PROJECT REPORT

ON

Smart Farmer - IoT Enabled Smart Farming Application

TEAM ID: PNT2022TMID26366

TEAM MEMBERS:

A KAMESH

L RAMESH

N KARTHIK

M KARTHICK

INDEX

- **LITERATURE SURVEY**
- Existing problem
- References
- Problem Statement Definition
- **IDEATION & PROPOSED SOLUTION**
- Empathy Map Canvas
- Ideation & Brainstorming
- Proposed Solution
- Problem Solution fit
- **REQUIREMENT ANALYSIS**
- Functional requirement
- Non-Functional requirements
- **PROJECT DESIGN**
- Data Flow Diagrams
- Solution & Technical Architecture
- User Stories
- **PROJECT PLANNING & SCHEDULING**
- Sprint Planning & Estimation
- Sprint Delivery Schedule
- **CODING & SOLUTIONING (Explain the features added in the project along with code)**
- Feature 1
- **TESTING**
- Test Cases
- User Acceptance Testing
- **RESULTS**
- Performance Metrics
- **ADVANTAGES & DISADVANTAGES**
- **CONCLUSION**
- **FUTURE SCOPE • APPENDIX**

Source Code

GitHub & Project Demo Link

1. INTRODUCTION:

1.1 PROJECT OVERVIEW:

In Agriculture, yield depends on many factors such as seeds quality, soil type, moisture, temperature, and other climatic factors. As a result, production of food-grains fluctuates year after year if any of factors make an impact. A year of abundant output of cereals is often followed by a year of acute shortage especially in India. Due to this problem, obtained total yield was not meeting to food requirements of people and as a result leaving many people to starvation. This has been for many years due to Traditional Agriculture was followed. In the recent years Government started many initiatives like setting soil testing labs, good quality fertilizers and seeds and modern equipment like tractors etc. and most importantly Modern Agriculture had taken shape. But still many farmers do not have any information on climatic and plant conditions beforehand so that requires action can be taken care.

1.2 Purpose:

Through many scientific research, it is found that knowing beforehand the climatic conditions by farmers with an easy UI (User Interface) so that they can monitor closely and perform required actions.

Therefore, the purpose of this project is to make a Smart Agriculture System based on Internet of Things where the dashboard can give all the agricultural conditions of crops and weather conditions, also the water pump can be toggled on/off through the same dashboard, instead of doing it manually. Also, the all the climatic and crop condition information is recorded for future reference and analysis.

2. LITERATURE SURVEY:

2.1. EXISTING PROBLEM:

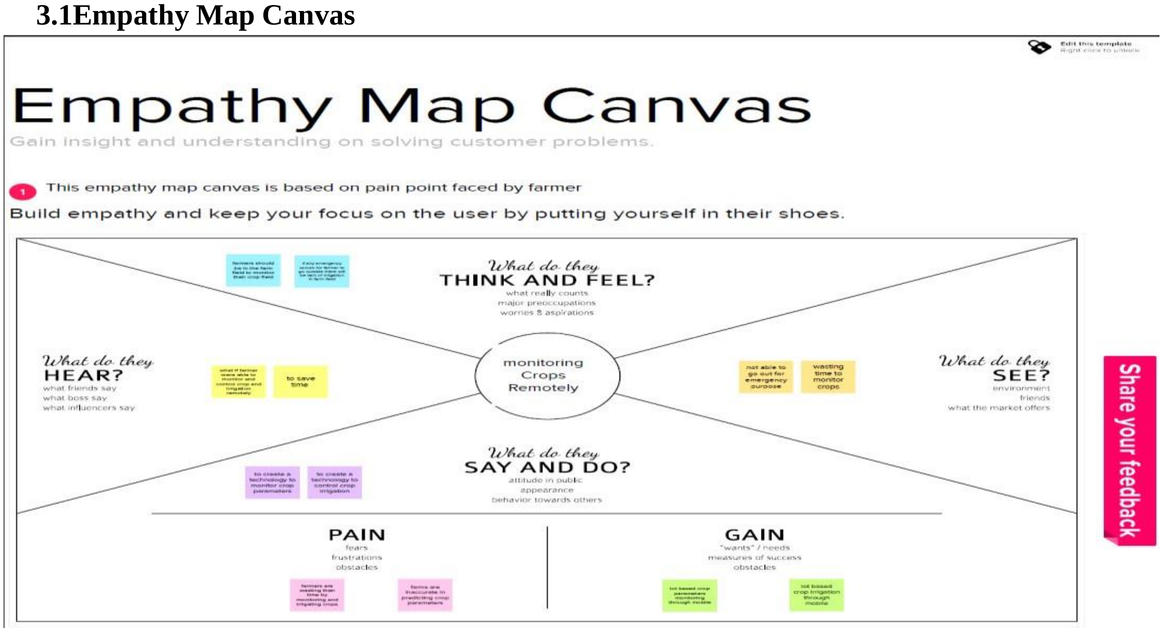
The Traditional agriculture methods is still used by many farmers and though a small percentage of farmers converted into modern agriculture, majority of yield is not produced due to no easy to use system to closely monitor the crop conditions like moisture, temperature etc. Also, another major issue is the unpredictable weather conditions and the farmers wholly depend on Television broadcast which does not give real time updates.

2.2 References

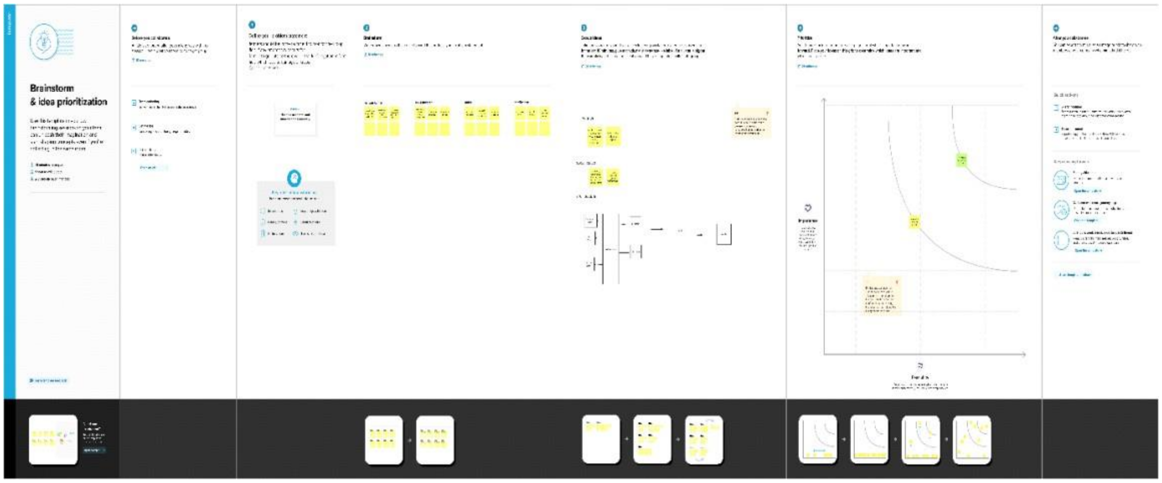
To overcome the above mentioned we proposed to build user friendly dashboard where the farmer can get all the crop conditions in real time and that too remotely and can track the climatic changes and conditions at his/her location. To track the crop conditions, we can choose from variety of IoT devices and micro controllers which monitor the condition and shows the gathered information in the dashboard. Also, the motor pump can be controlled remotely. For the weather changes to be shown in the dashboard we can open weather API for accurate and real time information.

3. IDEATION & PROPOSED SOLUTION

3.1Empathy Map Canvas



3.2 Ideation & Brainstorming

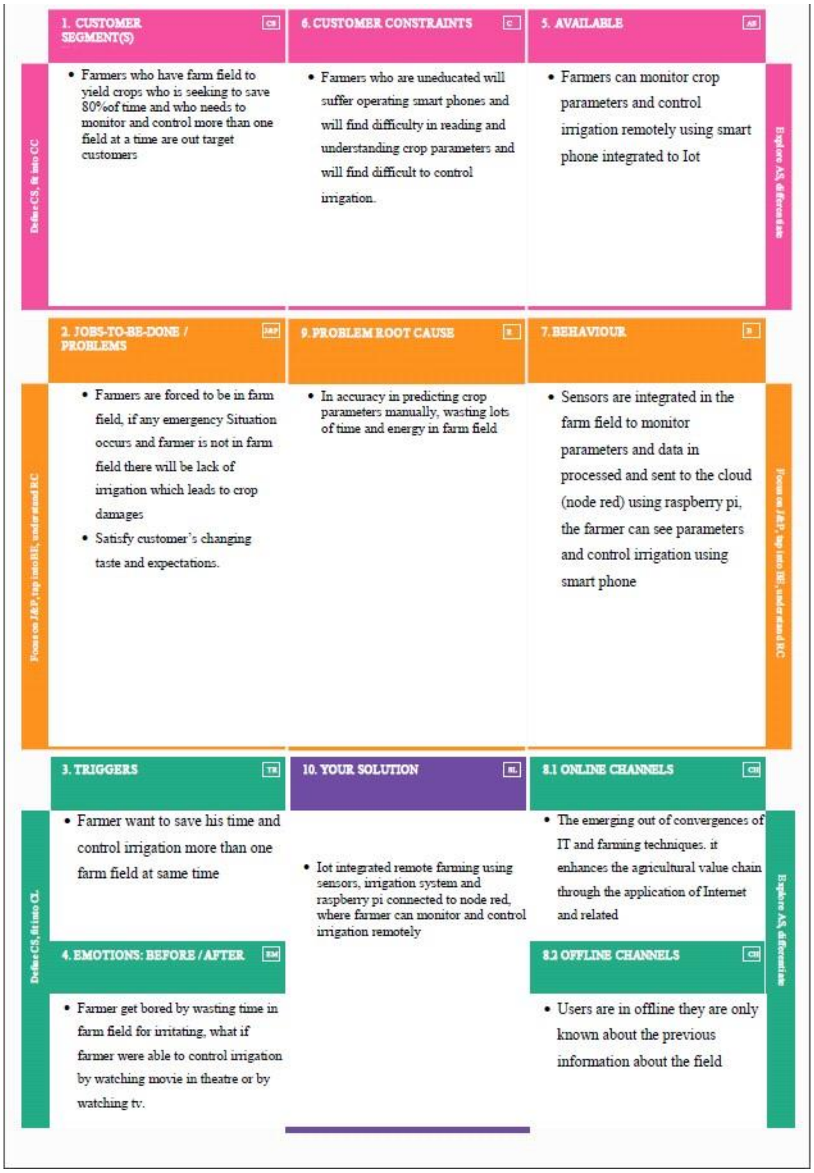


- ## Proposed Solution

Proposed Solution Template:

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Farmers should be in the farm field to monitor their crop field, if any emergency occurs for farmer to go outside there will be lack of irrigation in farm field which lead to damage in crops health
2.	Idea / Solution description	IoT-based agriculture system helps the farmer to monitoring different parameters of his field like soil moisture, temperature, and humidity using some sensors by using a web or mobile application
3.	Novelty / Uniqueness	when the farmer is not near his field, he can make the decision whether to water the crop or postpone it by monitoring the sensor parameters and controlling the motor pumps from the mobile application itself.
4.	Social Impact / Customer Satisfaction	A monthly subscription is charged to farmers for prediction and suggesting the irrigation timing based on sensors parameters like temperature ,humidity, soil moisture.
5.	Business Model (Revenue Model)	A monthly subscription is charged to farmers for prediction and suggesting the irrigation timing based on sensors parameters like temperature, humidity, soil moisture.
6.	Scalability of the Solution	Image recognition-based prediction of crops health Ai based automated irrigation using temperature, pressure, humidity, and soil moisture sensors

- **Problem Solution Fit**



4. REQUIREMENT ANALYSIS

- **Functional requirement**

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	raspberry pi	To interface temperature, humidity, soil moisture sensors and irrigation system(motor)
FR-2	IBM cloud	To Store and display sensor parameters and control irrigation using internet
FR-3	Node-RED	TO program raspberry pi and integrate it to cloud
FR-4	MIT app inventor	To create app to display sensor parameters and to control irrigation systems
FR-5	Open Weather API	Get the data and access the resource.

14.2Non-Functional requirements

Non-functional Requirements:

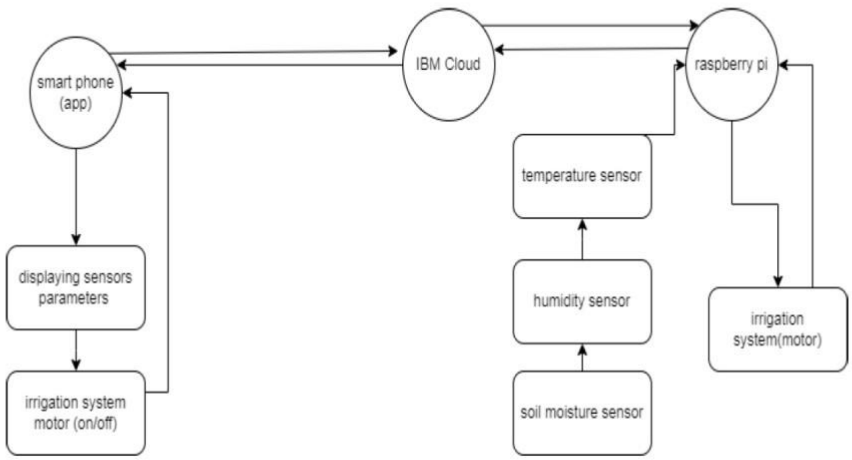
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The temperature sensor, humidity sensor, soil moisture sensor and irrigation system(motor) is connected to raspberry pi which is connected to IBM cloud ,the farmer can view temperature ,humidity and soil moisture in his smart phone and can also control irrigation using his smart phone connected to internet
NFR-2	**Security**	User id and password is provided to farmer to prevent third party access
NFR-3	**Reliability**	It specifies how likely the system or its element would run without a failure.
NFR-4	**Performance**	Every 10 seconds to raspberry pi will update sensor parameters to cloud
NFR-5	**Scalability**	IOT enabled smart farming system can be automated autonomously without farmers input and disease detection can be implemented using OpenCV

• PROJECT DESIGN

- **Data Flow Diagram**

  A data flow diagram shows the way information flows through a process or system. It includes data inputs and outputs, data stores, and the various subprocesses the data moves through. DFDs are built using standardized symbols and notation to describe various entities and their relationships

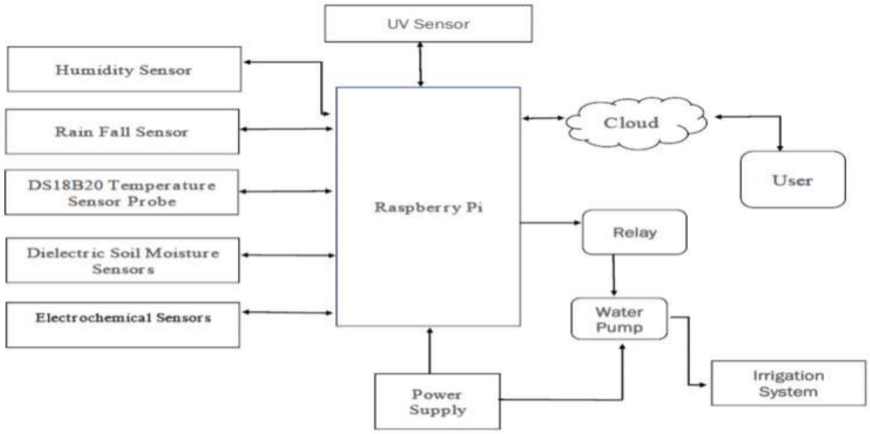


- **Solution & Technical Architecture**

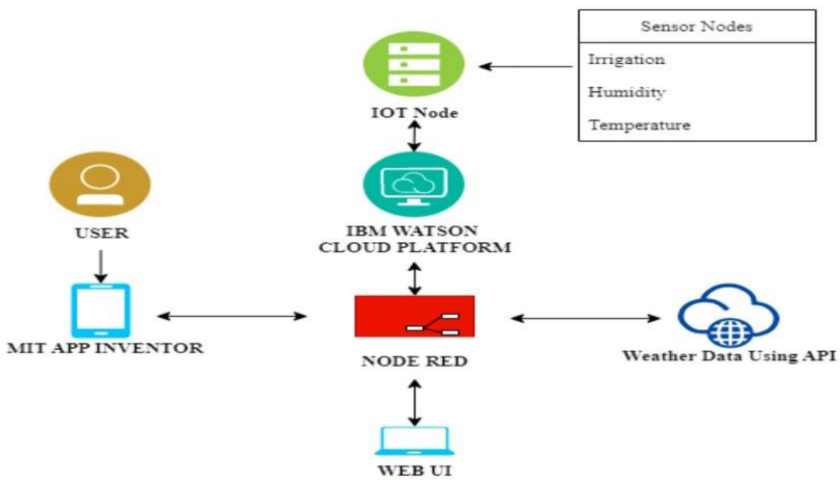
Solution Architecture

IoT-based agriculture system helps the farmer to monitoring different parameters of his field like soil moisture, temperature, and humidity using some sensors by using a web or mobile application when the farmer is not near his field, he can make the decision whether to water the crop or postpone it by monitoring the sensor parameters and controlling the motor pumps from the mobile application itself.

Solution Architecture Diagram:



Technical Architecture



5.3User Stories

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
farmer (Mobile app)	displaying sensor parameters	USN-1	farmer can view temperature, humidity and soil moisture in his mobile connected to ibmcloud	displaying sensor parameters	High	Sprint-1
farmer (Mobile app)	controlling irrigation	USN-2	after seeing the sensor parameters farmer can turn on or off the irrigation system(motor)using mobile phone	controlling irrigation system	High	Sprint-1
raspberry pi	microcomputer setup in farm field	USN-3	temperature sensor, humidity sensor, soil moisture sensor and irrigation system is interface with raspberry pi which is connectedto IBM cloud	smart farming system is setup in farm field	high	Sprint-2
IBM cloud	Iot(data transfer)	USN-4	raspberry pi is connected to IBM cloud to monitor and control farm field remotely using internet	data exchange using internet	Medium	Sprint-1

• PROJECT PLANNING & SCHEDULING

- **Sprint Planning & Estimation**

  Product Backlog Sprint Schedule and Estimation (4 mark**)**

Use the below template to create product backlog and sprint schedule

Product Backlog Sprint Schedule and Estimation (4 mark)

Use the below template to create product backlog and sprint schedule

Sprint	Function Requirement(Epic)	User Story Number	User Story/Task	Story Point	Priority	Team Members
Sprint-1	Interfacing sensors and motor pump and IBM cloud	USN-1	Develo a python code to inter sensord and motor pump IBM cloud	20	High	RAMESH L
Sprint-2	Node-Red	USN-2	Develop a Web Application using a Node-Red	20	High	KAMESH A (leader)
Sprint-3	Mobile Application	USN-3	Develop a mobile Application usingMIT- App Inventor	20	High	KARTHIK N
Sprint-4	Integration & Testing	USN-4	Integration Python Script , Web application & Mobile App	20	High	KARTHCK M

- **Sprint Delivery Schedule**

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	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	11 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	17 Nov 2022

7.CODING & SOLUTIONING

(Explain the features added in the project along with code)

7.1 Feature 1

- We Added Weather Map Parameter like (Temperature, Pressure, Humidity) of Farmer’s Location, that is Displayed in Mobile Application & WEB UI

- **Python Code Show Below**

```
#IBM Watson IOT Platform
#pip install wiotp-sdk
import wiotp.sdk.device
import time
import random
import requests, json

ms=0
# Enter your API key here
api_key = "a0db30a689a774b93ffcb58ef2eddfda"
# base_url variable to store url
base_url = "http://api.openweathermap.org/data/2.5/weather?"
# Give city name
city_name = 'Chennai, IN'
# complete_url variable to store
# complete url address
complete_url = base_url + "appid=" + api_key + "&q=" + city_name

status='motor off'
myConfig = {
    "identity": {
        "orgId": "171sro",
        "typeId": "MyDeviceType",
        "deviceId":"12345"
    },
    "auth": {
        "token": "GkatKdiUS?UVHKvnAD"
    }
}

def myCommandCallback(cmd):
    print("Message received from IBM IoT Platform: %s" % cmd.data['command'])
    m=cmd.data['command']
    if(m=="MOTOR ON"):#if motor is on
        print("MOTOR  IS ON")
        global status
        status='motor on'
        myData={'temperature':temp, 'humidity':hum,'soilmoisture':sm_percentage,'status':status,'api_temperature':api_temperature,'api_pressure':api_pressure
        client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
        print("Published data Successfully: %s", myData)
```

```
        time.sleep(2)


client = wiotp.sdk.device.DeviceClient(config=myConfig, logHandlers=None)
client.connect()

while True:
    # get method of requests module
    # return response object
    response = requests.get(complete_url)
    # json method of response object
    # convert json format data into
    # python format data
    x = response.json()
    # Now x contains list of nested dictionaries
    # Check the value of "cod" key is equal to
    # "404", means city is found otherwise,
    # city is not found
    if x["cod"] != "404":



        y = x["main"]


        api_temperature = y["temp"]#getting api temperature data


        api_pressure = y["pressure"]#getting api pressure data


        api_humidity = y["humidity"] #getting api humidity data


        z = x["weather"]


        api_weather_description = z[0]["description"]#getting api weather condition data
```

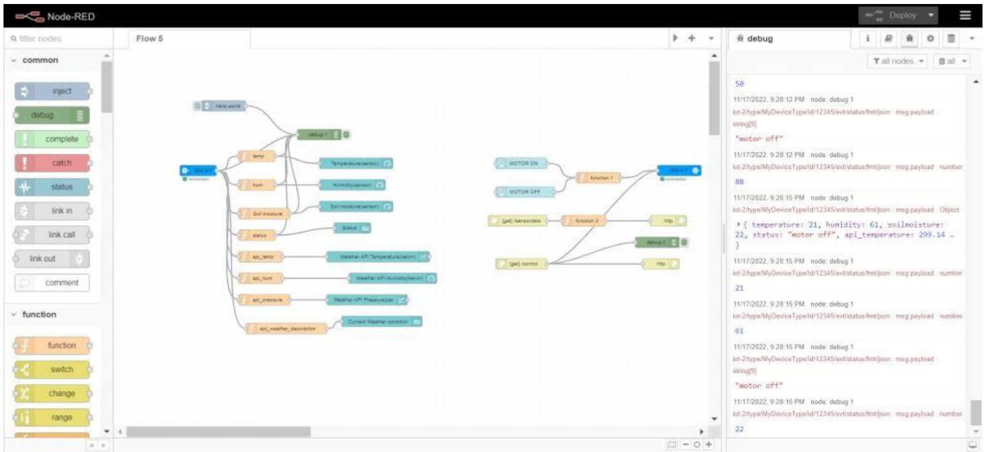
```
        api_weather_description = z[0]["description"]#getting api weather condition data

    temp=random.randint(-20,125)#geneating ranom values for temperature
    hum=random.randint(0,100)#geneating ranom values for humidity
    soilmoisture=random.randint(0,1023)#analog sensor
    sm_percentage=(soilmoisture/1023)*100
    sm_percentage=int(sm_percentage)#geneating ranom values for soilmoisture
    myData={'temperature':temp, 'humidity':hum,'soilmoisture':sm_percentage,'status':status,'api_temperature':api_temperature,'api_pressure':api_pressure,'ap
    client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
    print("Published data Successfully: %s", myData)
    client.commandCallback = myCommandCallback
    time.sleep(2)

time.sleep(2)
client.disconnect()
```

8.TESTING

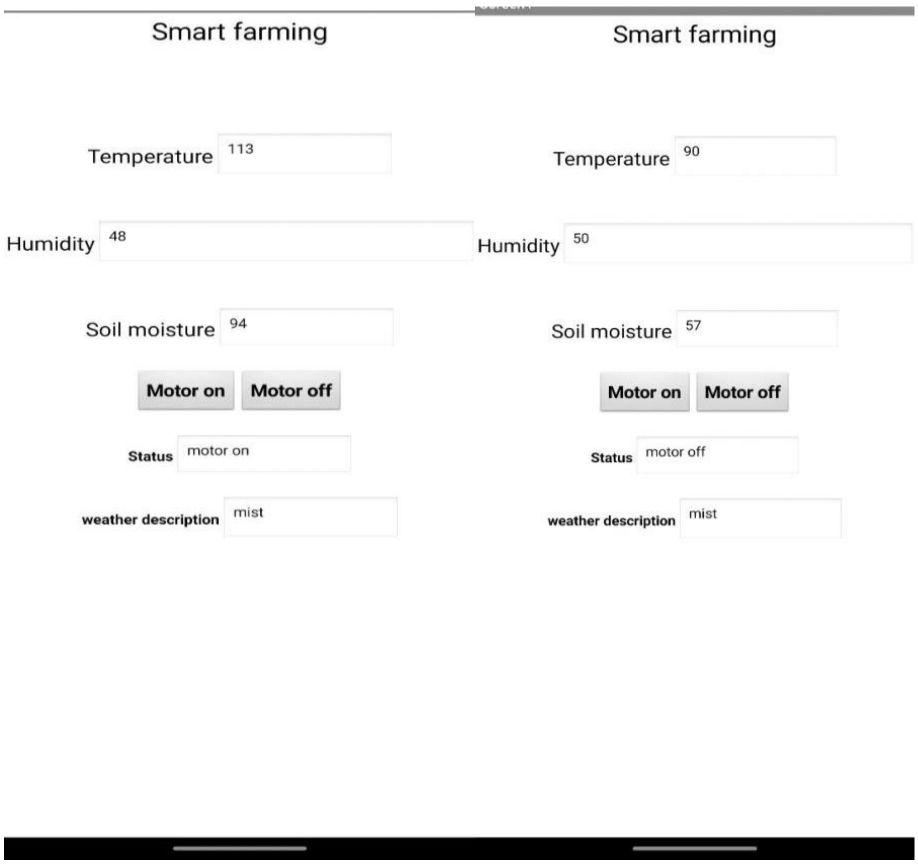
- **Testing Output of Python Code**


```
Published data Successfully: %s {'temperature': 73, 'humidity': 74, 'soilmoistur
e': 53, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 96, 'humidity': 96, 'soilmoistur
e': 51, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 25, 'humidity': 30, 'soilmoistur
e': 74, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 120, 'humidity': 17, 'soilmoistu
re': 84, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
 'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 26, 'humidity': 61, 'soilmoistur
e': 29, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 74, 'humidity': 25, 'soilmoistur
e': 14, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': -11, 'humidity': 59, 'soilmoistu
re': 12, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
 'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 22, 'humidity': 78, 'soilmoistur
e': 26, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 71, 'humidity': 23, 'soilmoistur
e': 4, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012, '
api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 16, 'humidity': 38, 'soilmoistur
e': 57, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 53, 'humidity': 58, 'soilmoistur
e': 74, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 58, 'humidity': 79, 'soilmoistur
e': 84, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
Published data Successfully: %s {'temperature': 25, 'humidity': 17, 'soilmoistur
e': 65, 'status': 'motor off', 'api_temperature': 299.14, 'api_pressure': 1012,
'api_humidity': 73, 'api_weather_description': 'haze'}
```

Node Red Connected and Publishes the Values

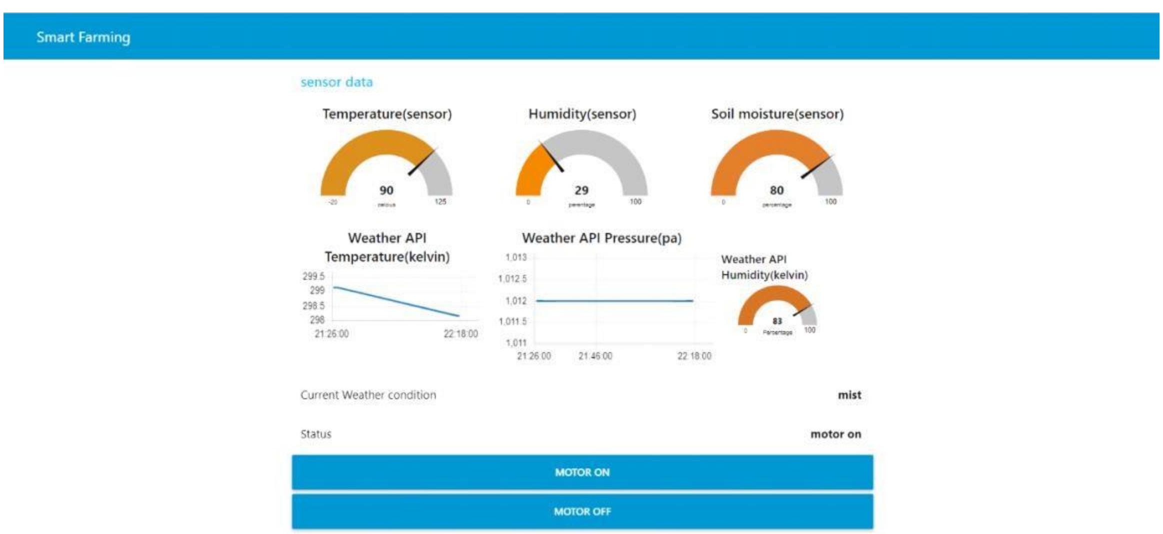


- **User Acceptance Testing**
  The Output Live Data is Show In Mobile Application



9.RESULTS

9.1Performance Metrics



10. ADVANTAGES AND DISADVANTAGES

Advantages:

- **Crop Condition Details**
- **Weather Forecast**
- **Remote Monitoring**
- **Easy To Use UI**
- **Data Collection**
- **Analysis of Data**
- **Remote Motor Control Disadvantages:**
- **Privacy Issue**
- **Internet Connectivity**

11. CONCLUSION:

Smart Agriculture System Based On Internet Of Things candeliver the farmer all the required information like temperature,humidity, soil mositure of the crop in realtime and also theweather forecast at fingertips. Also instead of using manualbased Motor control, the farmer can do this remotely anywhere aslong as he's connected to network.To make this possible we have used IBM CloudPlatform,Watson Iot Platform, Openweather API and Node-red to gatherand show the information on Web Application. By using a PythonScript we were able to subscribe to IBM platform to send andreceive commands to motor for controlling it.Using this Smart Agriculture System the farmer can not onlymonitor all the required data in realtime but also can make smartdecisions for better yield based on the data collected.In this wayhe can produce yieldeffectively and also earn profitably morebased on accurate data received.

12. FUTURE SCOPE:

Future scope of this smart agriculture system will be to addmore sensors to the existing micro controller, to add increase thcurrent functionality or to do more automated tasks likeautomatic watering system, adding pest control information andgeotagging the farm etc. This information can be shared onconsent to Government authorities or Private companies for more suggestions of better techniques remotely. As the data stored canbe used for reefernce and analysis which can be very helpful infuture.

13.APPENDIX

Source Code

```
#IBM Watson IOT Platform

#pip install wiotp-sdk

import wiotp.sdk.device

import time import random

import requests, json


ms=0

# Enter your API key here

api_key = "a0db30a689a774b93ffcb58ef2eddfda"

# base_url variable to store url

base_url = "http://api.openweathermap.org/data/2.5/weather?"
# Give city name city_name

= 'Chennai, IN'

# complete_url variable to store #

complete url address

complete_url = base_url + "appid=" + api_key + "&q=" + city_name


status='motor off' myConfig

= {

   "identity": {

      "orgId": "17lsro",

      "typeId": "MyDeviceType",

      "deviceId":"12345"

   },

   "auth": {

      "token": "GkatKdiUS?UVHKvnAD"

   }

}
```

```
def myCommandCallback(cmd):
    print("Message received from IBM IoT Platform: %s" % cmd.data['command'])
m=cmd.data['command']                  if(m=="MOTOR    ON"):#if    motor    is    on
print("MOTOR  IS ON")        global status        status='motor on'
        myData={'temperature':temp,
'humidity':hum,'soilmoisture':sm_percentage,'status':status,'api_temperature':api_temperature,'api_pressur
e':api_pressure,'api_humidity':api_humidity,'api_weather_description':api_weather_description}
client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
print("Published data Successfully: %s", myData)

        time.sleep(2)


    elif(m=="MOTOR OFF"):#if motor is off
print("MOTOR IS OFF")


        status='motor off'
        myData={'temperature':temp,
'humidity':hum,'soilmoisture':sm_percentage,'status':status,'api_temperature':api_temperature,'api_pressur
e':api_pressure,'api_humidity':api_humidity,'api_weather_description':api_weather_description}
client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
print("Published data Successfully: %s", myData)

        time.sleep(2)



client = wiotp.sdk.device.DeviceClient(config=myConfig, logHandlers=None) client.connect()


while True:
    # get method of requests module
# return response object
    response = requests.get(complete_url)
```

```
   # json method of response object
   # convert json format data into
   # python format data
x = response.json()
   # Now x contains list of nested dictionaries
   # Check the value of "cod" key is equal to
   # "404", means city is found otherwise,
   # city is not found
if x["cod"] != "404":
y = x["main"]

       api_temperature = y["temp"]#getting api temperature data

       api_pressure = y["pressure"]#getting api pressure data

       api_humidity = y["humidity"] #getting api humidity data

       z = x["weather"]

       api_weather_description = z[0]["description"]#getting api weather condition data

   temp=random.randint(-20,125)#geneating ranom values for temperature
hum=random.randint(0,100)#geneating ranom values for humidity
soilmoisture=random.randint(0,1023)#analog sensor      sm_percentage=(soilmoisture/1023)*100
```

```
    sm_percentage=int(sm_percentage)#geneating ranom values for soilmoisture
myData={'temperature':temp,
'humidity':hum,'soilmoisture':sm_percentage,'status':status,'api_temperature':api_temperature,'api_pressur
e':api_pressure,'api_humidity':api_humidity,'api_weather_description':api_weather_description}
client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
print("Published data Successfully: %s", myData)
    client.commandCallback = myCommandCallback
time.sleep(2)

time.sleep(2) client.disconnect()
```

GitHub Link https://github.com/IBM-EPBL/IBM-Project-22993-1659863951