IOT BASED SMART CROP PROTECTION USING IOT

TEAM ID : PNT2022TMID02468

TEAM MEMBERS:

ARVINTH SREERAM R S (190801015)

GOKULNATH A (190801050)

AKASH M (190801007)

KISHORE P (190801091)

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 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 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:

1.Mr. Pranav shitap, Mr. Jayesh redj , Mr .Shikhar Singh, Mr. Durvesh Zagade, Dr. Sharada Chougule. Department of ELECTRONICS AND TELECOMMUNICATION ENGINEERING, Finolex Academy of Management and technology, ratangir i, India.

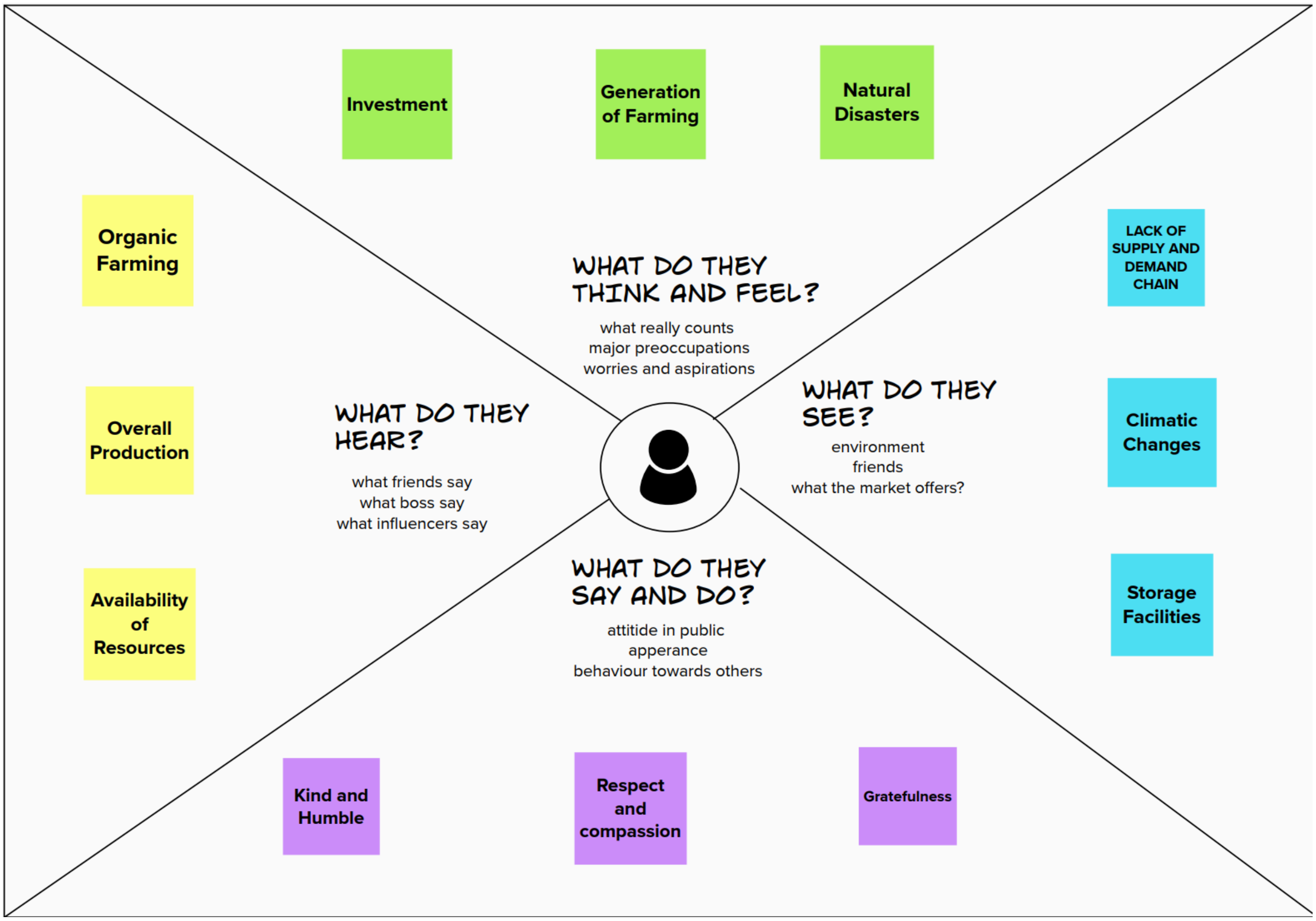
PROBLEM STATEMENT DEFINITION STATEMENT:

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

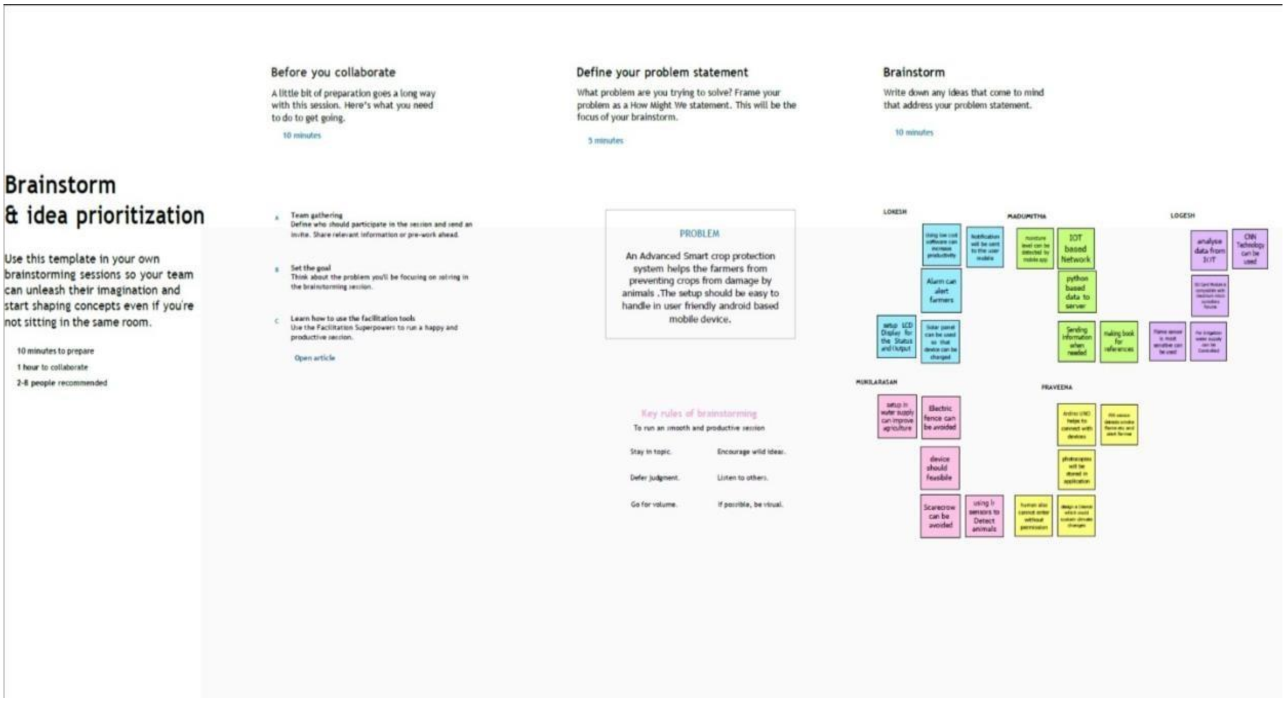
In spite of economic development agriculture is the backbone of the economy. Crops in forms are many times ravaged by local animals like buffaloes, cows, goats, birds and fire etc. this leads to huge loss for the farmers.it is not possible for farmers to blockade to entire fields or stay 24 hours and 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:



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

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article

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

An Advanced Smart crop protection system helps the farmers from preventing crops from damage by animals .The setup should be easy to handle in user friendly android based mobile device.

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.

Brainstorm

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

10 minutes

LOKESH

Using low cost software can increase productivity

Notification will be sent to the user mobile

Alarm can alert farmers

setup LCD Display for the Status and Output

Solar panel can be used so that device can be charged

MADUMITHA

moisture level can be detected by mobile app

IOT based Network

python based data to server

Sending information when needed

making book for references

LOGESH

analyse data from IOT

CNN Technology can be used

SD card Module compatible with [illegible]

Flame sensor is most sensitive can be used

For irrigation water supply can be Controlled

MUKILARASAN

setup in water supply can improve agriculture

Electric fence can be avoided

device should feasible

Scarecrow can be avoided

using Ir sensors to Detect animals

PRAVEENA

Arduino UNO helps to connect with devices

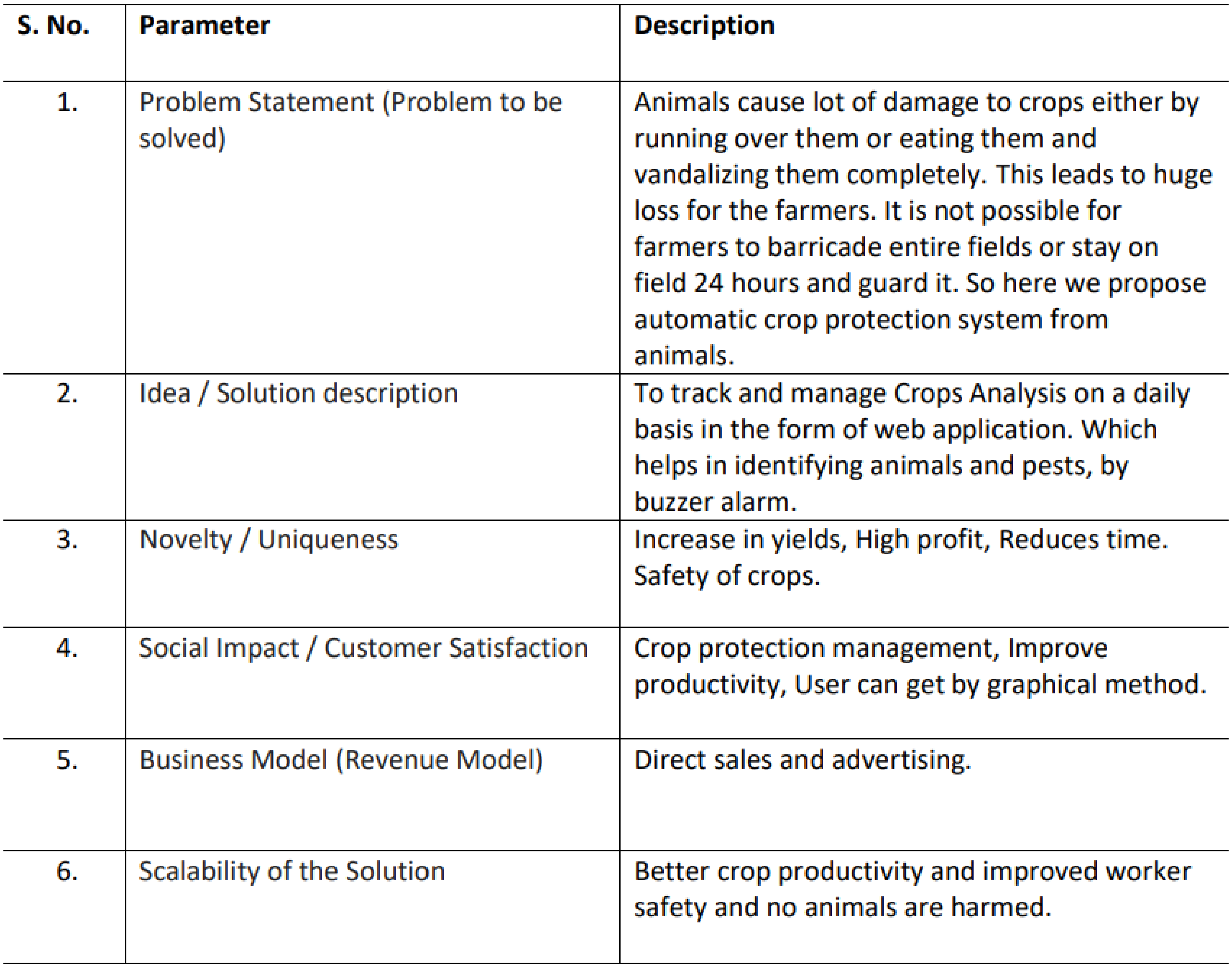
PIR sensor detects smoke flame etc and alert farmer

photocopies will be stored in application

human also cannot enter without permission

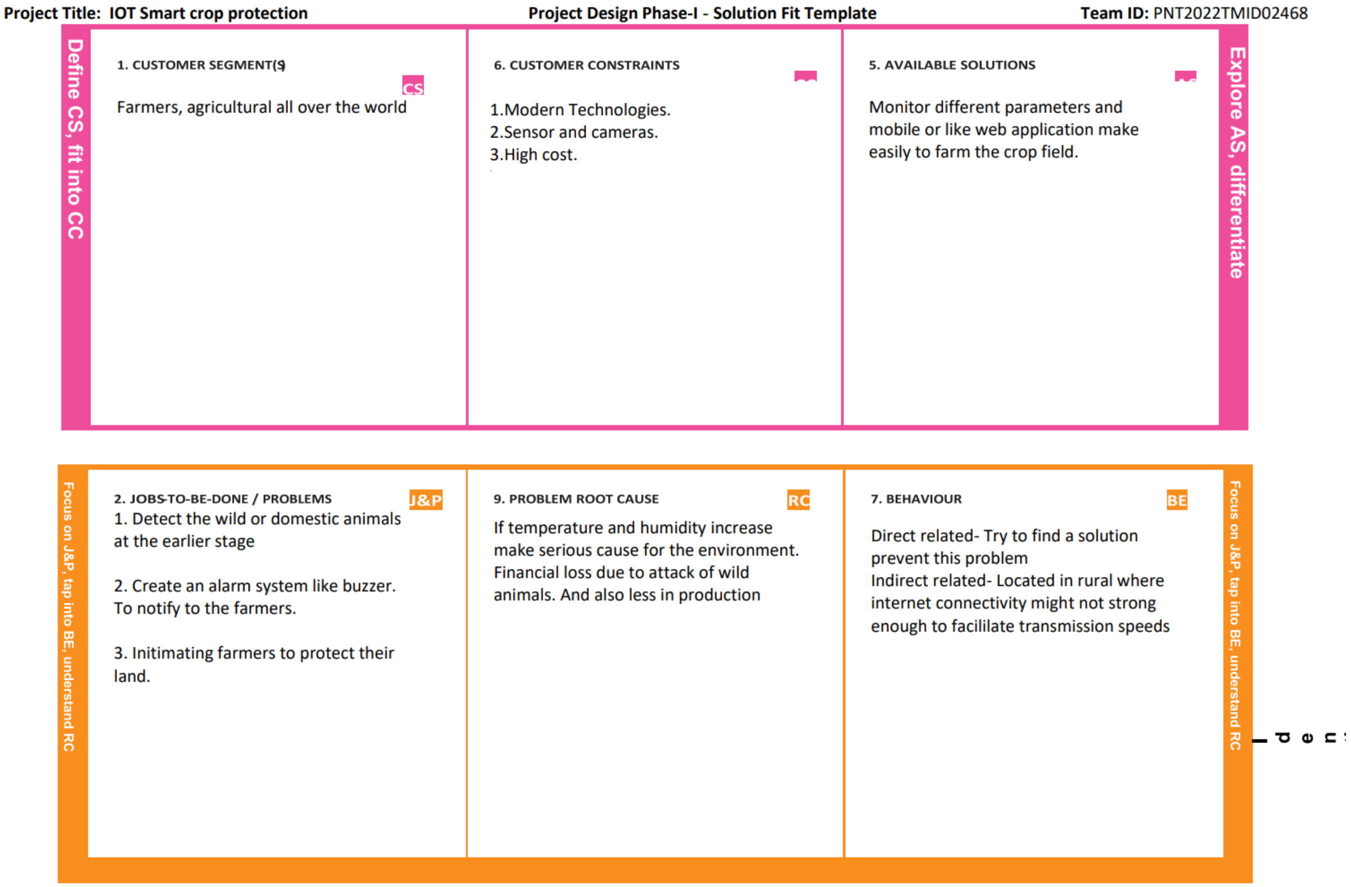
design a device which could sustain climate changes

PROPOSED SOLUTION:



S. No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Animals cause lot of damage to crops either by running over them or eating them and vandalizing them completely. This leads to huge loss 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.
2.	Idea / Solution description	To track and manage Crops Analysis on a daily basis in the form of web application. Which helps in identifying animals and pests, by buzzer alarm.
3.	Novelty / Uniqueness	Increase in yields, High profit, Reduces time. Safety of crops.
4.	Social Impact / Customer Satisfaction	Crop protection management, Improve productivity, User can get by graphical method.
5.	Business Model (Revenue Model)	Direct sales and advertising.
6.	Scalability of the Solution	Better crop productivity and improved worker safety and no animals are harmed.

PROBLEM SOLUTIONFIT:



Project Title: IOT Smart crop protection **Project Design Phase-I - Solution Fit Template** **Team ID:** PNT2022TMID02468

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Farmers, agricultural all over the world

6. CUSTOMER CONSTRAINTS

1.Modern Technologies.
2.Sensor and cameras.
3.High cost.

5. AVAILABLE SOLUTIONS

Monitor different parameters and mobile or like web application make easily to farm the crop field.

Explore AS, differentiate

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

2. JOBS-TO-BE-DONE / PROBLEMS J&P

1. Detect the wild or domestic animals at the earlier stage

2. Create an alarm system like buzzer. To notify to the farmers.

3. Initimating farmers to protect their land.

9. PROBLEM ROOT CAUSE RC

If temperature and humidity increase make serious cause for the environment. Financial loss due to attack of wild animals. And also less in production

7. BEHAVIOUR BE

Direct related- Try to find a solution prevent this problem
Indirect related- Located in rural where internet connectivity might not strong enough to facililate transmission speeds

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

3. TRIGGERS TR

1.Information from neighbour farmers.

2.Occurrence attack of wild animals.
3.To lift people out of poverty in developing nations.
4.Dosen't know whether the application work properly or not.

4. EMOTIONS: BEFORE / AFTER EM

Before- Heavy work overload, Wild animals create stress to farmers , and finances.
After- Easy to make more yield in field.

10. YOUR SOLUTION SL

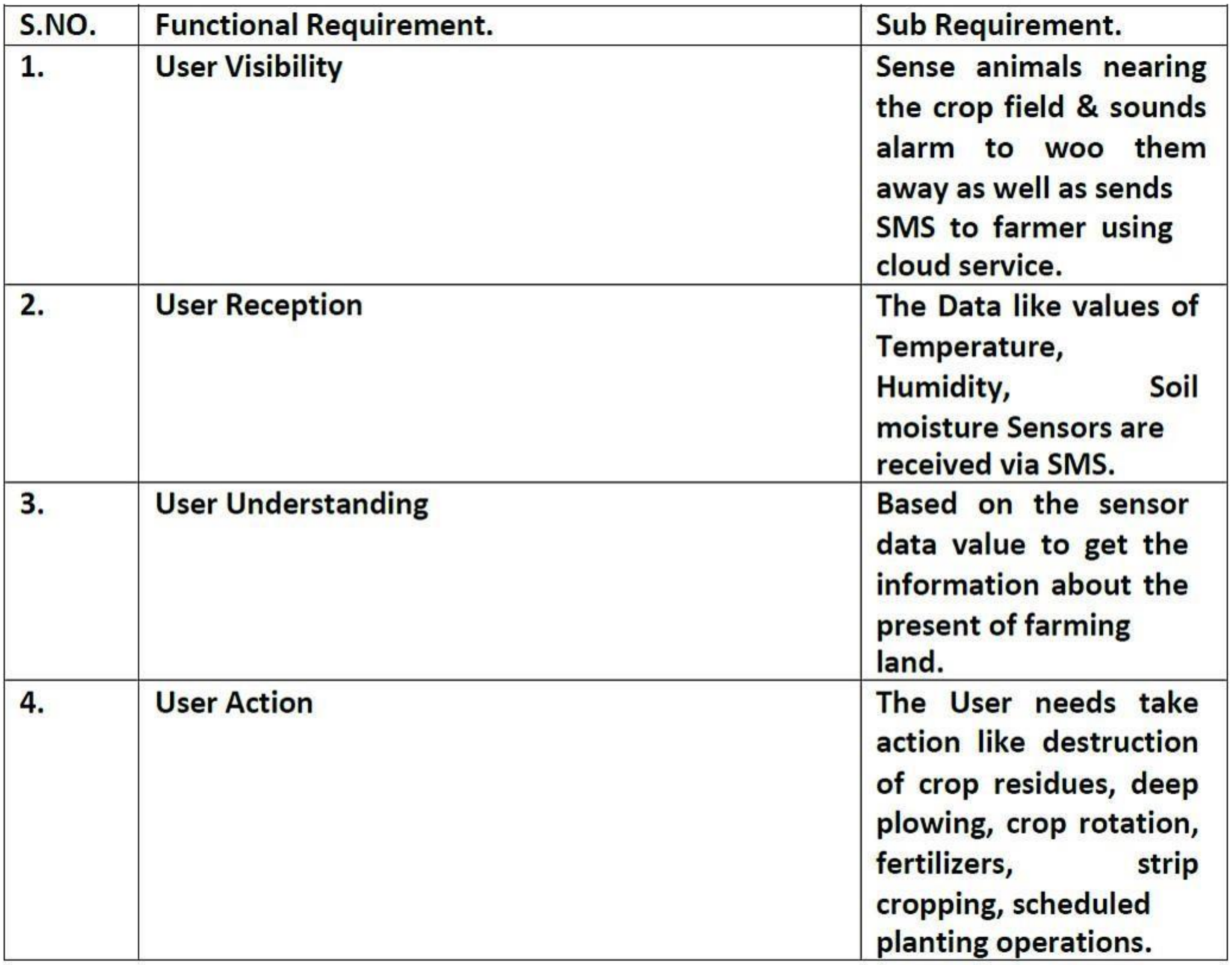
!SMART CROP PROTECTION!
It help farmers grow more yield in Field.
Protection crops from pests and diseases
Protecting crops from wild animals using sensors.

8. CHANNELS of BEHAVIOUR CH

ONLINE-Data send through application for farmers to know about the farmers.
OFFLINE-The control action taken by the farmers immediately to the spot.

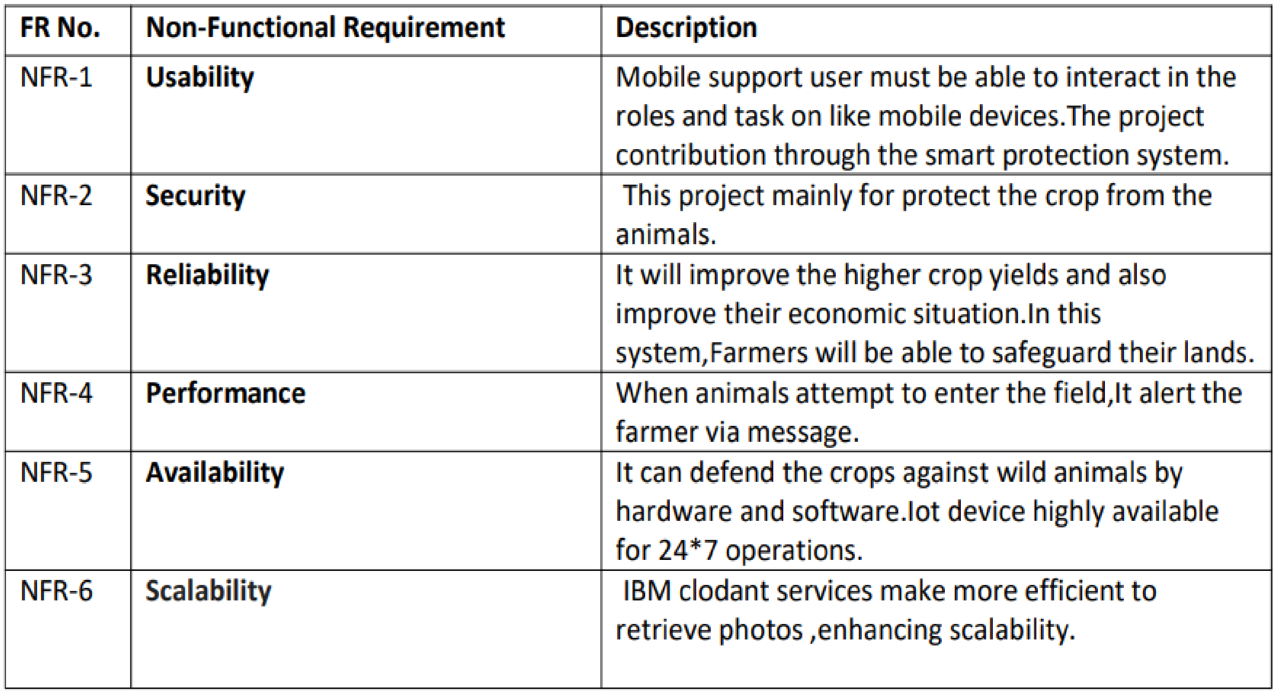
REQUIREMENT ANALYSIS

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.

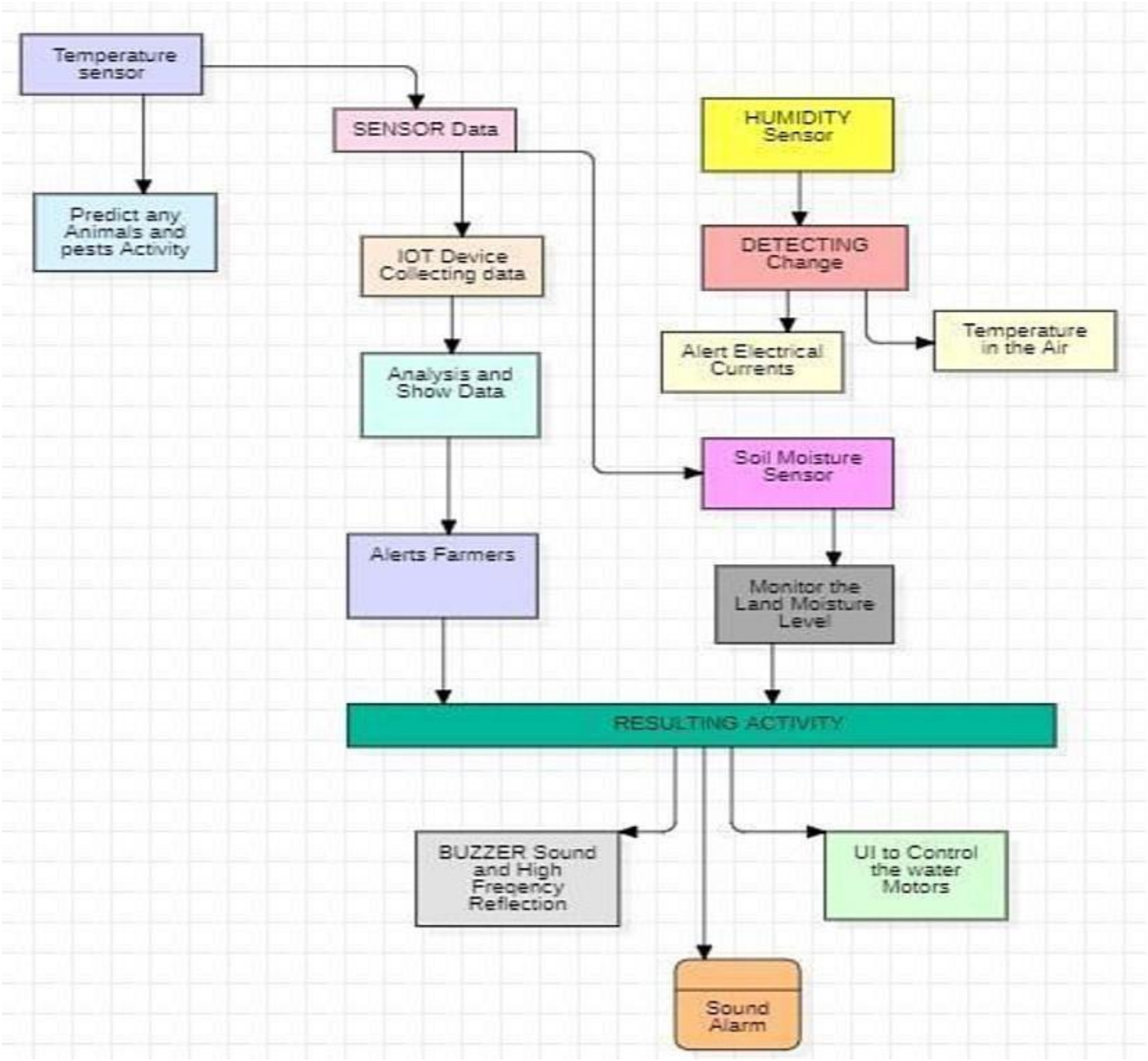
NON FUNCTIONAL REQUIREMENT:



FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Mobile support user must be able to interact in the roles and task on like mobile devices.The project contribution through the smart protection system.
NFR-2	**Security**	This project mainly for protect the crop from the animals.
NFR-3	**Reliability**	It will improve the higher crop yields and also improve their economic situation.In this system,Farmers will be able to safeguard their lands.
NFR-4	**Performance**	When animals attempt to enter the field,It alert the farmer via message.
NFR-5	**Availability**	It can defend the crops against wild animals by hardware and software.Iot device highly available for 24*7 operations.
NFR-6	**Scalability**	IBM clodant services make more efficient to retrieve photos ,enhancing scalability.

PROJECT DESIGN

DATA FLOW DIAGRAM:

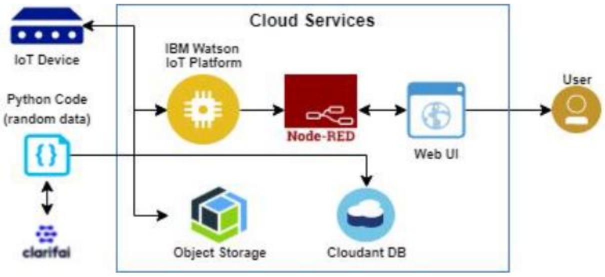


Temperature sensor
SENSOR Data
HUMIDITY Sensor
Predict any Animals and pests Activity
IOT Device Collecting data
DETECTING Change
Alert Electrical Currents
Temperature in the Air
Analysis and Show Data
Soil Moisture Sensor
Alerts Farmers
Monitor the Land Moisture Level
RESULTING ACTIVITY
BUZZER Sound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

SOLUTION AND TECHNICAL ARCHITECTURE:

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.



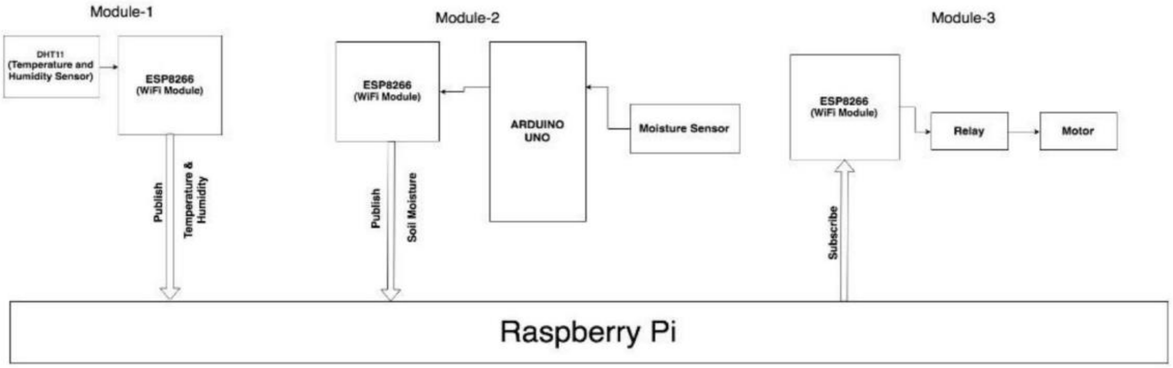
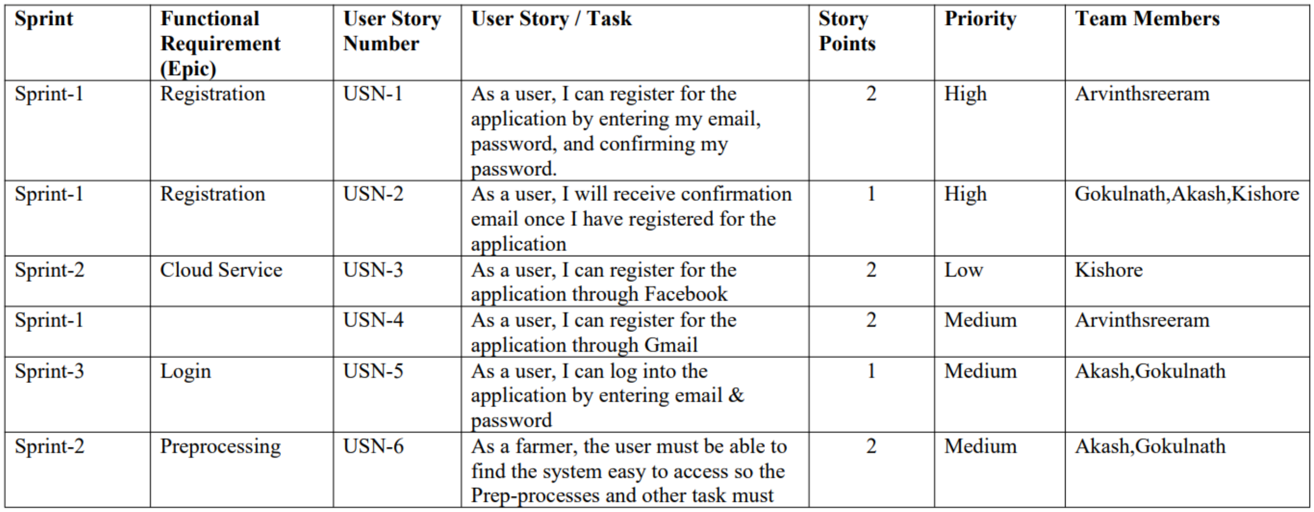


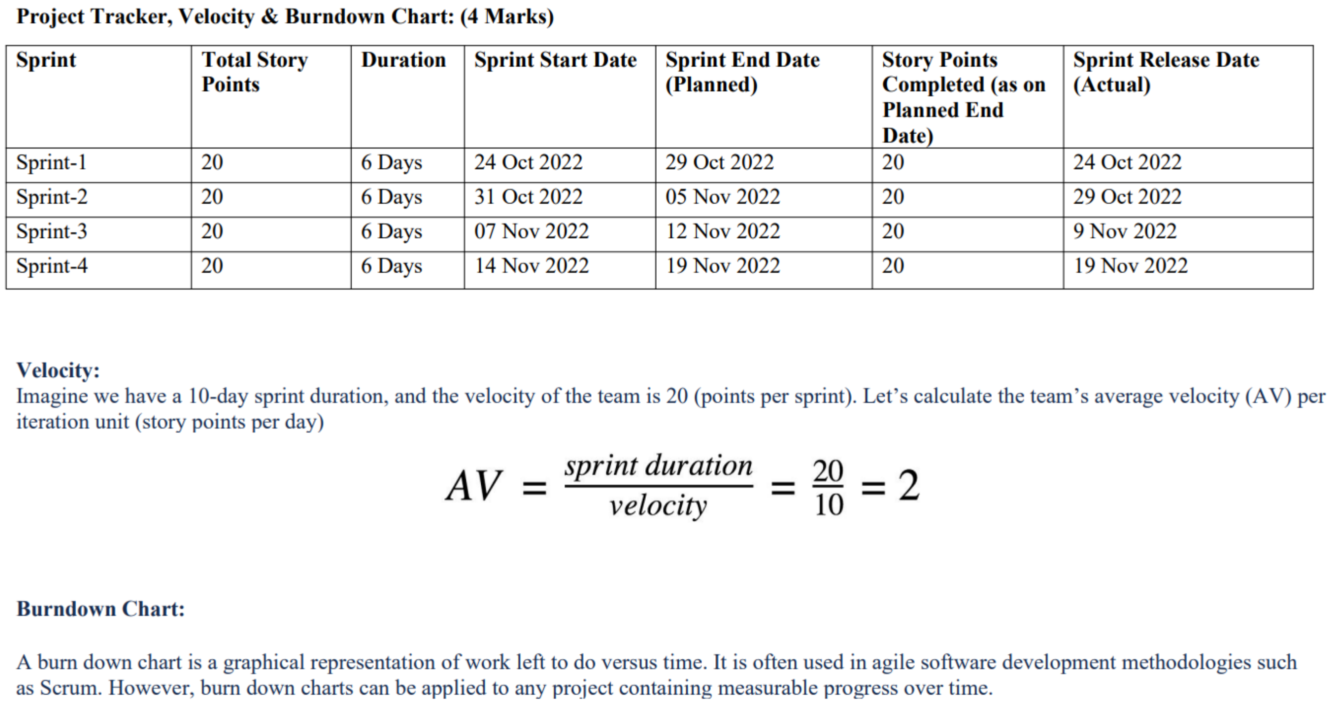
Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Technology used

PROJECT PLANNING AND SCHEDULING:



Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Arvinthsreeram
Sprint-1	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Gokulnath,Akash,Kishore
Sprint-2	Cloud Service	USN-3	As a user, I can register for the application through Facebook	2	Low	Kishore
Sprint-1		USN-4	As a user, I can register for the application through Gmail	2	Medium	Arvinthsreeram
Sprint-3	Login	USN-5	As a user, I can log into the application by entering email & password	1	Medium	Akash,Gokulnath
Sprint-2	Preprocessing	USN-6	As a farmer, the user must be able to find the system easy to access so the Prep-processes and other task must	2	Medium	Akash,Gokulnath



Project Tracker, Velocity & Burndown Chart: (4 Marks)

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	24 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	29 Oct 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	9 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

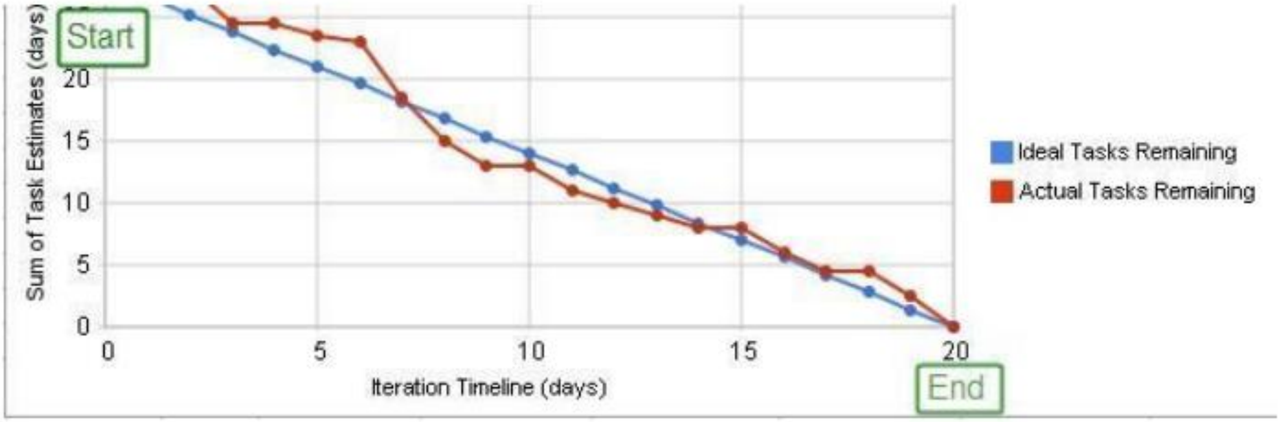
Velocity:

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 (AV) per iteration unit (story points per day)

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

Burndown Chart:

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


Start
Sum of Task Estimates (days)
20
15
10
5
0
0
5
10
15
20
Iteration Timeline (days)
End
🟦 Ideal Tasks Remaining
🟥 Actual Tasks Remaining

CODING AND SOLUTIONING

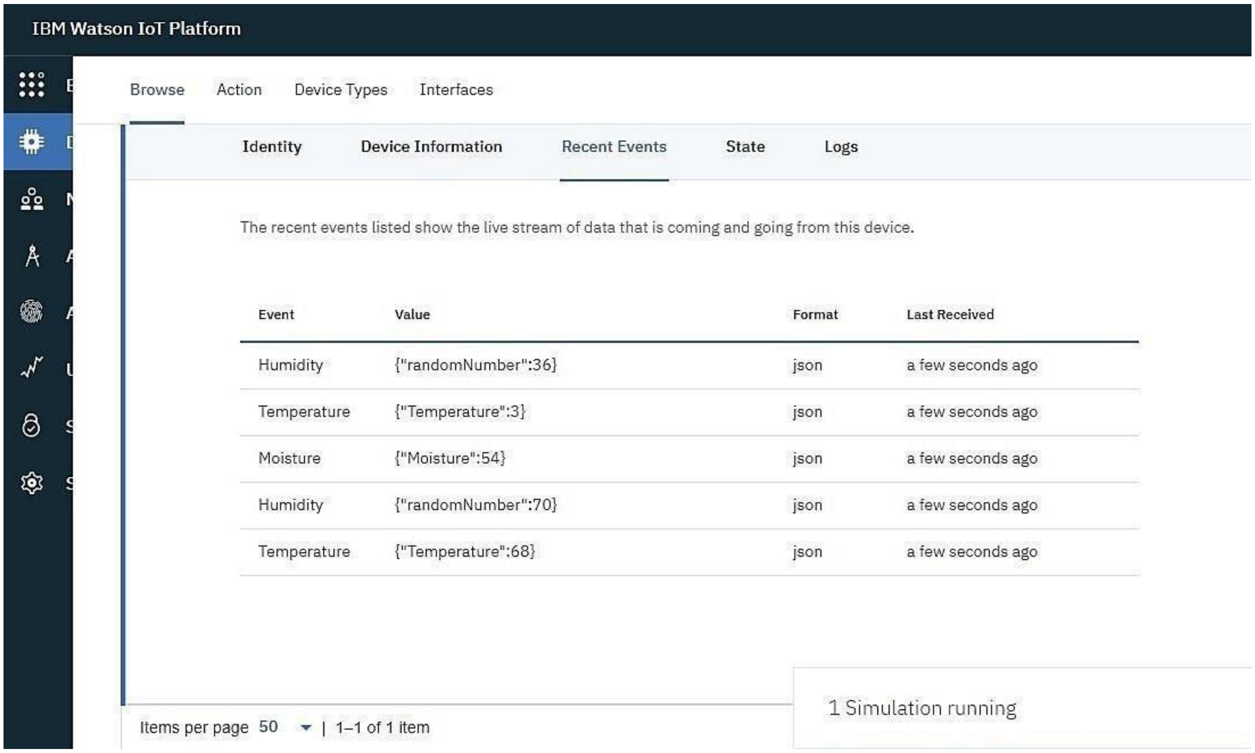
FEATURE-1

```
import random import
ibmiotf
.application import
ibmiotf.device from
time import
sleep import sys
my config ={
“identify”:{
“orgId”:”it3aoz,
“typeId”:”ESP-32”,
“deviceId”:”2731”
},
“auth”:{
“token”:”87654321”
}
}
print("Command received: %s" %
cmd.data['command'])
status=cmd.data['command']  if
status=="sprinkler_on":     print ("sprinkler is
ON")  else :     print ("sprinkler is OFF")
 #print(cmd)


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

```
deviceCli.connect() while
True:
#Getting values from sensors.
temp_sensor = round(
random.uniform(0,80),2)  PH_sensor
= round(random.uniform(1,14),3)
 camera = ["Detected","Not Detected","Not Detected","Not Detected","Not
Detected","Not Detected",]  camera_reading = random.choice(camera)  flame =
["Detected","Not Detected","Not Detected","Not Detected","Not
Detected","Not Detected",]  flame_reading = random.choice(flame)  moist_level
= round(random.uniform(0,100),2)
 water_level = round(random.uniform(0,30),2)

#storing the sensor data to send in json format to cloud.

 temp_data = { 'Temperature' :
temp_sensor }  PH_data = { 'PH Level' :
PH_sensor }  camera_data = { 'Animal
attack' : camera_reading}  flame_data =
{ 'Flame' :
flame_reading }  moist_data = { 'Moisture
Level' : moist_level}   water_data = {
'Water Level' : water_level}

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

 success = deviceCli.publishEvent("PH sensor", "json",
PH_data, qos=0)  sleep(1)  if success:     print
```

```
("Published PH Level = %s" % PH_sensor, "to IBM
Watson")

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

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

 if (temp_sensor > 35):     print("sprinkler-1 is ON")  success = deviceCli.publishEvent("Alert1", "json",{
'alert1' : "Temperature(%s) is high, sprinkerlers are turned ON" %temp_sensor }  , qos=0)  sleep(1)  if
success:     print( 'Published alert1 : ', "Temperature(%s) is high, sprinkerlers are turned ON"
%temp_sensor,"to IBM Watson")  print("") else:
 print("sprinkler-1 is OFF")
print("")

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

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

```
 #To send alert message to farmer that animal attack on crops.


 if (camera_reading == "Detected"):      success = deviceCli.publishEvent("Alert3",
"json", { 'alert3' : "Animal attack on crops detected" }, qos=0)  sleep(1)  if success:
print('Published alert3 : ' , "Animal attack on crops detected","to IBM Watson","to
IBM Watson")  print("")
 #To send alert message if flame detected on crop land and turn ON the splinkers to take immediate
action.


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


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



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

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

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 defence 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:

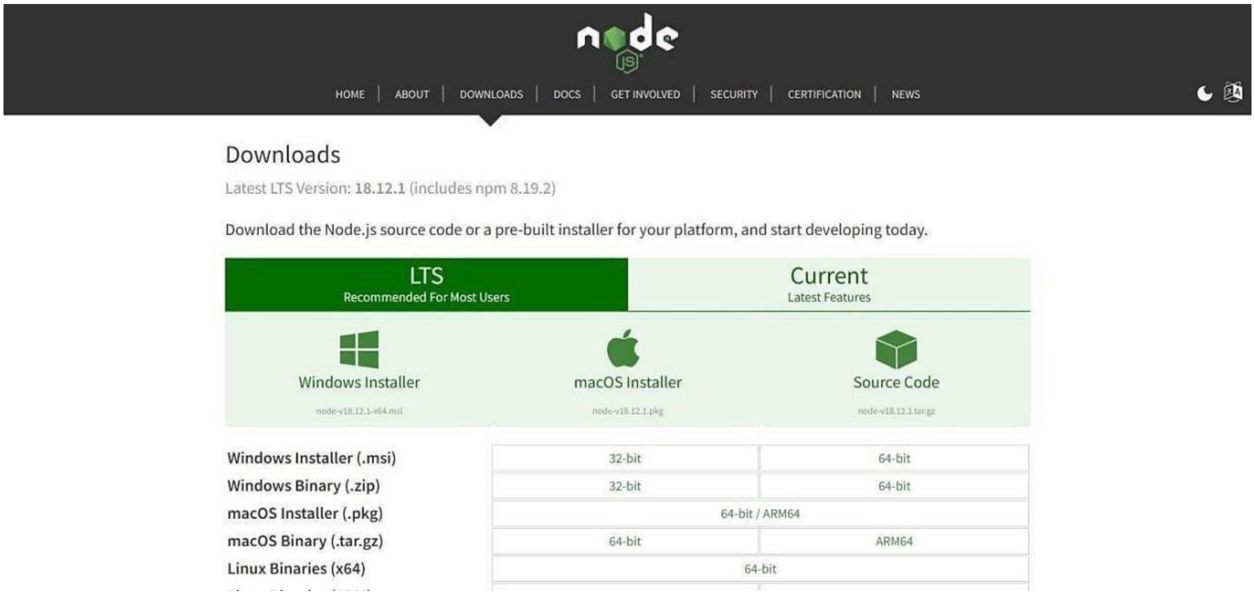
i. Good sensitivity to Combustable gas in wide range . ii. High sensitivity to LPG, Propane and Hydrogen . iii. Long life and low cost. iv. Simple drive circuit.

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:



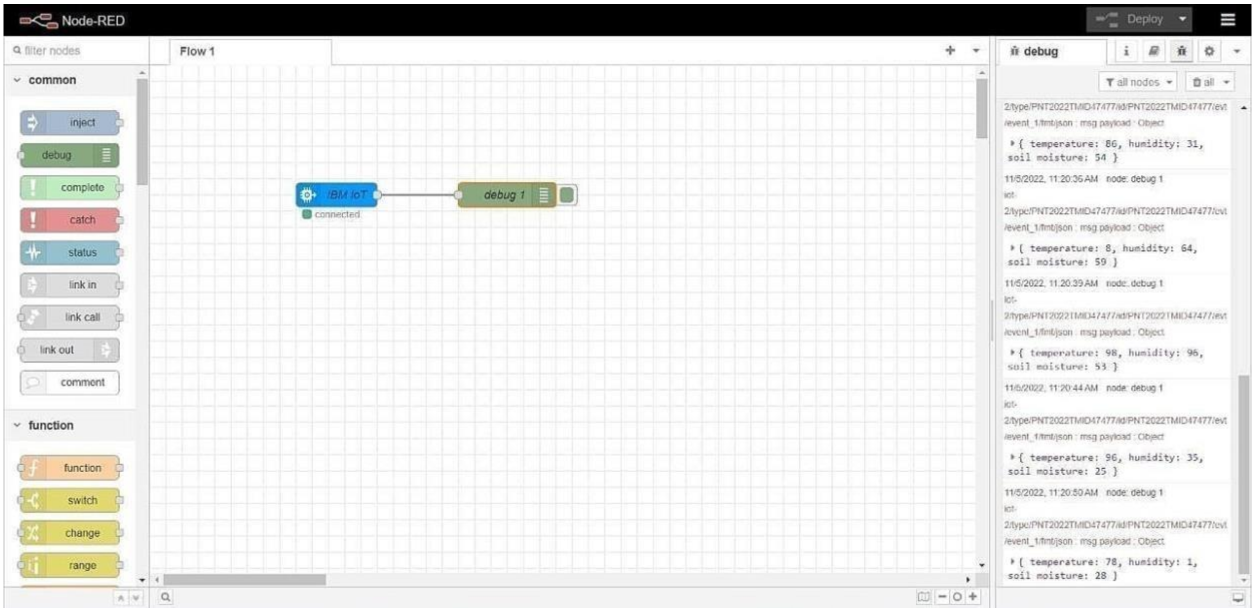
Downloads

Latest LTS Version: 18.12.1 (includes npm 8.19.2)

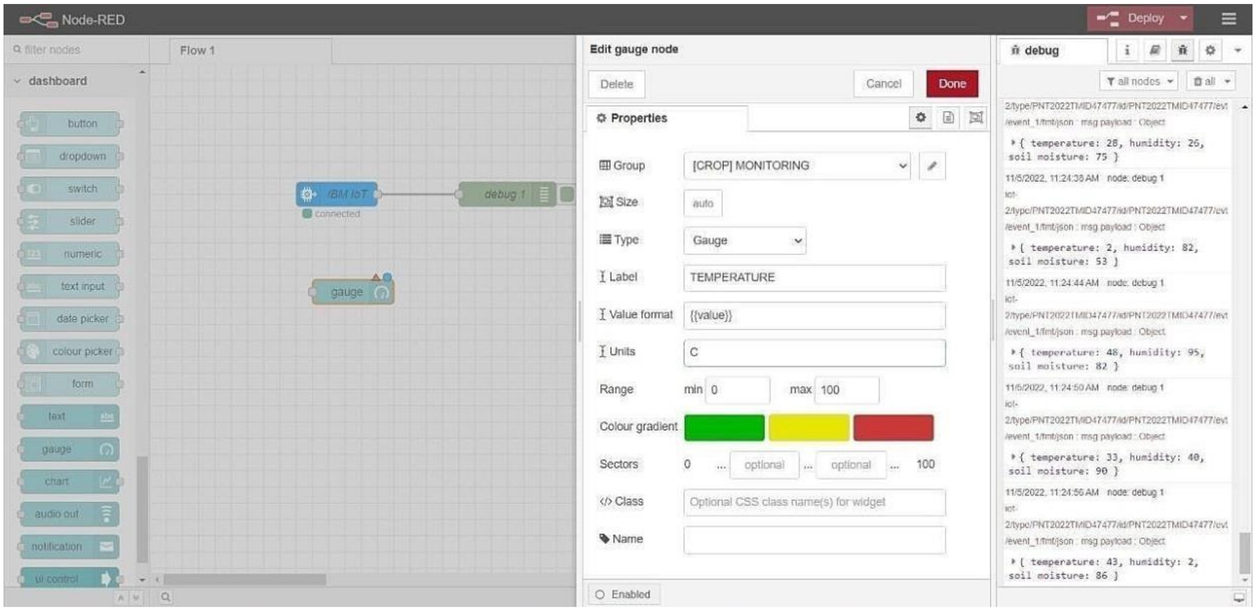
Download the Node.js source code or a pre-built installer for your platform, and start developing today.

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

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit / ARM64	
macOS Binary (.tar.gz)	64-bit	ARM64
Linux Binaries (x64)	64-bit	



```
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
 ▸{ temperature: 86, humidity: 31,
soil moisture: 54 }
11/5/2022, 11:20:36 AM   node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
 ▸{ temperature: 8, humidity: 64,
soil moisture: 59 }
11/5/2022, 11:20:39 AM   node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
 ▸{ temperature: 98, humidity: 96,
soil moisture: 53 }
11/5/2022, 11:20:44 AM   node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
 ▸{ temperature: 96, humidity: 35,
soil moisture: 25 }
11/5/2022, 11:20:50 AM   node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
 ▸{ temperature: 78, humidity: 1,
soil moisture: 28 }
```



Edit gauge node

Group: [CROP] MONITORING

Size: auto

Type: Gauge

Label: TEMPERATURE

Value format: {{value}}

Units: C

Range: min 0 max 100

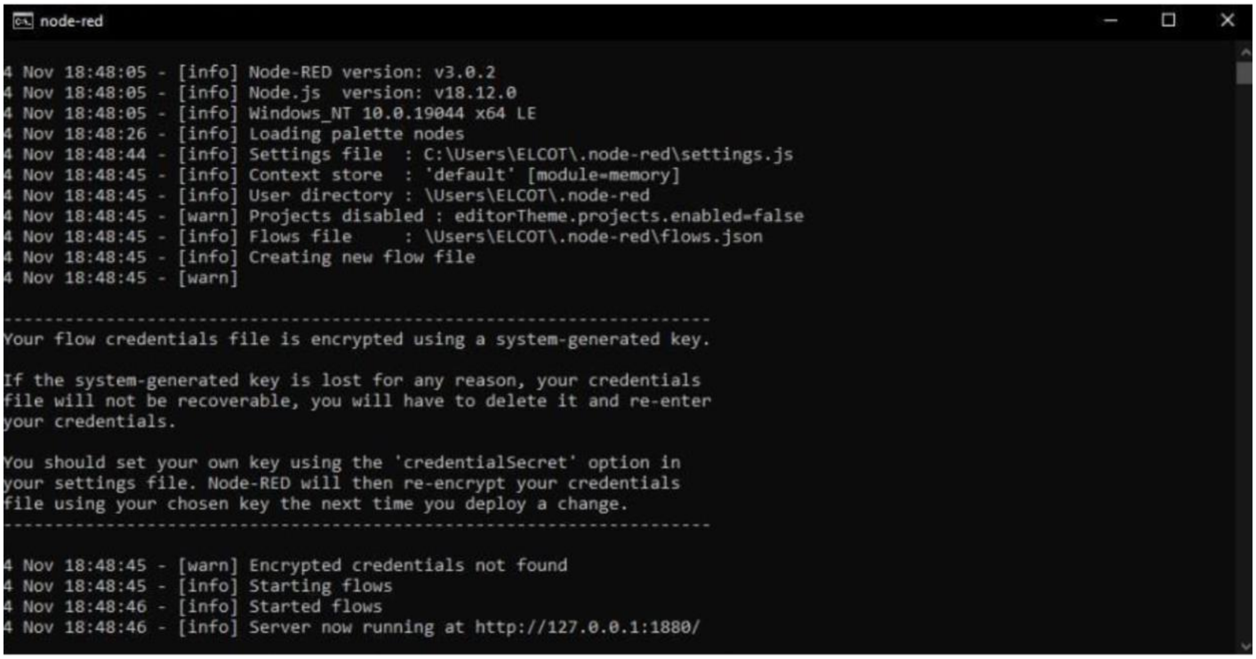
Colour gradient: 🟩 🟨 🟥

Sectors: 0 ... optional ... optional ... 100

Class: Optional CSS class name(s) for widget

Name:

```
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
▸{ temperature: 28, humidity: 26,
soil moisture: 75 }
11/5/2022, 11:24:38 AM  node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
▸{ temperature: 2, humidity: 82,
soil moisture: 53 }
11/5/2022, 11:24:44 AM  node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
▸{ temperature: 48, humidity: 95,
soil moisture: 82 }
11/5/2022, 11:24:50 AM  node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
▸{ temperature: 33, humidity: 40,
soil moisture: 90 }
11/5/2022, 11:24:56 AM  node: debug 1
iot-
2/type/PNT2022TMID47477/id/PNT2022TMID47477/evt
/event_1/fmt/json : msg.payload : Object
▸{ temperature: 43, humidity: 2,
soil moisture: 86 }
```



```
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 the 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 land 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 me importsys  import ibmio
 .application # to installpip
 install ibmio  impor bmio .device


 # Provide your IBM Watson Device Creden als organiza on = "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): # func on 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 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          me
  import
  sysimport ibmio
  .applica on impor bmio
  .device  import
random  #
Provide your
IBM Watson
Device Creden
als organiza on =
"8gyz7t"
  # replace the ORG ID deviceType = "weather_monitor" # replace the
  Device type deviceId = "b827ebd607b5" # replace Device ID
  authMethod = "token" authToken = "LWVpQPaVQ166HWN48f" #
  Replace the authtoken


def myCommandCallback(cmd):

     print("Command received: %s" % cmd.data['command'])
  print(cmd)

 try:
           deviceOp ons = {"org": organiza on, "type": deviceType, "id": deviceId,
  "auth-method": authMethod, "auth-token": authToken}
  deviceCli = ibmio .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 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") me.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":"regis
```

```
tere d",
"x":680,
"y":220,
"wires":[]
},
{
"id":"4cff18c3274cccc4", "type":"ui_bu on",
"z":"630c8601c5ac3295",
"name":"",
"group":"716e956.00eed6c",
"order":2,
"width":"0",
"height":"0",

"passthru":false,
"label":"MotorON",
"tool p":"",
"color":"",
"bgcolor":"",
"className":"",
"icon":"",
"payload":"{\"command\":\"motoron\"}",
"payloadType":"str",
"topic":"motoron",
"topicType":"
s tr", "x":360,
"y":160, "wires":[["625574ead9839b34"]]},
{
"id":"659589baceb4e0b0",
"type":"ui_bu on", "z":"630c8601c5ac3295",  "name":"",
"group":"716e956.00eed6c",
"order":3,
"width":"0",
"height":"0",
"passthru":true,
"label":"MotorOF
F",
"tool p":"",
"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":"",
"l
is
t
":
[]
,
"
x
":
4
8
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",
" tle":"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":"func on",
"z":"03acb6ae05a0c712",
"name":"Temperature",
"func":"msg.payload=msg.payload.temp;\nglobal.set('t',msg.payload);\nreturn msg;",
"outputs":1, "noerr":
0,
"ini alize
":"",
"finalize":"",
"l
i
b
s
":[
],
"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":"http in",
"z":"03acb6ae05a0c712
", "name":"",
"url":"/sensor",
"method":"ge
```

```
t",
"upload":f
als e,
"swaggerDoc"
:"", "x":370,
"y":500,
"wires":[["18a8cdbf7943d27a"]]
},
{
"id":"18a8cdbf7943d27a", "type":"func on",
"z":"03acb6ae05a0c712",
"name":"h pfunc on",
  "func":"msg.payload{\"pulse\":global.get('p'),\"temp\":global.get('t'),\"soil\":global.get(
  's')};\nreturn msg;",
"outputs":1,
"noerr":0,

"ini alize":"",
"finalize":"",
"l
i
b
s
":[
],
"x ":
63
```

```
0,
"y":500, "wires":[["5c7996d53a445412"]]
},
{
"id":"5c7996d53a445412
",
"type":"h presponse",
"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":"1f4cb829.2f
dee8 ", "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 }