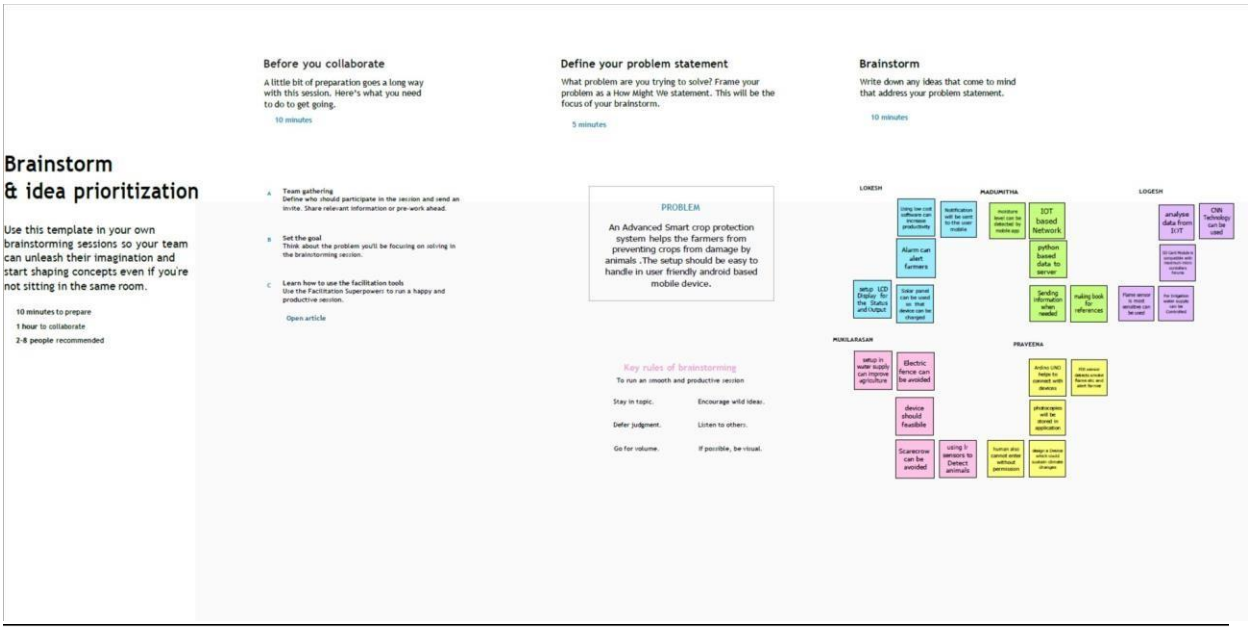
IDEATION AND PROPOSED SOLUTION

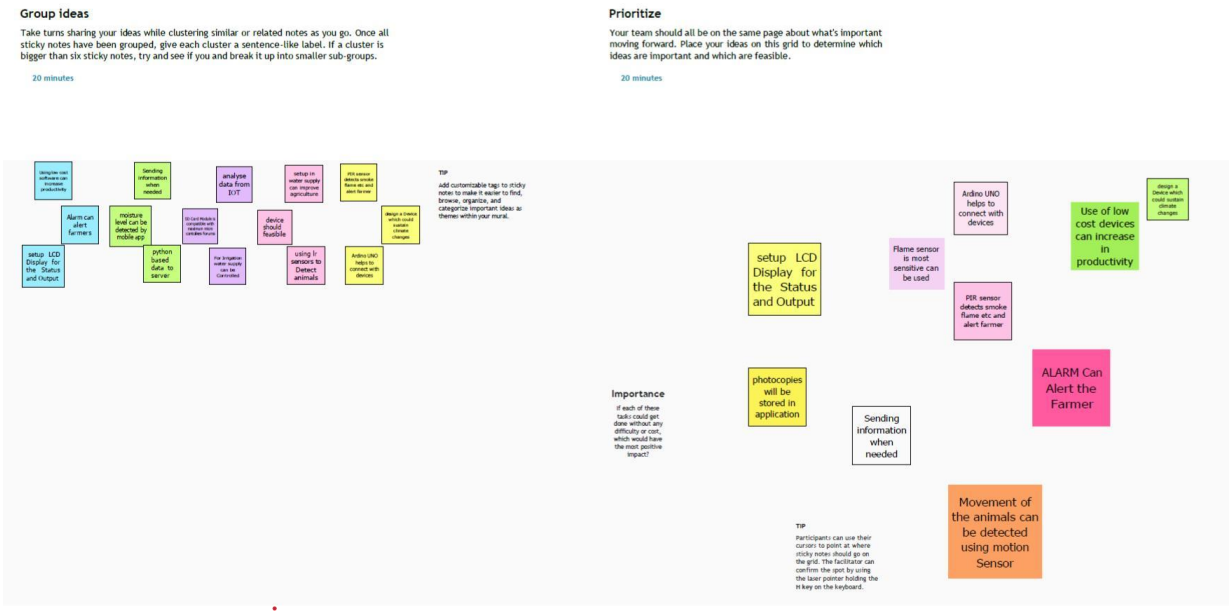
EMPATHY MAP CANVAS:

a.



IDEATION AND BRAINSTORMING:



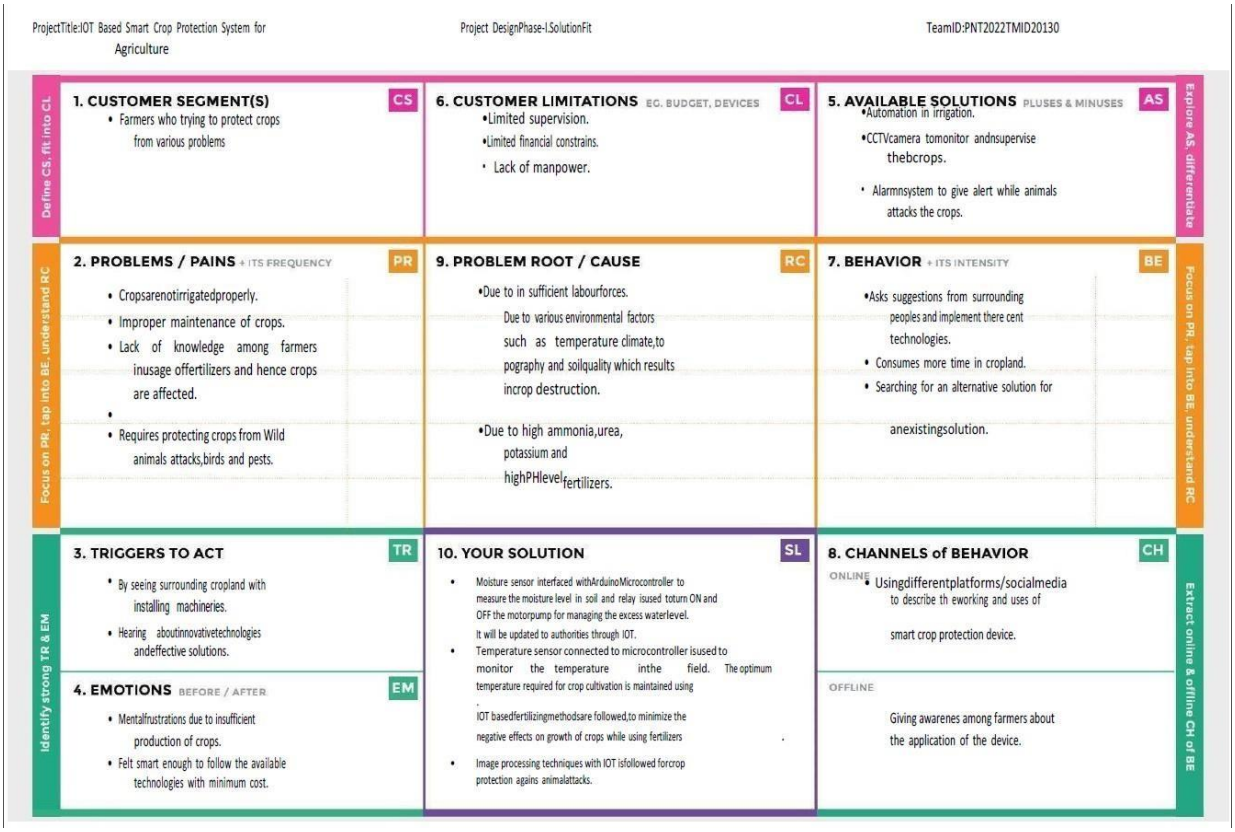


PROPOSED SOLUTION:

S.NO.	Parameter	Description
1.	Problem Statement. (Problem to be solved)	✓ Crops are not irrigated properly due to insufficient labour forces. ✓ Improper maintenance of crops against various environmental factors such as temperature climate, topography and soil quantity which results in crop destruction. ✓ Requires protecting crops from wild animals attacks birds and pests.
2.	Idea /Solution Description.	✓ Moisture sensor is interfaced with Arduino Microcontroller to measure the moisture level in soil and relay is used to turn ON & OFF the motor pump for managing the excess water level. It will be updated to authorities through IOT. ✓ Temperature sensor connected to microcontroller is used to monitor the temperature in the field. ✓ Image processing techniques with IOT is followed for crop protection against animal attack.
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:



REQUIREMENT ANALYSIS

FUNCTIONAL REQUIREMENT:

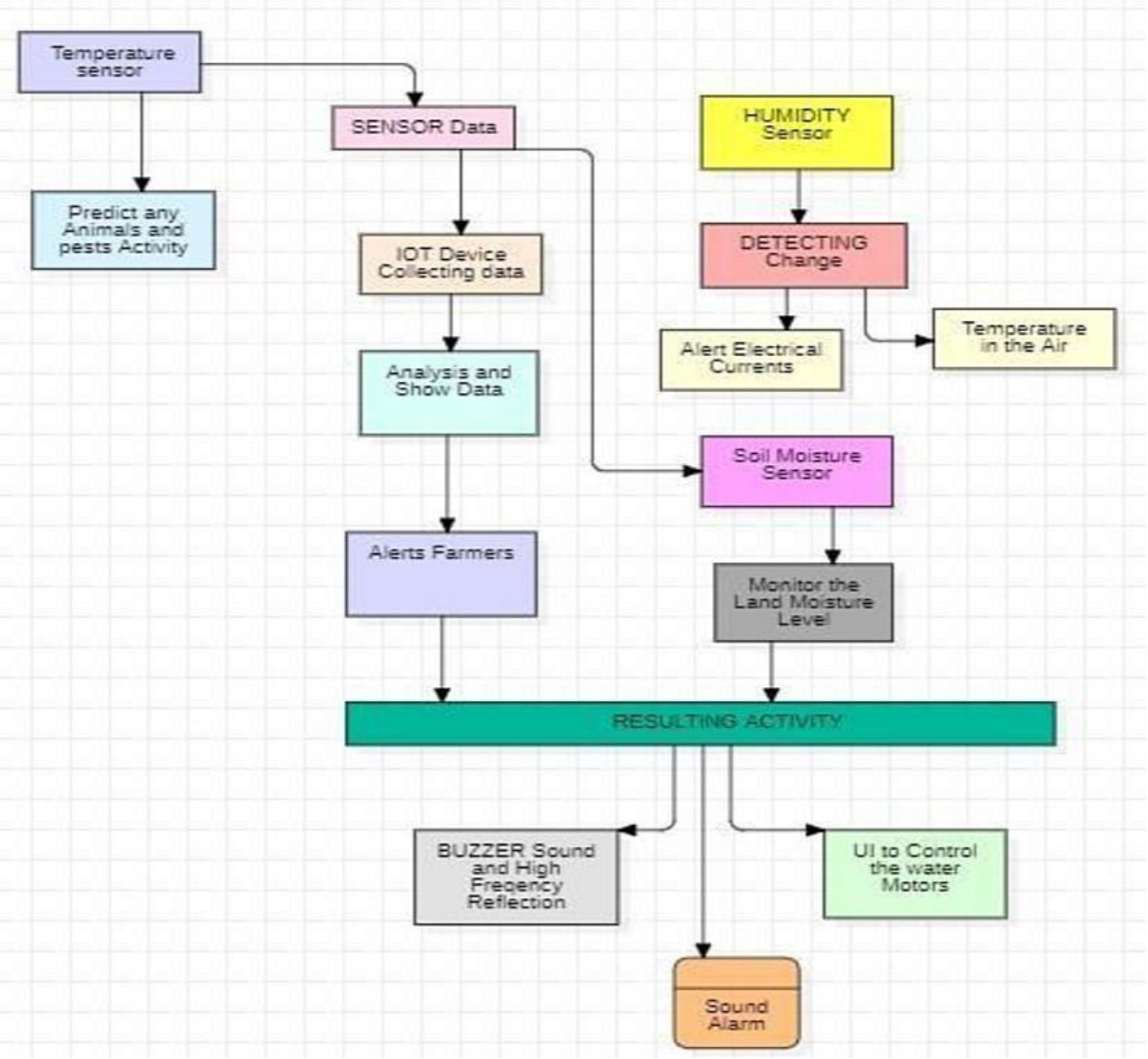
S.NO.	Functional Requirement.	Sub Requirement.
1.	User Visibility	Sense animals nearing the crop field & sounds alarm to woo them away as well as sends SMS to farmer using cloud service.
2.	User Reception	The Data like values of Temperature, Humidity, Soil moisture Sensors are received via SMS.
3.	User 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 FUNCTINAL REQUIREMENT:

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 productiondon'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:

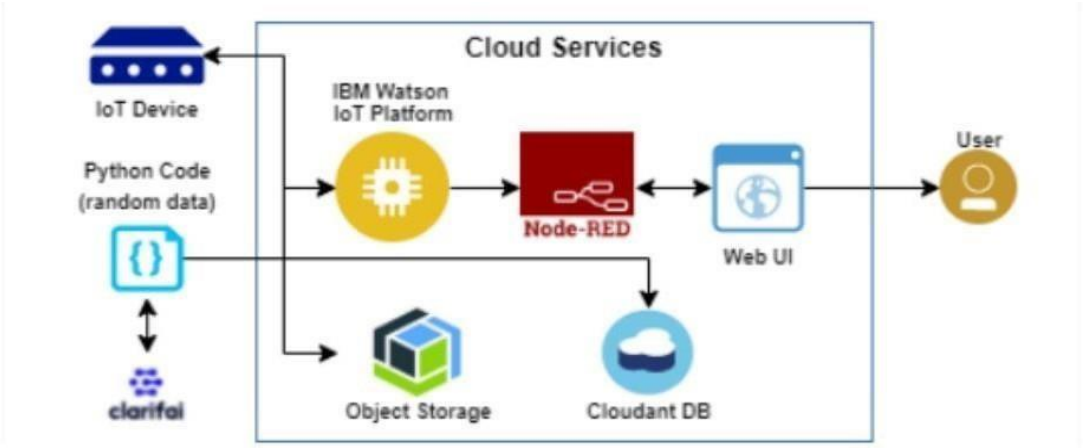


TABLE-1:

sno	components	description	Technology
1	User interface	Interacts with iot device	Html,css,angular js etc..
2	Application logic-1	Logic for a process in the application	Python
3	Application logic-2	Logic for process in the application	Clarifai
4	Application logic-3	Logic for process in the application	IBM Waston Iot platform
5	Application logic-4	logic for the process	Node red app service
6	User friendly	Easily manage the net screen appliance	Web uI

TABLE-2: APPLICATION AND CHARACTERISTICS

sno	Characteristics	Description	Technology
1	Open source framework	Open source framework used	Python
2	Security implementations	Authentication using encryption	Encryptions
3	Scalable architecture	The scalability of architecture consists of 3 models	Web UI Application server-python, clarifai Database server-ibm cloud services.
4	Availability	It is increased by cloudant database	IBM cloud services

USER STORIES:

SPRINT	FUNCTIONAL REQUIREMENT	USER STORY NUMBER	USER STORY/TASK	STORY POINTS	PRIORITY
Sprint-1		US-1	Create the IBM Cloud services which are being used in this project.	7	high
Sprint-1		US-2	Create the IBM Cloud services which are being used in this project.	7	high
Sprint-2		US-3	IBM Watson IoT platform acts as the mediator to connect the web application to IoT devices, so create the IBM Watson IoT platform.	5	medium
Sprint-2		US-4	In order to connect the IoT device to the IBM cloud, create a device in the IBM Watson IoT platform and get the device credentials	6	high
Sprint-3		US-1	Configure the connection security and create API keys that are used in the Node-RED service for accessing the IBM IoT Platform.	10	high
Sprint-3		US-3	Create a Node-RED service	8	high
Sprint-3		US-2	Develop a python script to publish random sensor data such as temperature, moisture, soil and humidity to the IBM IoT platform	6	medium
Sprint-3		US-1	After developing python code, commands are received just print the statements which represent the control of the devices.	8	high
Sprint-4		US-3	Publish Data to The IBM Cloud	5	high
Sprint-4		US-2	Create Web UI in Node- Red	8	high
Sprint-4		US-1	Configure the Node-RED flow to receive data from the IBM IoT platform and also use Cloudant DB nodes to store the received sensor data in the cloudant DB	6	high

PROJECT PLANNING AND SCHEDULING

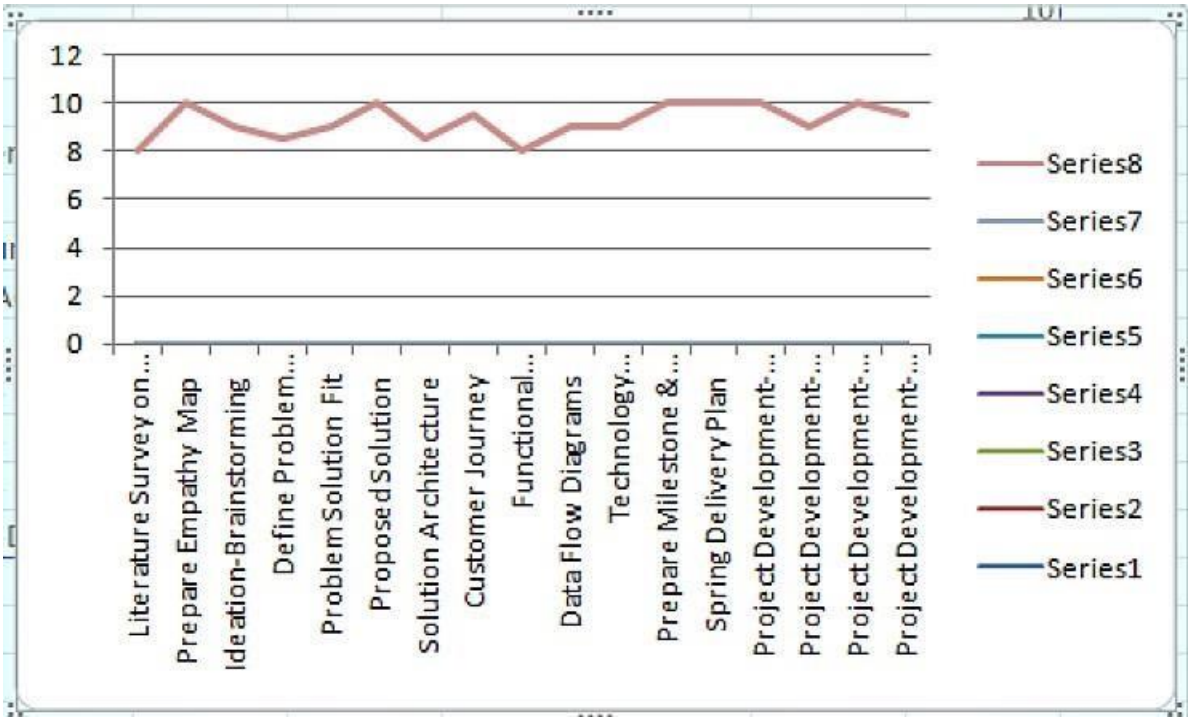
SPRINT PLANNINGAND ESTIMATION:

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)

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



CODING AND SOLUTIONING
FEATURE-1

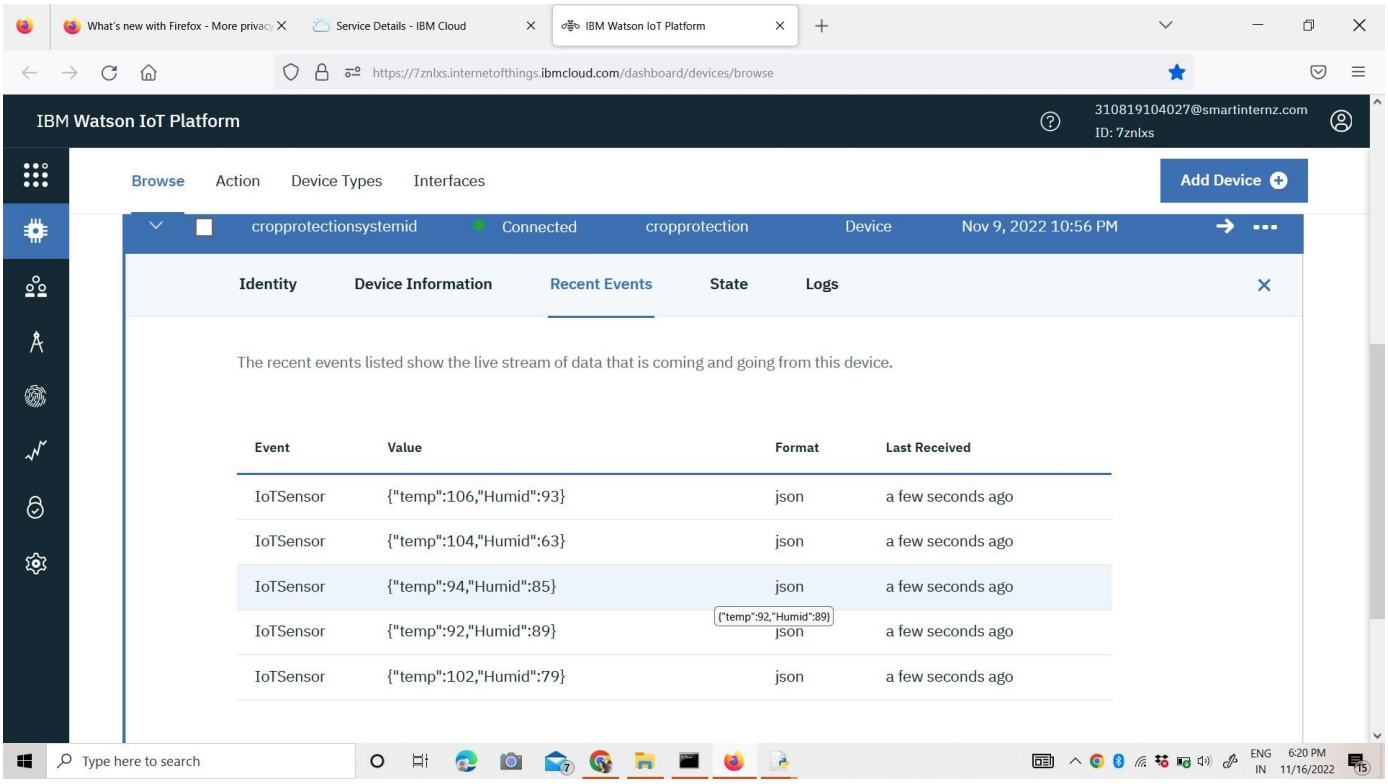
```
import random
import
ibmiotf.applicatio
n
import
ibmiotf.device
from time import
sleep import sys
#IBM Watson Device Credentials.
organization = "7znlxs" deviceType =
"cropprotection" deviceId =
"cropprotectionsystemid" authMethod =
"token" authToken =
"ejrRfZRywhhZCz!mUR" def
myCommandCallback(cmd):
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
succes
s:
  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
succes
s:
  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
succes
s:
  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()
```

OUTPUT:



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*: ≤30mA

- Sound Output 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 vehicles such as ambulances, police cars and fire trucks. There are two general types, pneumatic and electronic.

FEATURE-2:

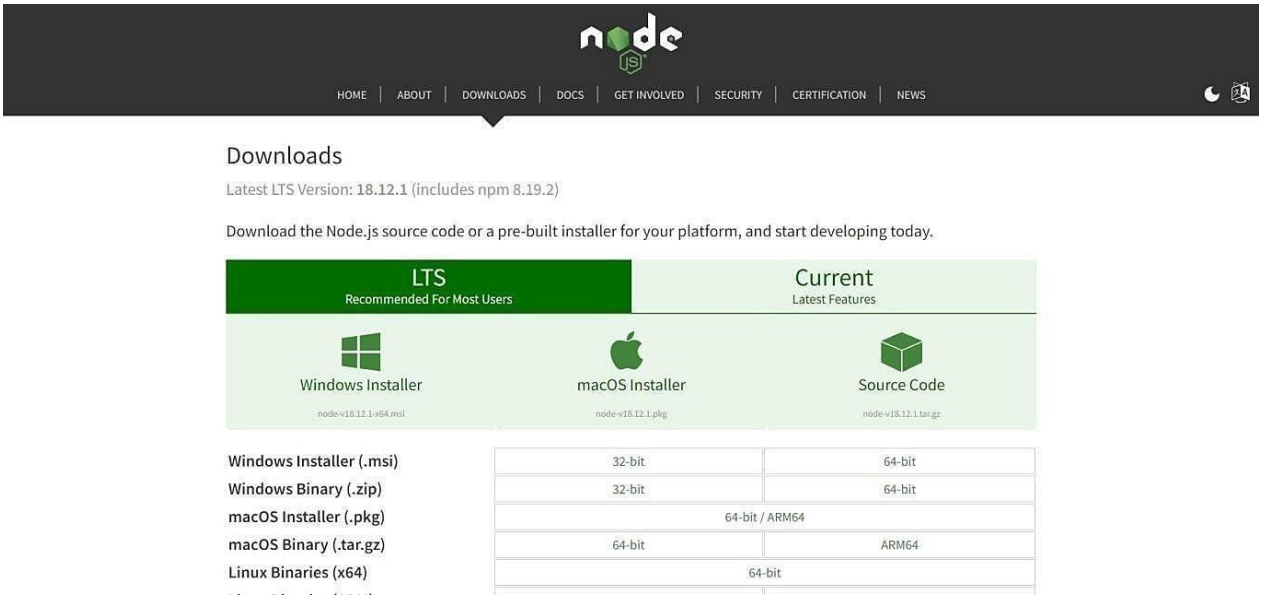
i. Good sensitivity to Combustible gas in wide range.
ii. High sensitivity to LPG, Propane and Hydrogen.
iii. Long life and low cost.
iv. Simple drive circuit.

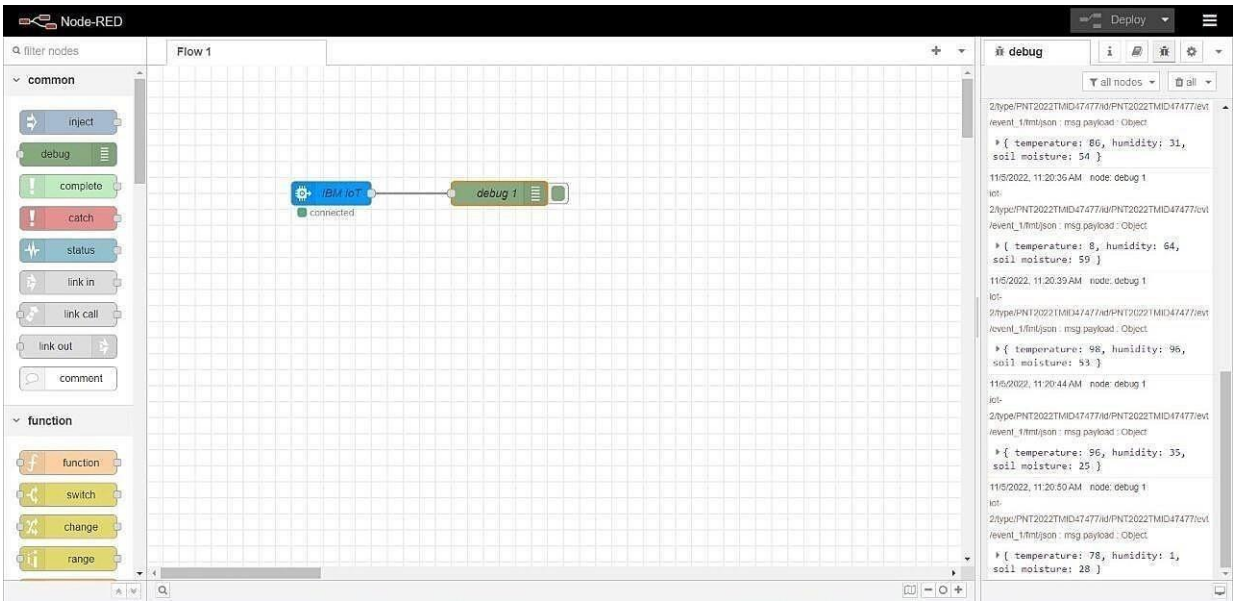
TESTING

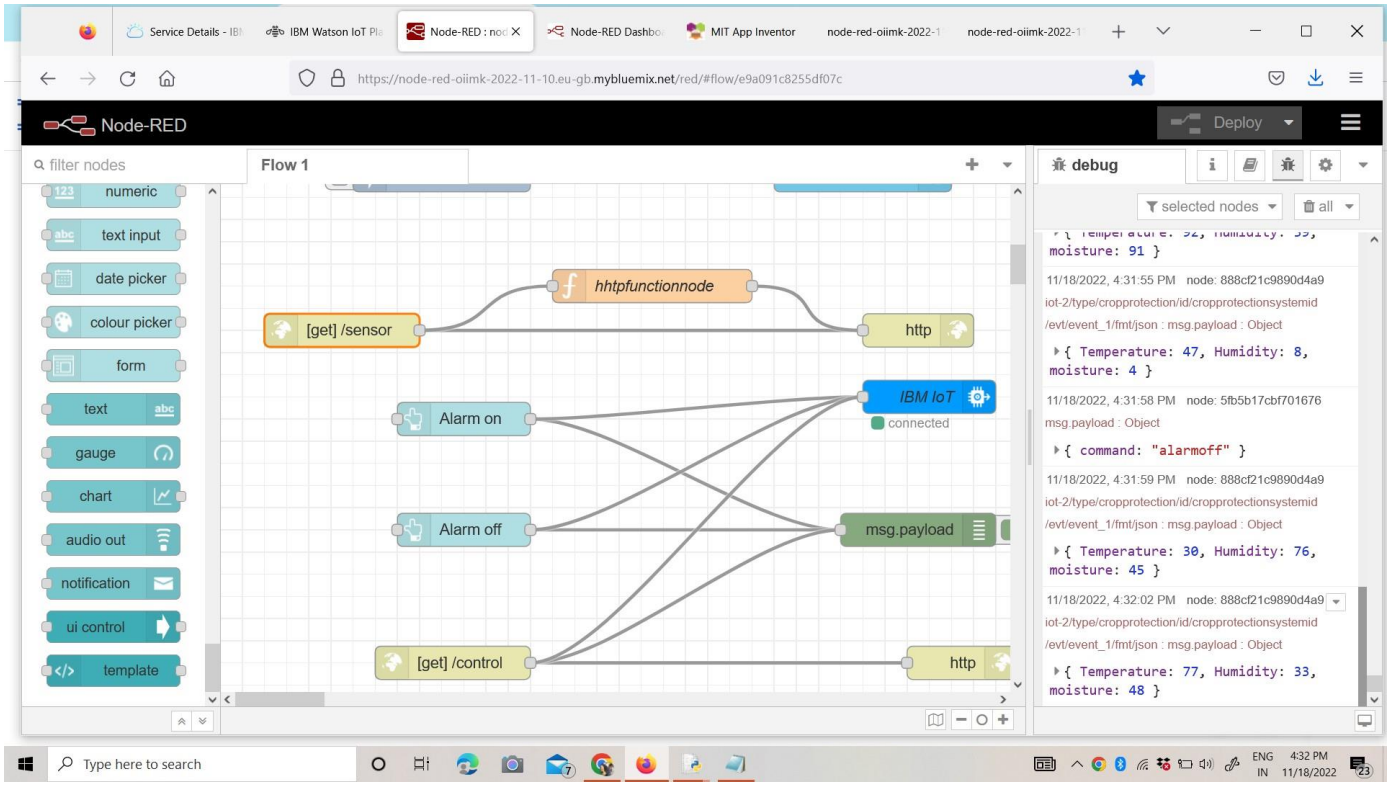
TEST CASES:

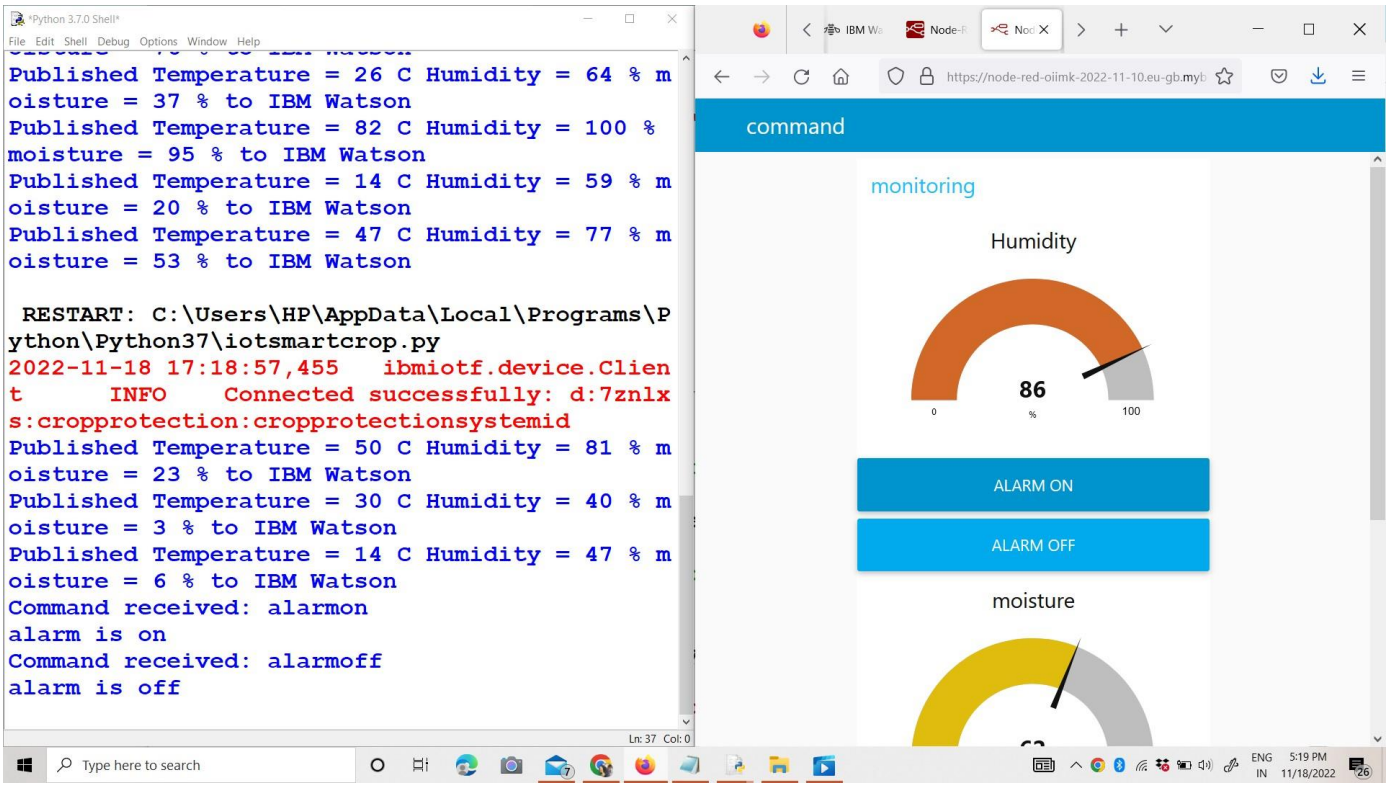
S no	parameter	Values
1	Model summary	-
2	accuracy	Training accuracy- 95% Validation accuracy- 72%
3	Confidence score	Class detected- 80% Confidence score-80%

User Acceptance Testing:




Node-RED
Deploy
filter nodes
Flow 1
common
inject
debug
complete
catch
status
link in
link call
link out
comment
function
function
switch
change
range
IBM IoT
connected
debug 1
debug
all nodes
all
{ temperature: 86, humidity: 31, soil moisture: 54 }
{ temperature: 8, humidity: 64, soil moisture: 59 }
{ temperature: 98, humidity: 96, soil moisture: 53 }
{ temperature: 96, humidity: 35, soil moisture: 25 }
{ temperature: 78, humidity: 1, soil moisture: 28 }


Service Details - IB
IBM Watson IoT Pla
Node-RED : noc
Node-RED Dashbo
MIT App Inventor
node-red-oiimk-2022-1
node-red-oiimk-2022-1
https://node-red-oiimk-2022-11-10.eu-gb.mybluemix.net/red/#flow/e9a091c8255df07c
Node-RED
Deploy
filter nodes
Flow 1
numeric
text input
date picker
colour picker
form
text
gauge
chart
audio out
notification
ui control
template
hhtpfunctionnode
[get] /sensor
http
IBM IoT
connected
Alarm on
Alarm off
msg.payload
[get] /control
http
debug
selected nodes
all
moisture: 91 }
11/18/2022, 4:31:55 PM node: 888cf21c9890d4a9
iot-2/type/cropprotection/id/cropprotectionsystemid
/evt/event_1/fmt/json : msg.payload : Object
{ Temperature: 47, Humidity: 8, moisture: 4 }
11/18/2022, 4:31:58 PM node: 5fb5b17cbf701676
msg.payload : Object
{ command: "alarmoff" }
11/18/2022, 4:31:59 PM node: 888cf21c9890d4a9
iot-2/type/cropprotection/id/cropprotectionsystemid
/evt/event_1/fmt/json : msg.payload : Object
{ Temperature: 30, Humidity: 76, moisture: 45 }
11/18/2022, 4:32:02 PM node: 888cf21c9890d4a9
iot-2/type/cropprotection/id/cropprotectionsystemid
/evt/event_1/fmt/json : msg.payload : Object
{ Temperature: 77, Humidity: 33, moisture: 48 }
Type here to search
ENG
IN
4:32 PM
11/18/2022



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 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 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 importsys import ibmiotf.application # toinstallpip
install ibmiotf importibmiotf.device


# Provide your IBM Watson Device Credentials organization =
"7znlxs" # replace the ORG ID deviceType = "cropprotection" #replace
the Device type deviceId = "cropprotectionsystemid" # replace Device
ID authMethod =
"token" authToken = "ejrRfZRywhhZCz!mUR" # Replace the authtoken


def myCommandCallback(cmd): # function for Callbackif


    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
  requiredinformation: 'interval'")


  interval = cmd.data['interval']

elif cmd.command == "print":
if 'message' not in cmd.data:
        print("Error - commandis missing requiredinformation:
        '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)#
  .....
...

  exceptException 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 oftype
                                                                                    "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
    sysimport
    ibmiotf.application
    importibmiotf.device
  import random


    # Provide your IBM Watson Device Credentials organization =
    "7znlxs" # replace the ORG ID deviceType = "cropprotection" #replace
    the Device type deviceId = "cropprotectionsystemid" # replace Device
    ID authMethod = "token" authToken = "ejrRfZRywhhZCz!mUR" #
    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)
        #.....
...

exceptException 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 oftype
  "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":"registe
re 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","a71f164bc378bcf1"]]
},
{
"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":"",
"classNam
e":"",
"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 }

GitHub & Project Demo Link

https://github.com/IBM-EPBL/IBM-Project-35368-660284332