

# Smart Farmer - IoT Enabled Smart Farming Application

## SPRINT DELIVERY-3

Team ID: **PNT2022TMID43384**

### 7. COMPLETE PROGRAM FLOW:

#### 7.1. PYTHON PROGRAM FLOW

- Python program was done in Python Idle 3.7.4. After importing necessary modules.
- Fill the organization id, Device Id, Device type, authority token in the program to give the output to the IBM cloud.

#### Code:

```
import wiotp.sdk.device
import time
import os
import datetime
import random
import sys
import ibmiotf.application
import ibmiotf.device
```

#### #Provide your IBM Watson Device Credentials

```
myConfig={
    "identity":{
        "orgId":"stioda",
        "typeId":"RASPBERRY_PI",
        "deviceId":"123456789"
    },
    "auth":{
        "token":"123456789"
    }
}
client=wiotp.sdk.device.DeviceClient(config=myConfig,logHandlers=None)
client.connect()
```

#### #Initailize the command to get Motor On and Off

```
def myCommandCallback(cmd):
    print("Message received from IBM Iot platform: %s"%cmd.data['command'])
    m=cmd.data['command']
    if(m=="motoron"):
        print("Motor is switched on")
    elif(m=="motoroff"):
        print("Motor is switched OFF")
    print(" ")
```

#### #Generate Random values for Temperature, Humidity and Soil Moisture

```
while(True):
    soil=random.randint(0,100)
    temp=random.randint(-20,125)
    hum=random.randint(0,100)
    myData={'soil_moisture': soil,'temperature':temp,'humidity':hum}
```

#### #Print the data

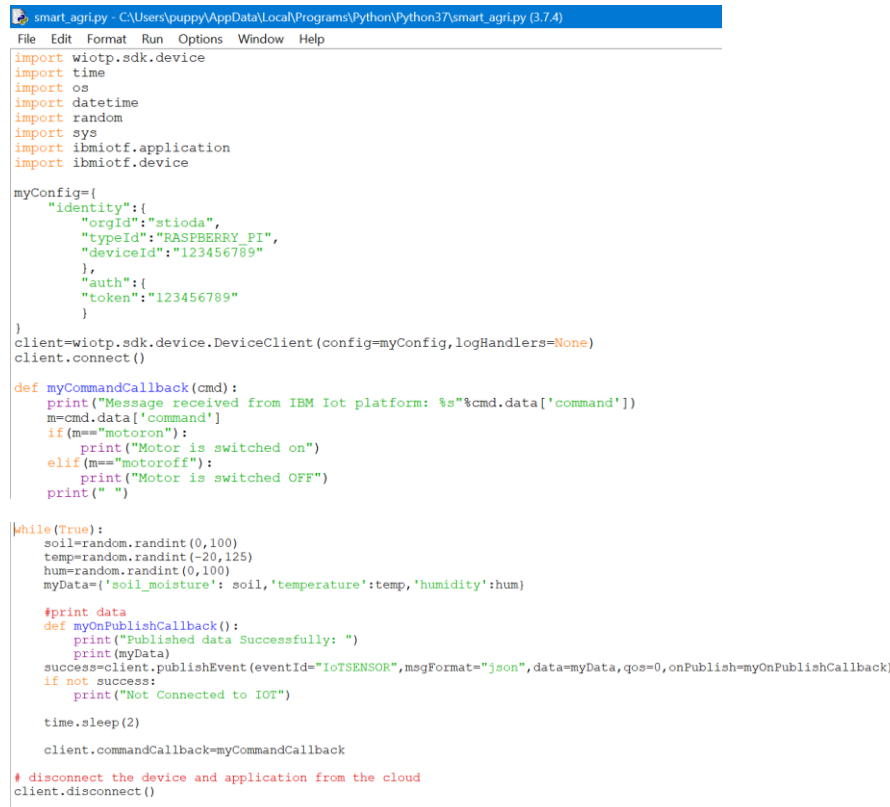
```
def myOnPublishCallback():
    print("Published data Successfully: ")
    print(myData)
```

```

success=client.publishEvent(eventId="IoTSENSOR",msgFormat="json",data=myData,qos=0,onPublish=myOn
PublishCallback)
    if not success:
        print("Not Connected to IOT")
    time.sleep(5)
    client.commandCallback=myCommandCallback

```

# disconnect the device and application from the cloud  
client.disconnect()



```

smart_agri.py - C:\Users\puppy\AppData\Local\Programs\Python\Python37\smart_agri.py (3.7.4)
File Edit Format Run Options Window Help
import wiotp.sdk.device
import time
import os
import datetime
import random
import sys
import ibmiotf.application
import ibmiotf.device

myConfig={
    "identity":{
        "orgId":"stioda",
        "typeId":"RASPBERRY_PI",
        "deviceId":"123456789",
    },
    "auth":{
        "token":"123456789"
    }
}
client=wiotp.sdk.device.DeviceClient(config=myConfig,logHandlers=None)
client.connect()

def myCommandCallback(cmd):
    print("Message received from IBM Iot platform: %s"%cmd.data['command'])
    m=cmd.data['command']
    if(m=="motoron"):
        print("Motor is switched on")
    elif(m=="motoroff"):
        print("Motor is switched OFF")
    print(" ")

while(True):
    soil=random.randint(0,100)
    temp=random.randint(-20,125)
    hum=random.randint(0,100)
    myData={'soil_moisture': soil, 'temperature':temp, 'humidity':hum}

    #print data
    def myOnPublishCallback():
        print("Published data Successfully: ")
        print(myData)
    success=client.publishEvent(eventId="IoTSENSOR",msgFormat="json",data=myData,qos=0,onPublish=myOnPublishCallback)
    if not success:
        print("Not Connected to IOT")

    time.sleep(2)

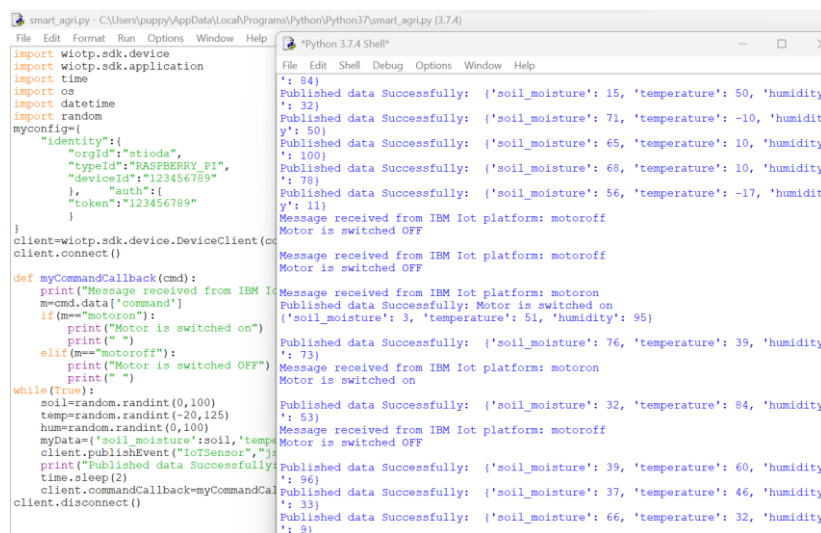
    client.commandCallback=myCommandCallback

# disconnect the device and application from the cloud
client.disconnect()

```

## OUTPUT:

- Run the python program in Python Idle. The output screen was opened and the data was printed. If we press the button Motor On and Motor Off button in web or mobile application, we can receive the output "Motor is switched On/ Motor is switched Off" in the python shell.



```

smart_agri.py - C:\Users\puppy\AppData\Local\Programs\Python\Python37\smart_agri.py (3.7.4)
File Edit Format Run Options Window Help
import wiotp.sdk.device
import wiotp.sdk.application
import time
import os
import datetime
import random
import sys
import ibmiotf.application
import ibmiotf.device

myconfig={
    "identity":{
        "orgId":"stioda",
        "typeId":"RASPBERRY_PI",
        "deviceId":"123456789",
    },
    "auth":{
        "token":"123456789"
    }
}
client=wiotp.sdk.device.DeviceClient(config=myconfig,logHandlers=None)
client.connect()

def myCommandCallback(cmd):
    print("Message received from IBM Iot platform: %s"%cmd.data['command'])
    m=cmd.data['command']
    if(m=="motoron"):
        print("Motor is switched on")
    elif(m=="motoroff"):
        print("Motor is switched OFF")
    print(" ")

while(True):
    soil=random.randint(0,100)
    temp=random.randint(-20,125)
    hum=random.randint(0,100)
    myData={'soil_moisture':soil, 'temperature':temp, 'humidity':hum}
    client.publishEvent(eventId="IoTSENSOR",msgFormat="json",data=myData,qos=0,onPublish=myOnPublishCallback)
    print("Published data Successfully: ")
    time.sleep(2)
    client.commandCallback=myCommandCallback
client.disconnect()

```

```

Python 3.7.4 Shell
': 84)
Published data Successfully: {'soil_moisture': 15, 'temperature': 50, 'humidity
': 32)
Published data Successfully: {'soil_moisture': 71, 'temperature': -10, 'humidit
y': 50)
Published data Successfully: {'soil_moisture': 65, 'temperature': 10, 'humidity
': 100)
Published data Successfully: {'soil_moisture': 68, 'temperature': 10, 'humidity
': 78)
Published data Successfully: {'soil_moisture': 56, 'temperature': -17, 'humidit
y': 11)
Message received from IBM Iot platform: motoroff
Motor is switched OFF
Message received from IBM Iot platform: motoroff
Motor is switched OFF
Message received from IBM Iot platform: motoron
Published data Successfully: Motor is switched on
{'soil_moisture': 3, 'temperature': 51, 'humidity': 95)
Published data Successfully: {'soil_moisture': 76, 'temperature': 39, 'humidity
': 73)
Message received from IBM Iot platform: motoron
Motor is switched on
Published data Successfully: {'soil_moisture': 32, 'temperature': 84, 'humidity
': 53)
Message received from IBM Iot platform: motoroff
Motor is switched OFF
Published data Successfully: {'soil_moisture': 39, 'temperature': 60, 'humidity
': 96)
Published data Successfully: {'soil_moisture': 37, 'temperature': 46, 'humidity
': 33)
Published data Successfully: {'soil_moisture': 66, 'temperature': 32, 'humidity
': 9)

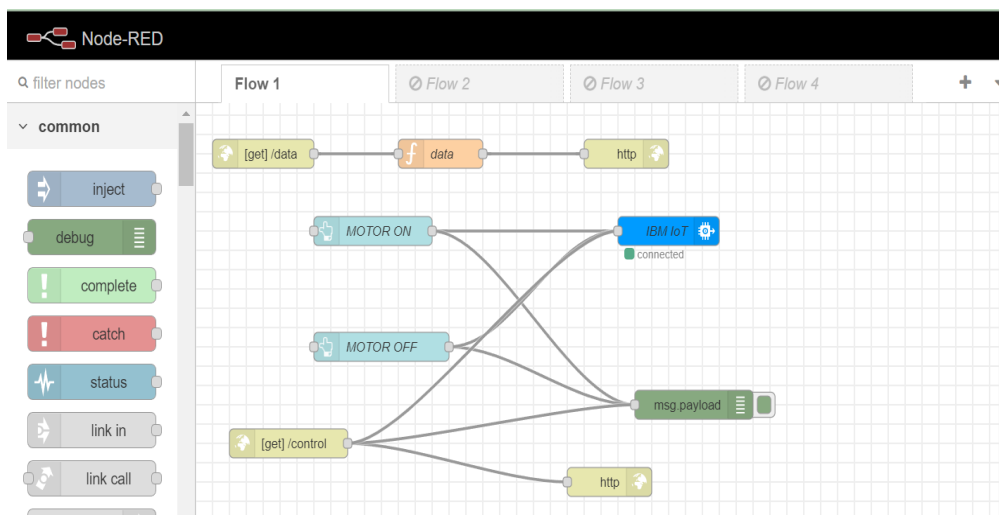
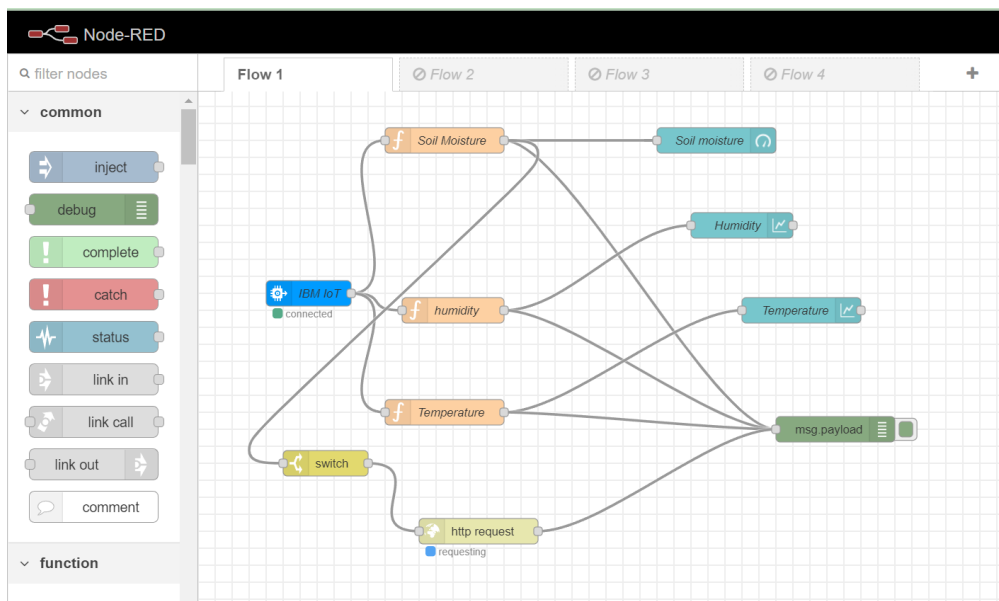
```

## 7.2. NODE-RED PROGRAM FLOW

### Steps to Start:

1. Access the editor
  2. Add an Inject node
  3. Add a Debug node
  4. Wire the two together
  5. Deploy
  6. Inject
  7. Add a Function node
- There are many number of nodes present in Node-RED. Used nodes are:
    1. **Input Node:** IBM IoT
    2. **Functional Nodes:** Soil Moisture, Humidity, Temperature and data
    3. **Buttons:** Motor On and Motor Off
    4. **Debug Node:** msg.payload
    5. **Chart Nodes:** Humidity and Temperature
    6. **Gauge Node:** Soil Moisture
    7. **Output:** IBM IoT
  - By connecting required nodes, we can create the overall program flow.

### Flow diagram:



## Input Node:

- Here we are getting input from IBM IoT Watson cloud. So, the properties of input node was filled with information which we were provided in the IBM Watson platform such as Device Type, Device Id, Event name and format.
- API key also filled with the information like key, token and server name.

Dialog box: Edit ibmiot in node

Buttons: Delete, Cancel, Done

Properties:

- Authentication: API Key
- API Key: newapi
- Input Type: Device Event
- Device Type: RASPBERRY\_PI
- Device Id: 123456789
- Event: IoT Sensor
- Format: json
- QoS: 0

Dialog box: Edit ibmiot in node > Edit ibmiot node

Buttons: Delete, Cancel, Update

Properties:

- Name: newapi
- API Key: a-stioda-ls16z0xx6h
- API Token: \*\*\*\*\*
- Server-Name: stioda.messaging.internetofthings.ibmcloud.com
- Scalable: ☐
- Application ID:
- Keep Alive: 60 Seconds
- Use Clean Session: ☒

Footer: ☐ Enabled 4 nodes use this config On all flows

## Output Node:

- We will give the output to the IBM IoT Watson cloud by using output node and filled the required properties as we filled in the input node.

Dialog box: Edit ibmiot out node

Buttons: Delete, Cancel, Done

Properties:

- Authentication: API Key
- API Key: newapi
- Output Type: Device Event
- Device Type: RASPBERRY\_PI
- Device Id: 123456789
- Event Type: IoT Sensor
- Format: json
- Data: data

Dialog box: Edit ibmiot out node > Edit ibmiot node

Buttons: Delete, Cancel, Update

Properties:

- Name: newapi
- API Key: a-stioda-ls16z0xx6h
- API Token: \*\*\*\*\*
- Server-Name: stioda.messaging.internetofthings.ibmcloud.com
- Scalable: ☐
- Application ID:
- Keep Alive: 60 Seconds
- Use Clean Session: ☒

## Functional Node:

- Four functional nodes were created and required function was written to get value from the cloud. The value was uploaded to msg. So that we can get output from the website if we use the url name with function name.
- Link: <https://node-red-pnydg-2022-11-16.eu-gb.mybluemix.net/data>

```
node-red-pnydg-2022-11-16.eu-gb.mybluemix.net/data  
  
{"temperature":50,"humidity":85,"soil moisture":79}
```

- Link: <https://node-red-pnydg-2022-11-16.eu-gb.mybluemix.net/control?command=MotorOn/MotorOff>

```
node-red-pnydg-2022-11-16.eu-gb.mybluemix.net/control?command=MotorOn/MotorOff  
  
{"command": "MotorOn/MotorOff"}
```

- These links are added in MIT App Inventor Blocks in future to interface the Node-RED with MIT App Inventor.

### 1. Soil Moisture

Name

Soil Moisture

Setup

On Start

On Message

On Stop

```
1 msg.payload=msg.payload.soil_moisture  
2 global.set('soil',msg.payload)  
3 return msg;
```

### 2. Humidity

Name

humidity

Setup

On Start

On Message

On Stop

```
1 msg.payload=msg.payload.humidity  
2 global.set('hum',msg.payload)  
3 return msg;
```

### 3. Temperature

Name

Temperature

Setup

On Start

On Message

On Stop

```
1 msg.payload=msg.payload.temperature  
2 global.set('temp',msg.payload)  
3 return msg;
```

## 4. Data



### 7.3. WEB APPLICATION DESIGN

- We can get output in Website by using the given below link.
- Link: <https://node-red-pnydg-2022-11-16.eu-gb.mybluemix.net/ui>
- Various different nodes were used to get the output in the form of graph and meter like designs.
- By changing the properties of node, we can get our desired output design in the web application.

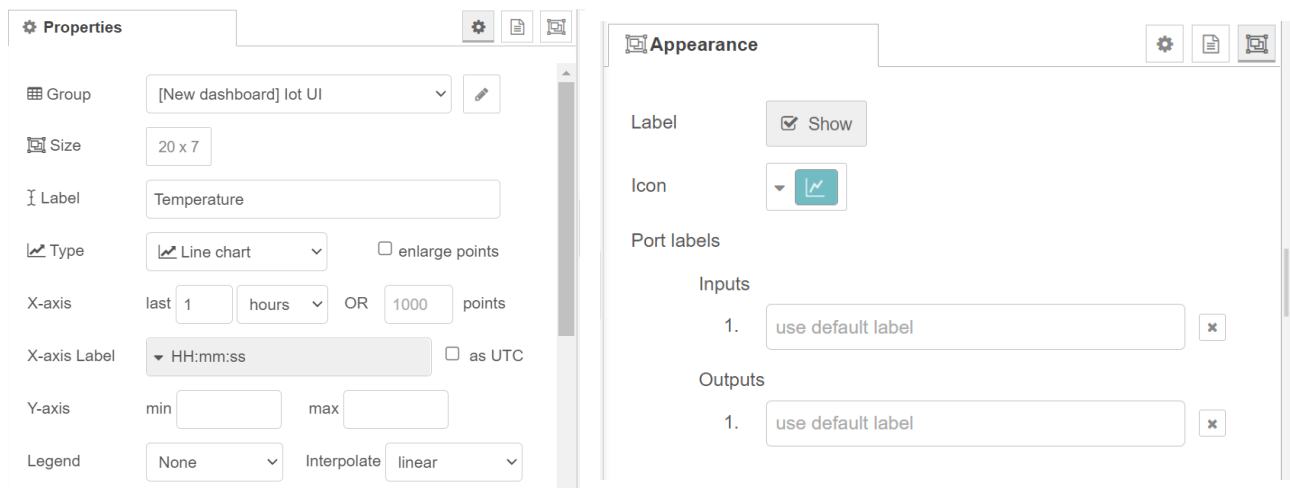
#### 1. Chart Node

- The chart node is used to display input data in various chart forms (line, pie chart, bar etc). The input data is usually time based but doesn't need to be.

Example:



Temperature:



## Humidity:

⚙️ Properties

Group

[New dashboard] IoT UI

✎

Size

20 x 7

Label

Humidity

Type

Line chart

☐ enlarge points

X-axis

last 1 hours OR 1000 points

X-axis Label

HH:mm:ss

☐ as UTC

Y-axis

min

max

Legend

None

Interpolate linear

🖼️ Appearance

Label

☒ Show

Icon

Port labels

Inputs

1. use default label

×

Outputs

1. use default label

×

## 2. Gauge Node

- A node-red ui node that creates a linear gauge with high/low limits and animated sliding pointer. This is very useful for quickly checking many different kinds of processes and how they are performing. Ideally, the pointer should be in the center of the gauge. This would indicate that the process is at it's setpoint.

Example:



## Soil Moisture:

⚙️ Properties

Group

[New dashboard] IoT UI

✎

Size

14 x 8

Type

Gauge

Label

Soil moisture

Value format

{{value}}

Units

units

Range

min 0

max 100

Colour gradient

🖼️ Appearance

Label

☒ Show

Icon

Port labels

Inputs

1. use default label

×

Outputs

none

### 3. Debug Node

- The debug node can be used to display messages in the Debug sidebar within the editor. This provides a structured view of the messages it is sent, making it easier to explore the message.

**Edit debug node**

Delete

Cancel

Done

⚙️ Properties

⚙️ 📄 🖼️

📄 Output

▼ msg. payload

🔗 To

☒ debug window

☐ system console

☐ node status (32 characters)

🏷️ Name

Name

### 4. Button Node

#### Motor On:

⚙️ Properties

⚙️ 📄 🖼️

🏷️ Label

MOTOR ON

📄 Tooltip

optional tooltip

🔥 Color

optional text/icon color

🔥 Background

optional background color

✉️ When clicked, send:

Payload

▼ {} {"command":"motoron"} ...

Topic

▼ msg. topic

➔ If msg arrives on input, emulate a button click:

☐

#### Motor Off:

⚙️ Properties

⚙️ 📄 🖼️

🏷️ Label

MOTOR OFF

📄 Tooltip

optional tooltip

🔥 Color

optional text/icon color

🔥 Background

optional background color

✉️ When clicked, send:

Payload

▼ {} {"command":"motoroff"} ...

Topic

▼ msg. topic

➔ If msg arrives on input, emulate a button click:

☐



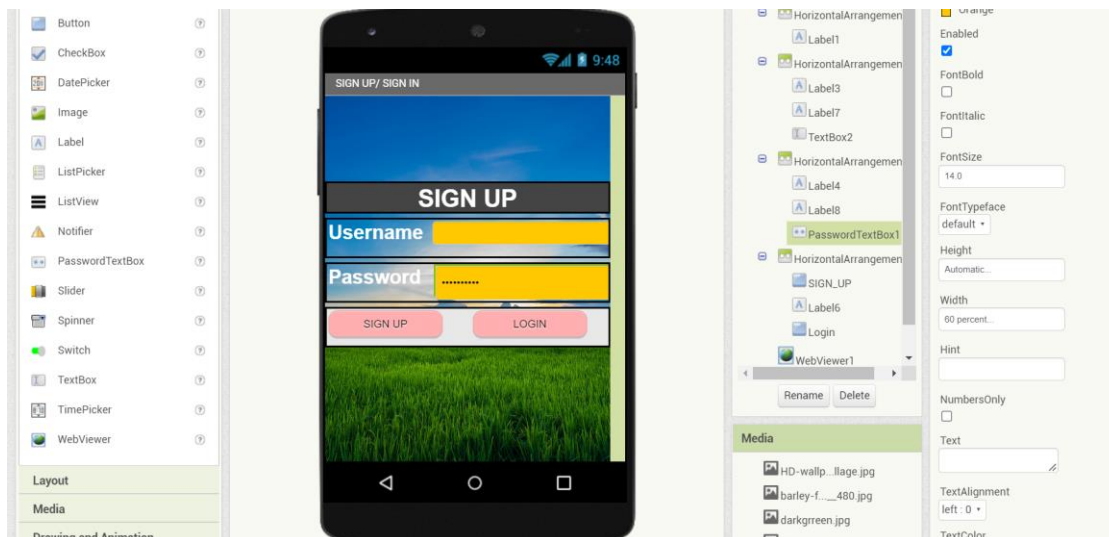
#### 7.4. MIT APP INVENTOR DESIGN

- MIT App Inventor is an intuitive, visual programming environment that allows everyone even children to build fully functional apps for smartphones and tablets.

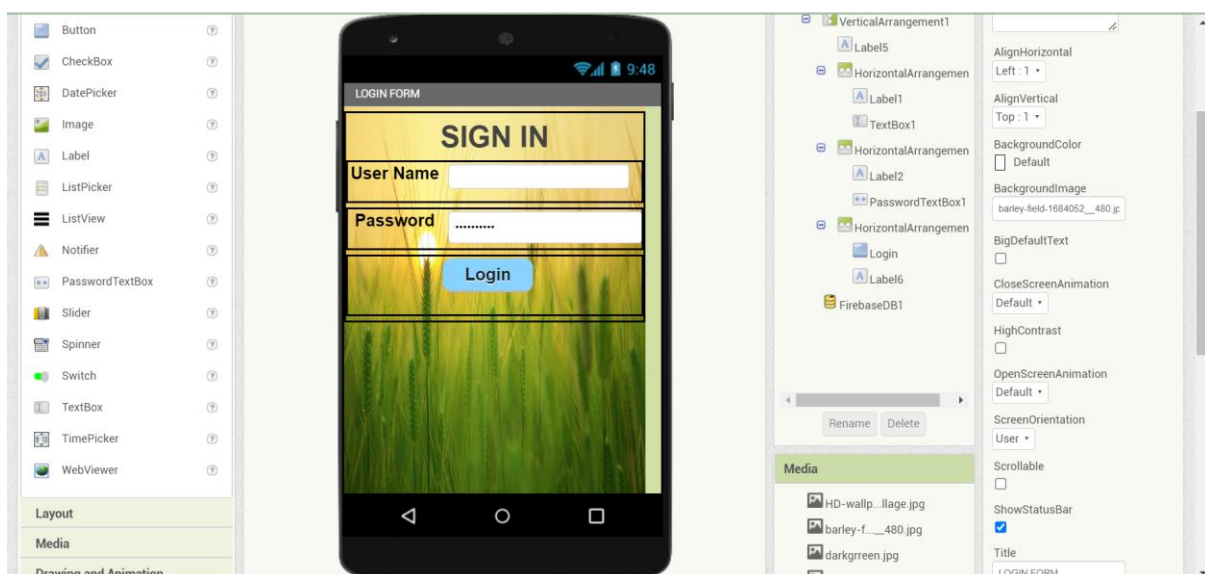
##### STEPS:

1. To get started, go to App Inventor on the web
2. Log in to App Inventor with a username and password
3. Click “continue” to dismiss the splash screen
4. Start a new project and name it
5. You will move to Designer window where you lay out the “user interface” of your app and you can start adding labels.
6. Connect app inventor to your phone for live testing
7. Get the MIT AI2 companion from the play store and install it on your phone or tablet.
8. Get the connection code from App Inventor and scan or type it into your companion app.
9. Switch over to the Blocks editor and make a button click event by adding required functional blocks and url from the Node-RED.

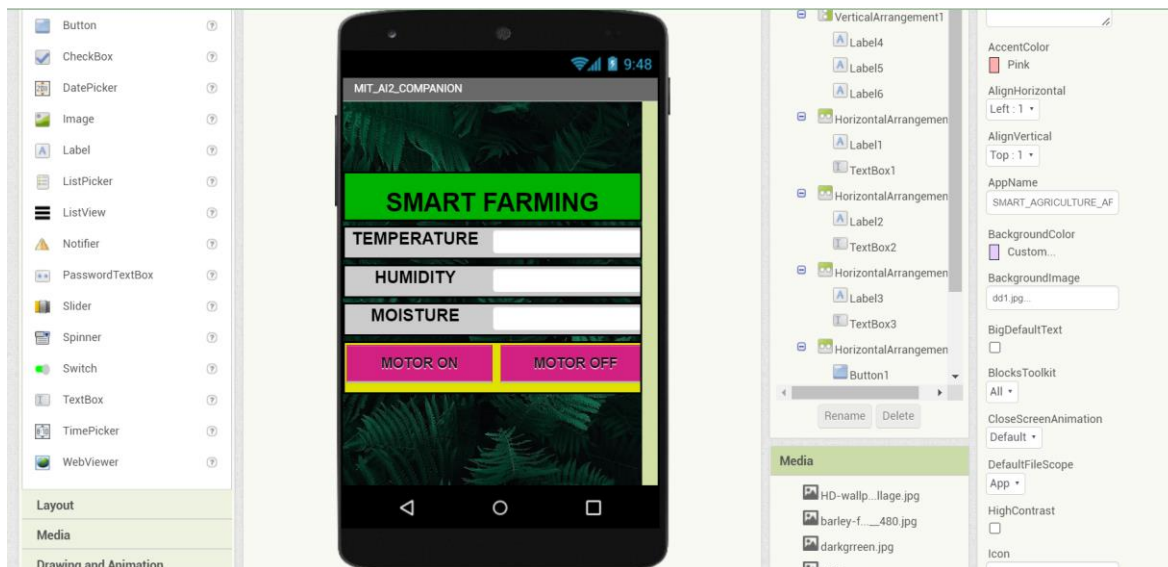
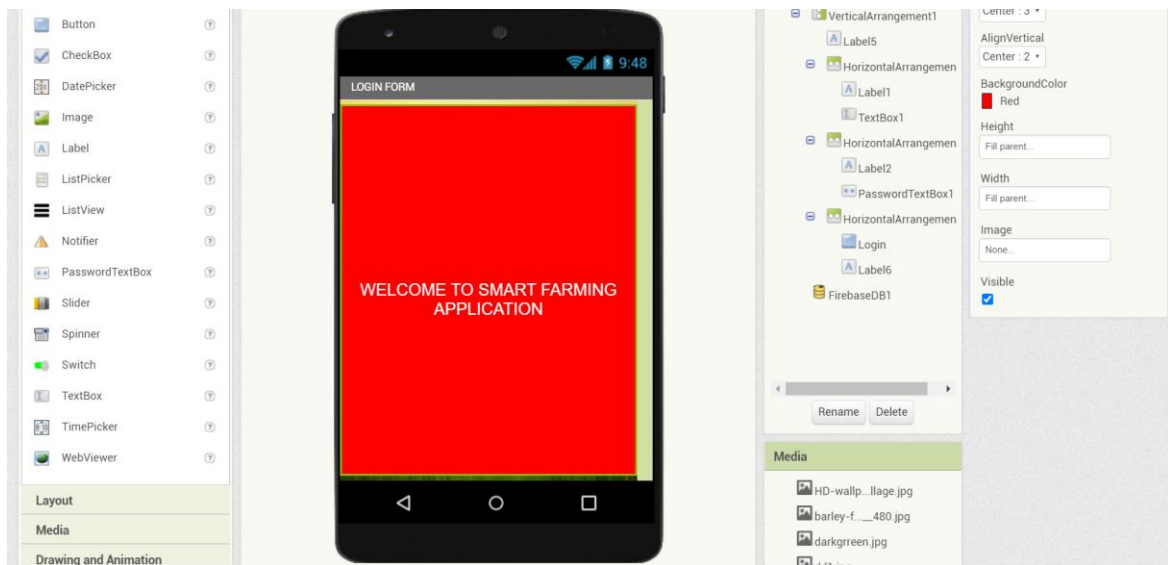
##### Screen 1:



##### Screen 2:

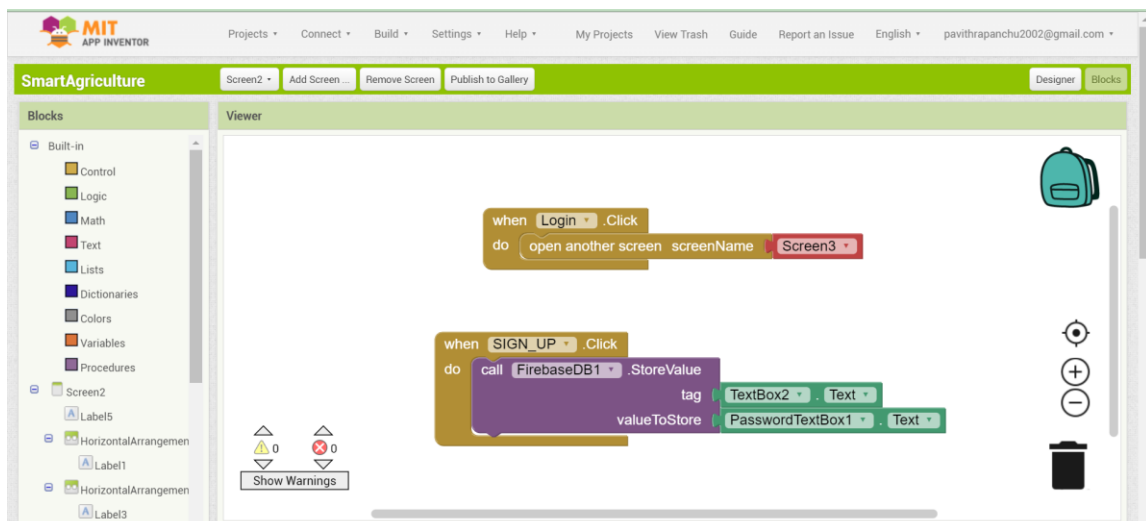


### Screen 3:

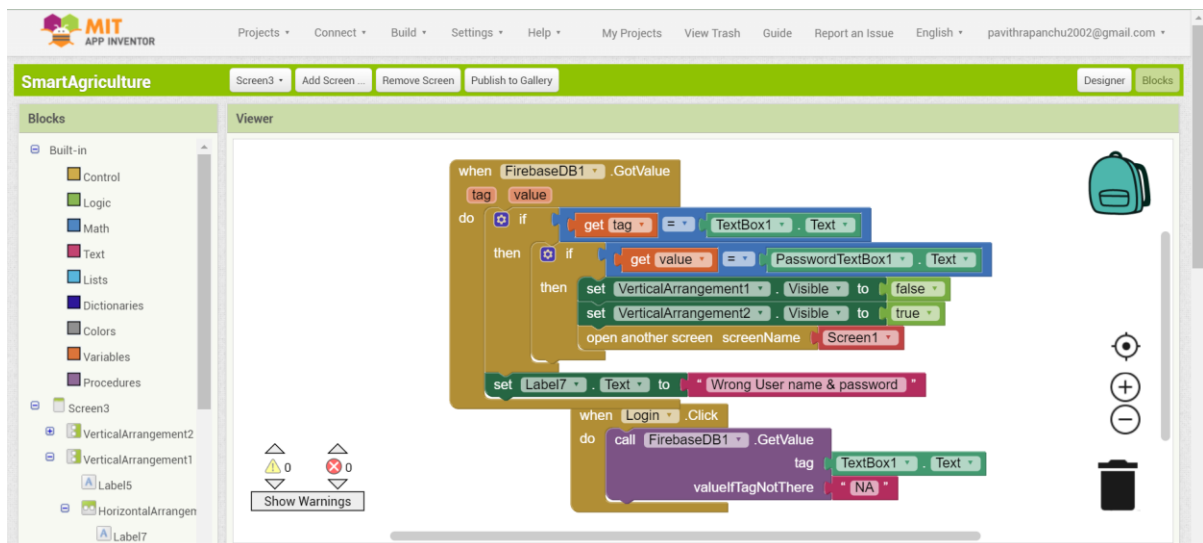


## 7.5. MIT APP INVENTOR BLOCKS

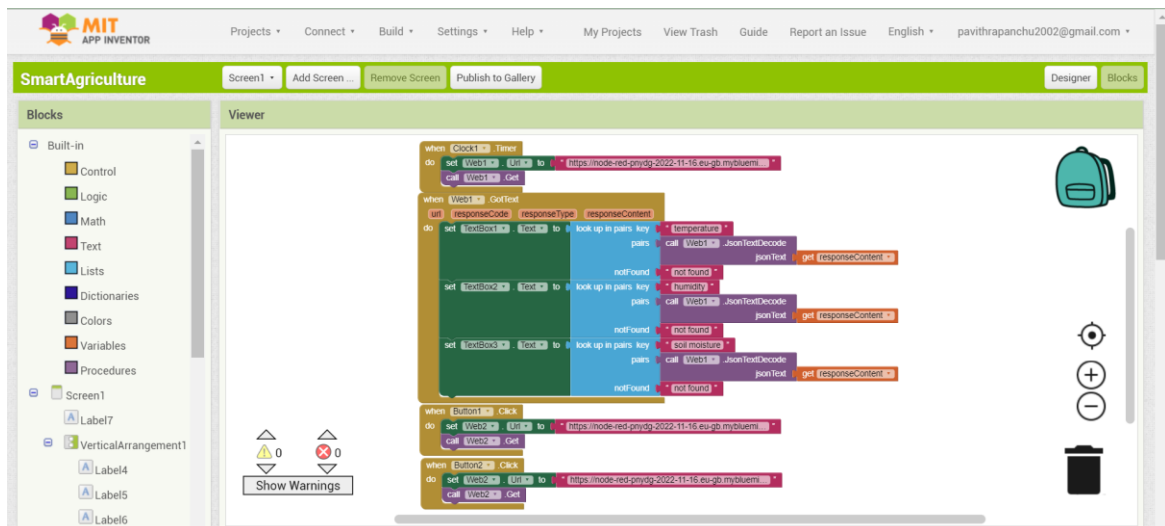
### Screen 1:



## Screen 2:



## Screen 3:



## 7.6. Database Creation

Firebase database was used to store the username and password of user. Only the owner of the application can visit the database to make any changes like addition or deletion of data. This is used for security purpose.

