# SMART IRRIGATION SYSTEM

**1.INTRODUCTION**

**1.1.Overview**

This project will guide you in developing the smart irrigation system project using the Open Weather , Fast2Sms, IBM Cloud Services and MIT app inventor under the IoT Platform.

**1.2.Purpose**

To give the characteristic data of weather and it's parameters and clear description of the present weather. And thereby alerting the user through Fast2Sms and the mobile application - "SMART IRRIGATION".

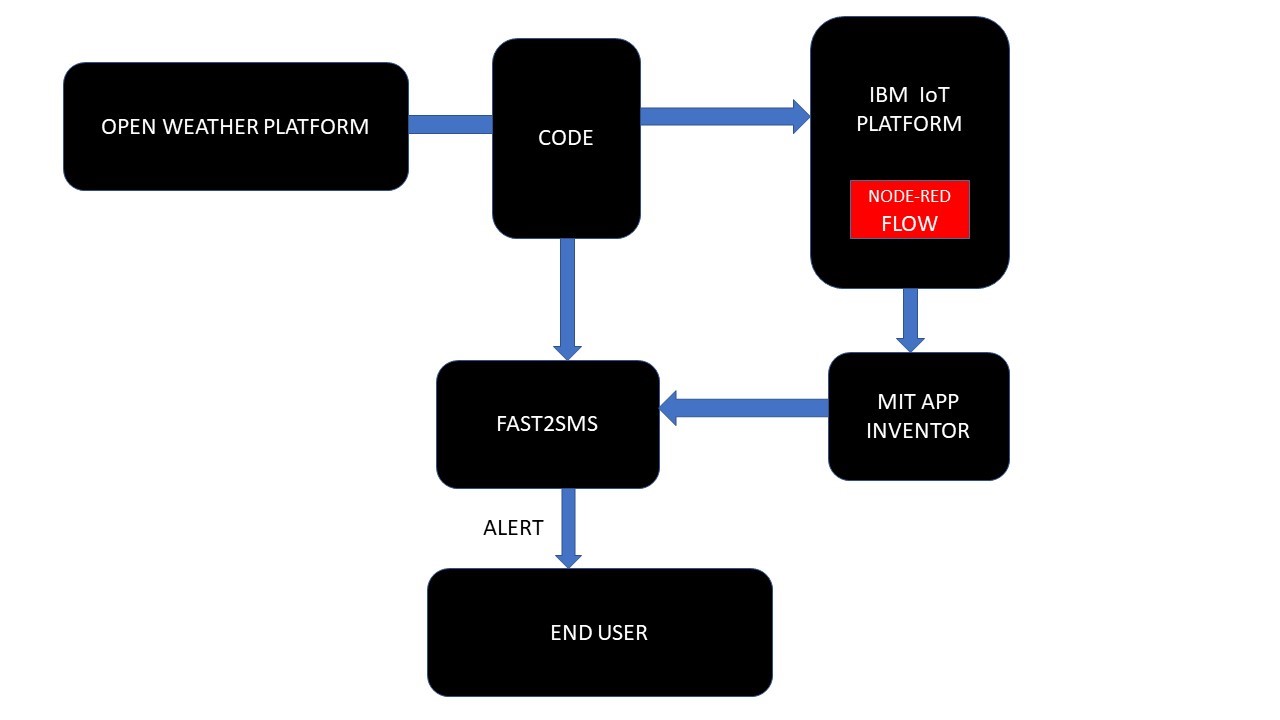
**2. LITERATURE SURVEY**

**2.1 .Existing problem**

Water is being overly used sometimes unnecessarily , which inturn causes shortage when actually the need of water is present. In short , justified usage of water is lacking due to the absence or less presence of knowledge over the weather conditions.

**2.2 .Proposed solution**

Using thedata from openweather, we integrate the ibm node-red platform with the mit app inventor and Fast2Sms to send alerts to the user about the usage of water in a justified way.

**3.THEORITICAL ANALYSIS**  
 **3.1.Block diagram**  


**3.2 .Hardware / Software designing**

**IBM** : Acts as a platform for using various services like "NODE-RED" and also for displaying the values of open weather which come as outputs from the written python code.

**NODE-RED** : Software available within the IBM Platform and helps us in using the MIT app inventor and also has the UI option to display the data and required warnings/commands in the python.Nodes act as the backbone here.

**PYTHON(IDLE)** : Helps us in writing the code for getting the values from Open Weather and for codes to send alert to the user through Fast2Sms.

**OPEN WEATHER PLATFORM** : Has the information about weather and has in-built API'S to use and access the data within it.

**FAST2SMS** : It helps us in sending the alerts/messages to multiple users at same instance of time.

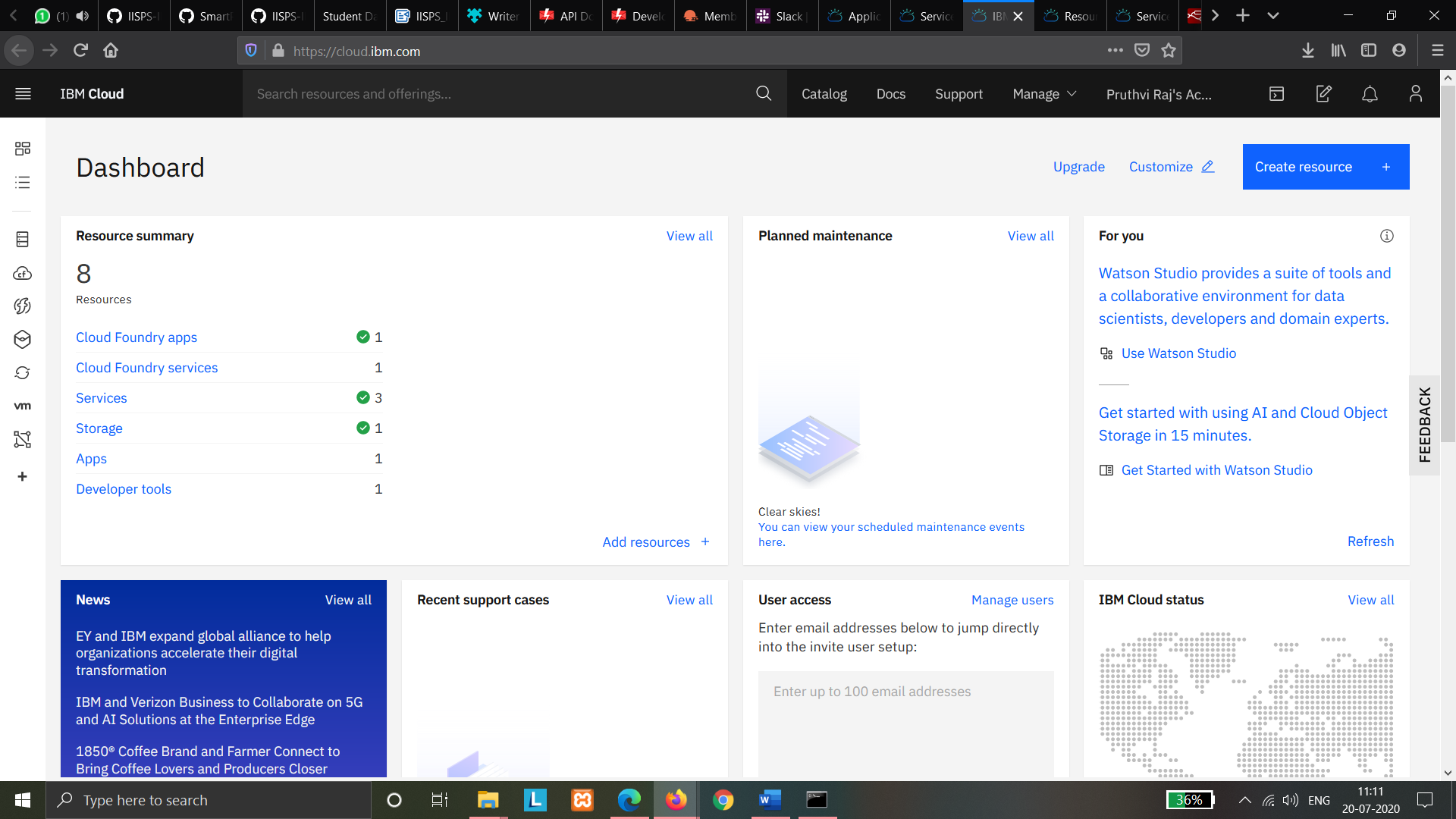
**MIT APP INVENTOR** : This helps us in building the model apk's/applications to simulate the data from IBM platform and also for sending alerts through FAST2SMS .Blocks act as a backbone in this platform.

**SMART IRRIGATION PLATFORM** : It displays the data to the users and thereby alerts and helps them in creating a better agricultural or irrigational facility in their respective areas.

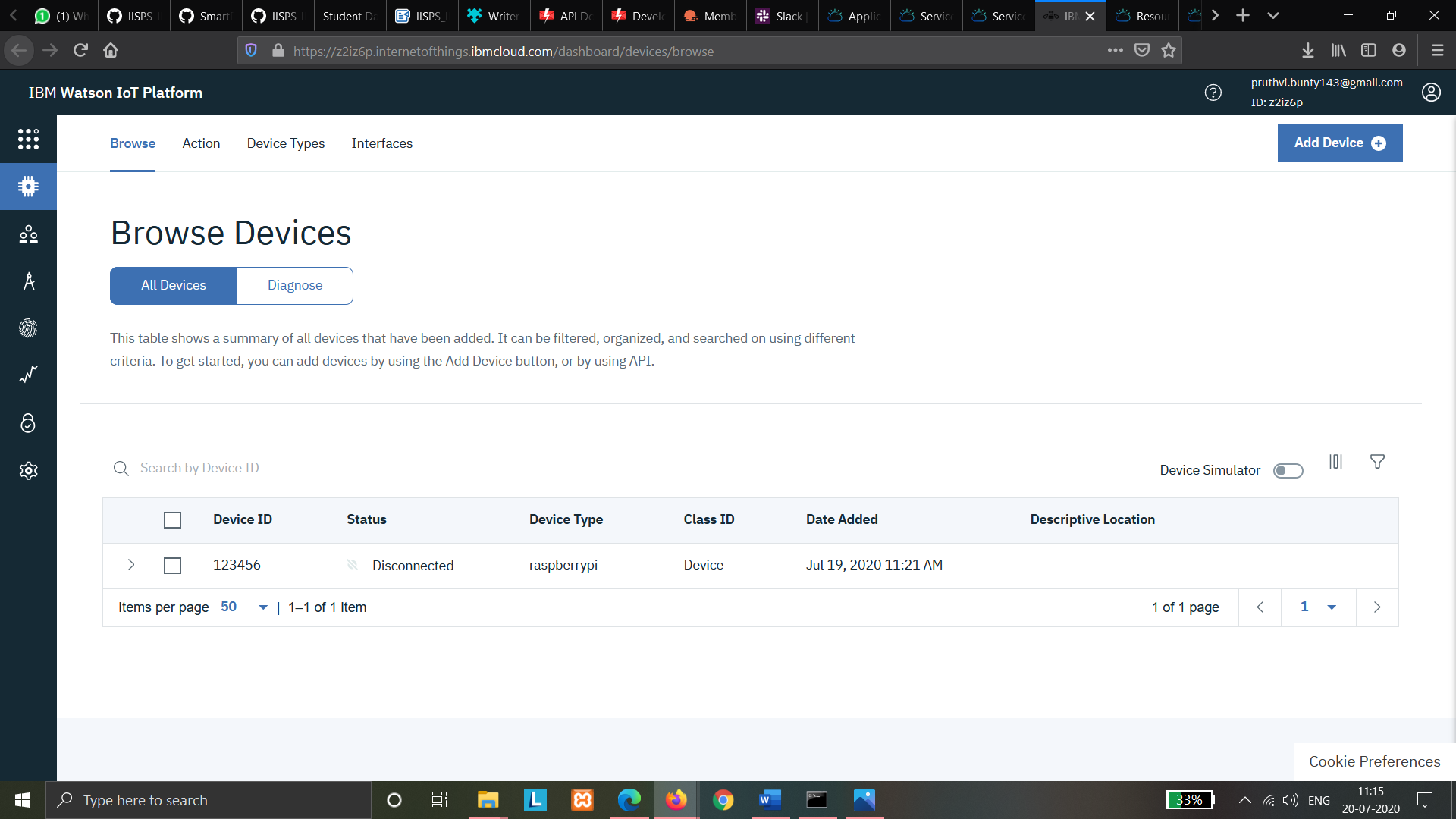
**4.EXPERIMENTAL INVESTIGATIONS**

IBM is a platform which is used to create several services and softwares. It helps to store the data and retrieve it whenever required. Initially using the API key and the URL from the Open weather, we use the python code to get values from the open weather app. Now these values can be further sent to IBM Platform through the python code written in IDLE. We need to create an IBM IOT Service in IBM Cloud Platform to store the open weather values and to perform the actions in Node-red. IBM IOT Service has it's own credentials to get connected with the python code. IBM Cloud also provides us a software platform called Node-Red which can be used for printing the values in debug section. Creation of HTTP requests are also done using node-red. Creating HTTP requests is compulsory for the communication with IBM Platform. Fast2Sms is a service used for sending messages to multiple users at same instance of time. Values of open weather app can be sent through Fast2Sms through python code,as we also have unique API Key for Fast2Sms service. MIT App Inventor can be used to alert the end users through Fast2Sms service about the weather conditions. Alert notifications can be sent through either Fast2Sms or MIT App Inventor so farmer can properly irrigate and give a better produce.

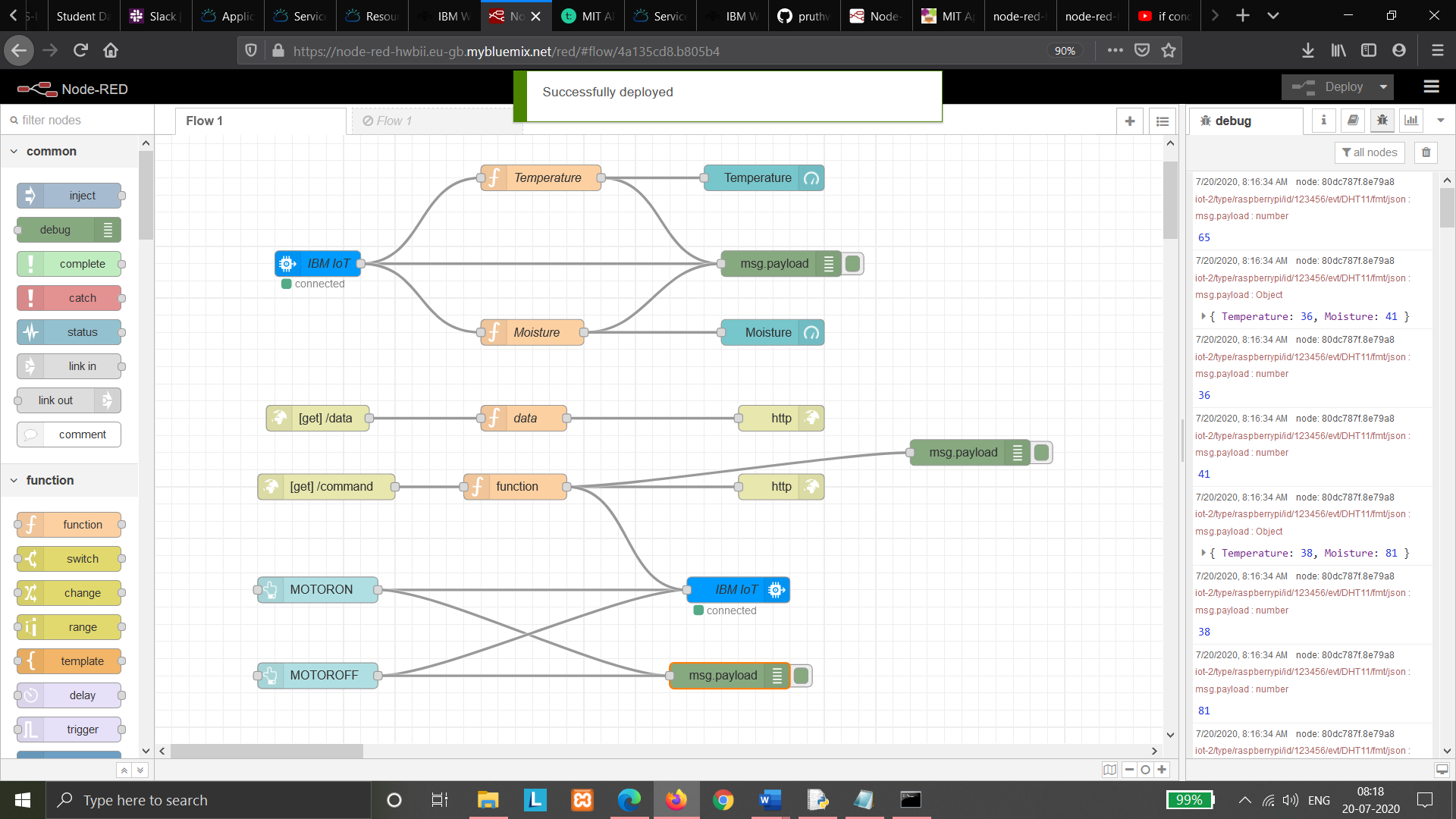
**IBM CLOUD PLATFORM(DASHBOARD)** :



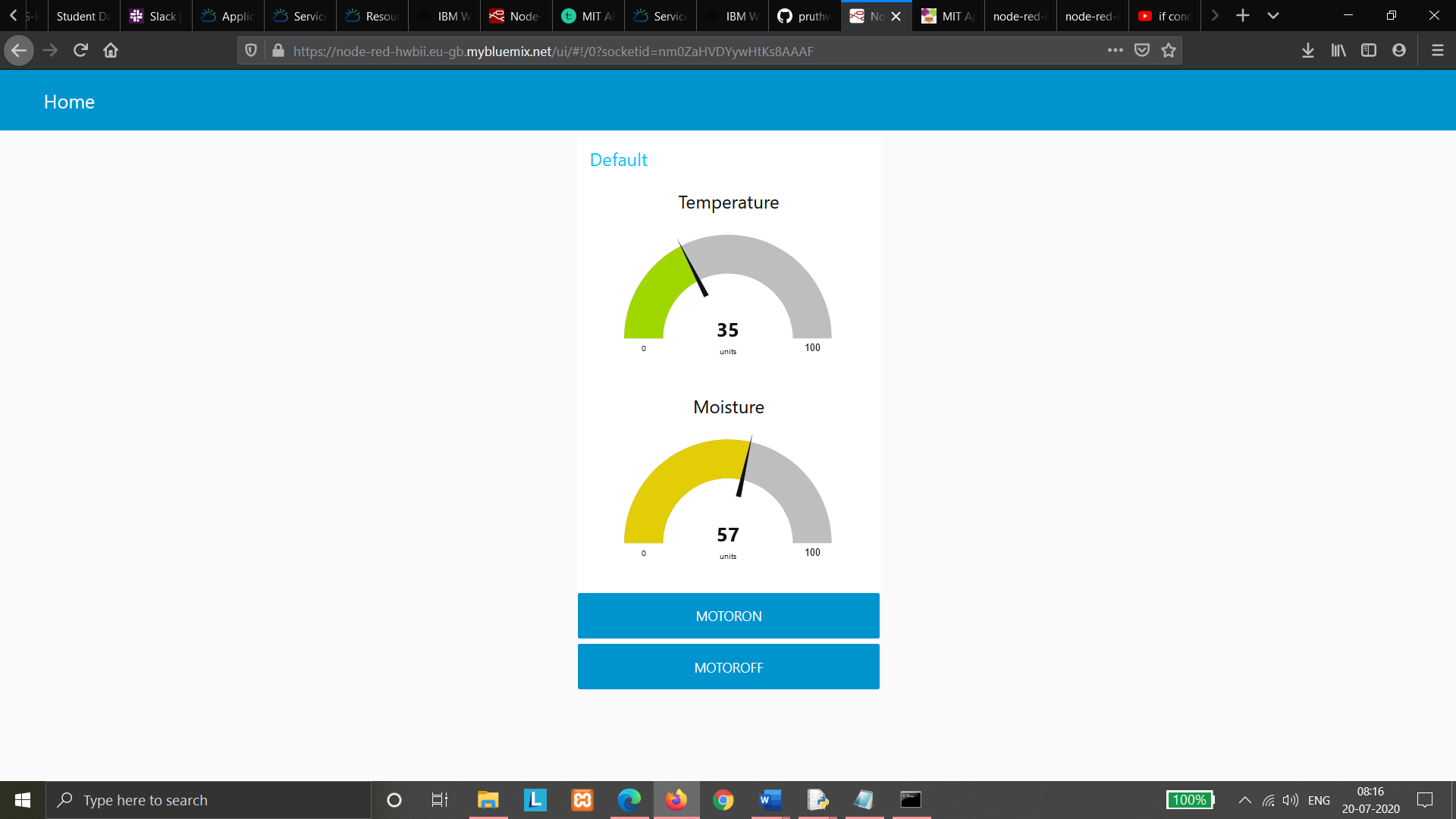
**IBM IoT PLATFORM** :



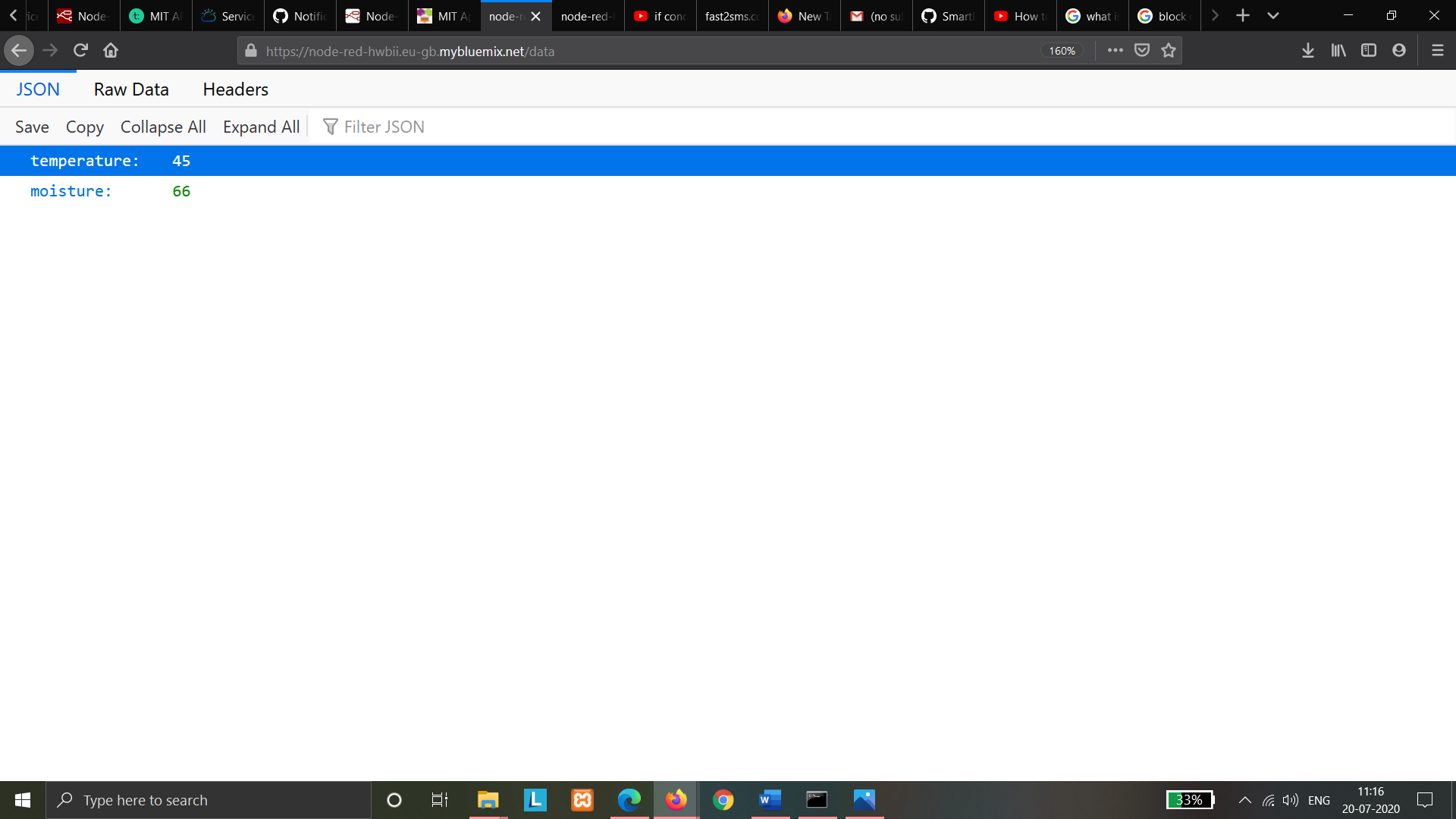
**NODE-RED PLATFORM** :



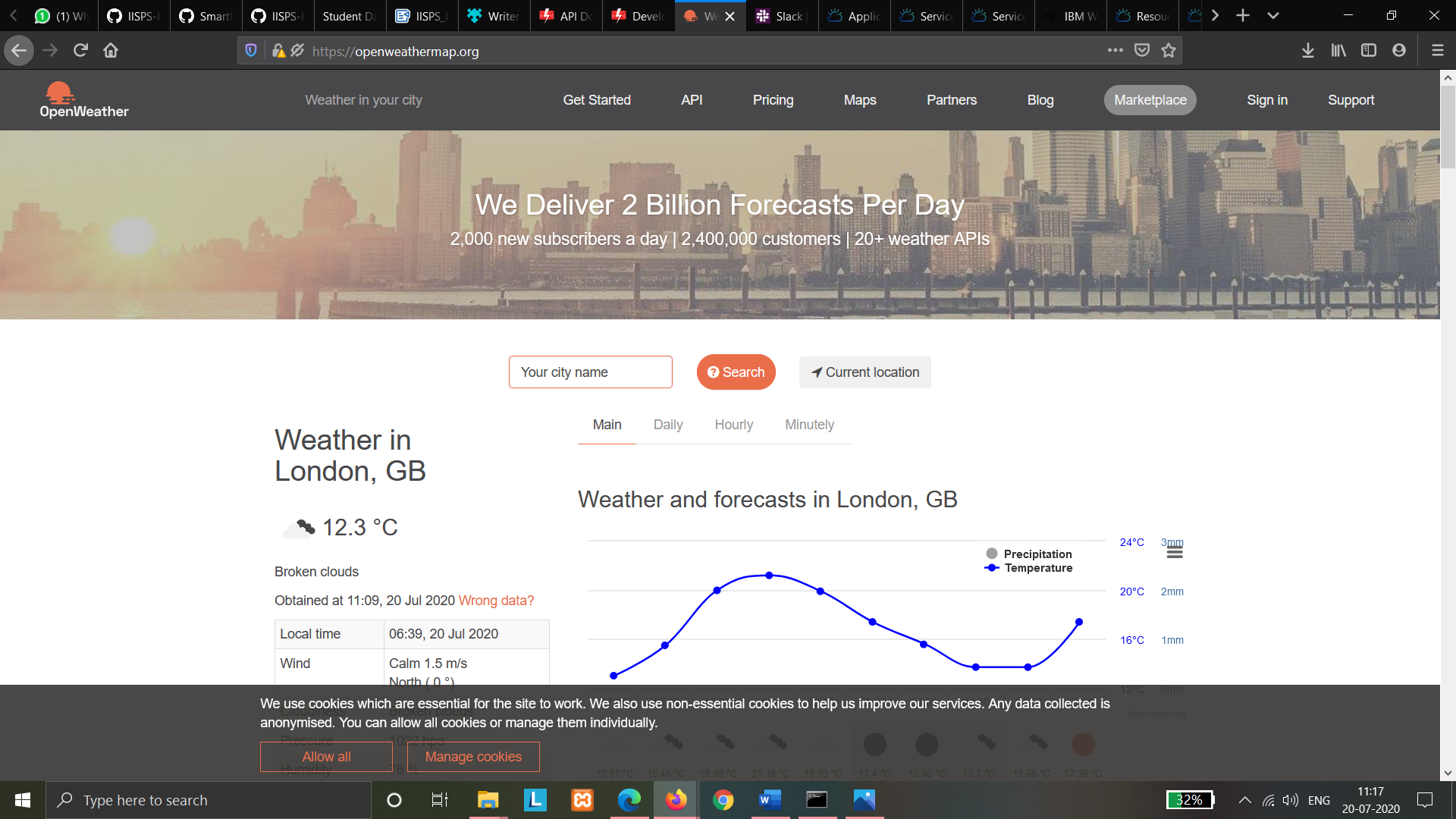
**NODE-RED UI WINDOW** :



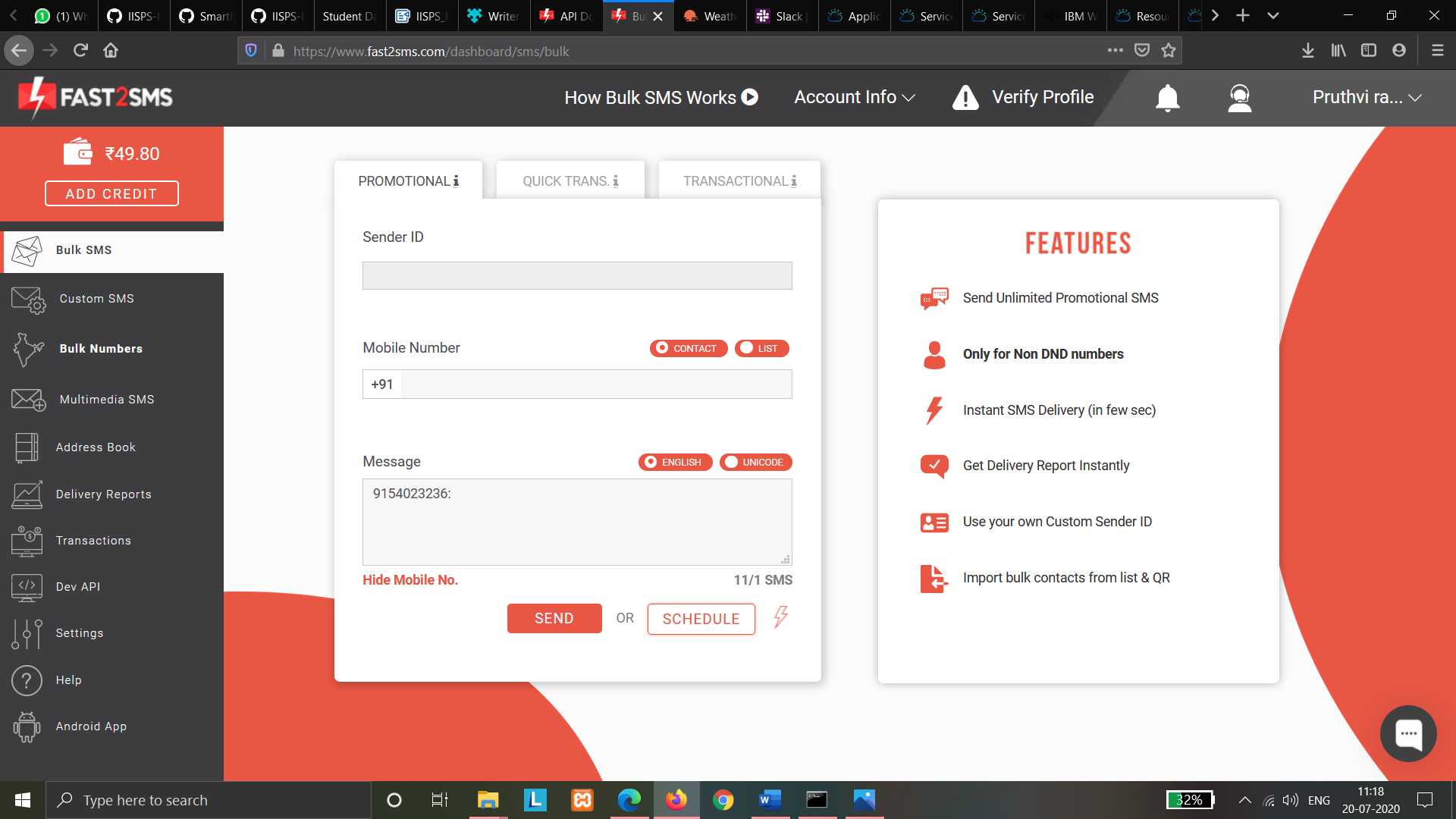
**NODE-RED DATA WINDOW** :



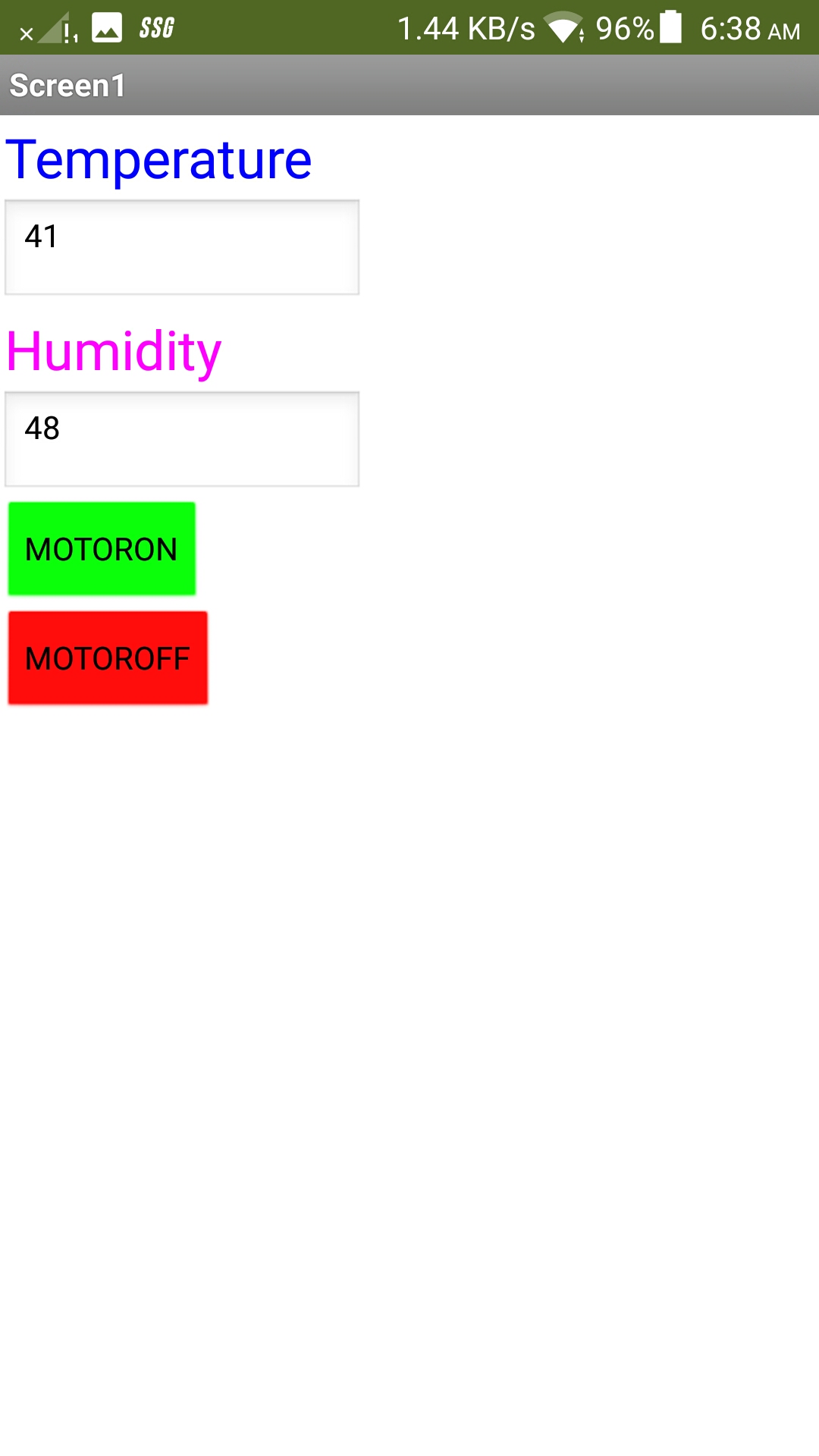
**OPEN WEATHER PLATFORM** :



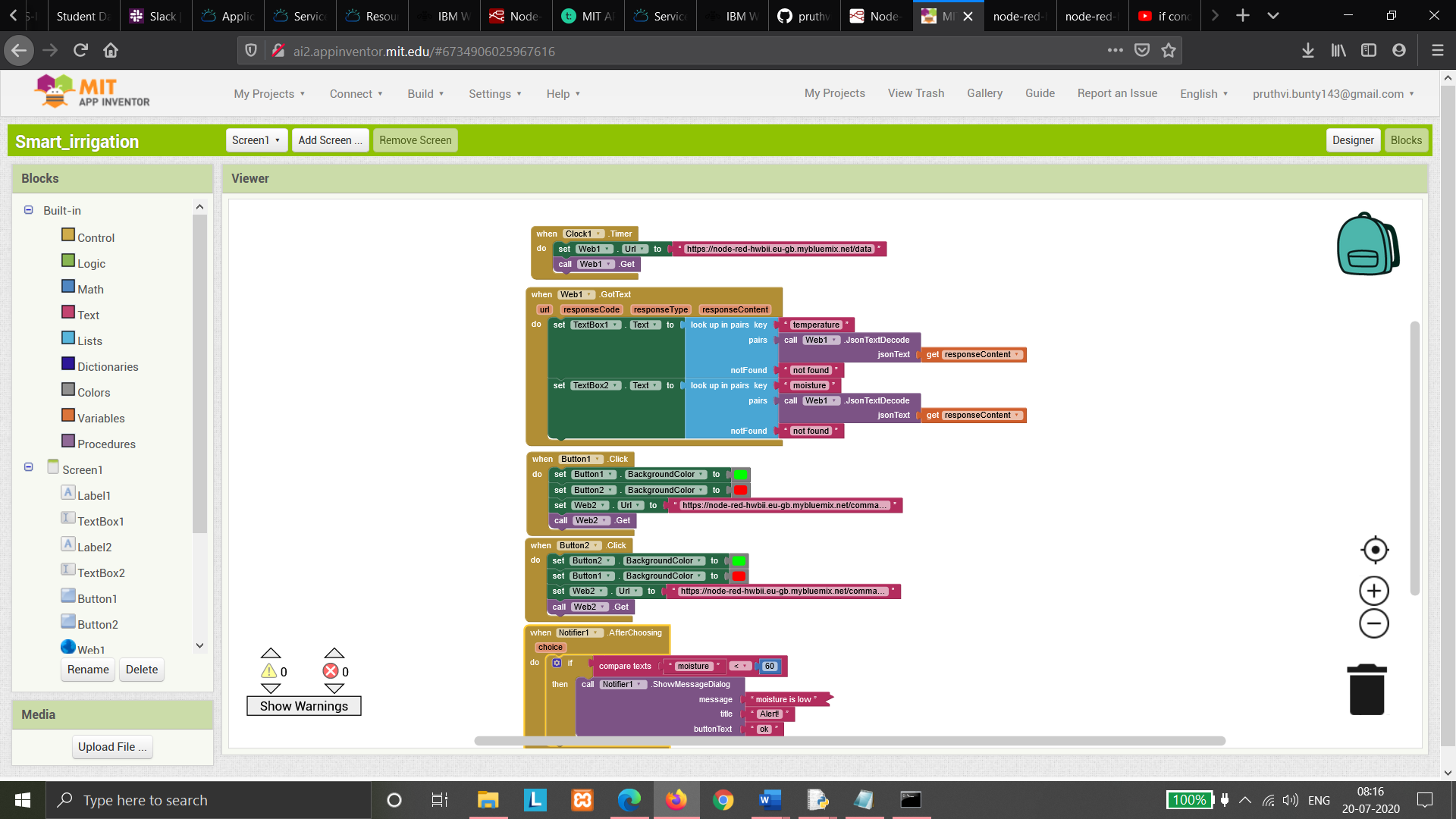
**FAST2SMS** :



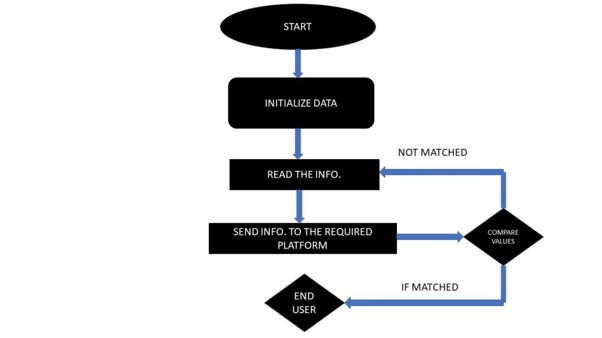
**MIT APP INVENTOR** :

:

**BLOCKS IN MIT** :



**5.FLOWCHART**



**6.RESULT** :

Hence, using the data from open weather, the weather conditions are assessed and the alerts are sent to the user using the Fast2Sms Service either through the values from MIT app inventor or through the code.

**7.1ADVANTAGES** :

>> A Smart Irrigation System prevents unnecessary loss of water.

>> It is very useful in places where water scarcity is a problem.

>> This also helps the End User to monitor the weather conditions.

>> All these inturn will benefit the farmer's economy or helps us to save a lot of

water resources for further cropping and for of future generations.

**7.2DIS ADVANTAGES** :

>> This project requires the availability of Internet, which is a little problem when it

comes to rural areas especially remote areas.

>> Small errors in the data from sensors/other platforms may influence the agricultural produce.

**8.APPLICATIONS :**  
 >>Residential applications such as gardens,small plantations,lawns..  
 >>Smart Greenhouse  
 >>Open Field  
 >>Golf Courses,Turf..

**9.CONCLUSION** :

A Smart Irrigation System prevents unnecessary loss of water and provides the information about the current weather conditions. This project alerts the End User whenever it is required and thereby creating a chance for better irrigational facilities and a chance for having better agricultural produce.

**10.FUTURE SCOPE** :

Agriculture forms the base of an economy. Better the agricultural system, better is the economy. Agriculture provides raw materials to many industries which form the backbone of the nation. Most importantly, it helps a nation to strive by providing food and other agrarian products. India, for example, which is primarily an agrarian country, has a bulk of the working population engaged in agricultural activities.

Agriculture is important not only for the Supply of Food but also for the Provision of Raw Materials for other Industries such as Textiles, Sugar, Jute, Vegetable oil and Tobacco. Agriculture is not only an Occupation for People but also a Way of life. Most Customs and Cultures in the World revolve around Agriculture. Irrigation plays a major role in any type of agricultural activities.

Better is the Irrigation - better will be the Agriculture. As we are already suffering with severe global warming and scarcity of water, we have a great responsibility of providing a better amount of water resources and irrigational facilities to our younger generations. Smart Irrigation System directly or indirectly plays a prominent role in doing so.

**11.BIBLIOGRAPHY** :

>> Some of the websites like openweathermap.org , www.fast2sms.com , cloud.ibm.com and appinventor.mit.edu .

>> Also the readily available "PYTHON IDLE" is another source which helped us in the project.

**12.APPENDIX**

**A. Source code**

**CODE FOR FAST2SMS** :

1. import requests
2. url = "https://www.fast2sms.com/dev/bulk"
3. querystring = {"authorization":"FPYnw1x9CrHcIk6R3AqTpOeZzvi85lQyjLa4dDJgMGVmf02suKFjdvKuCfeWxa8oY59y6Xw1b3R7UlJH","sender\_id":"FSTSMS","message":"humidity range: Temperature range","language":"english","route":"p","numbers":"9490678933,9398659128,9381274122,9676926774"}
4. headers = {'cache-control': "no-cache"}
5. response = requests.request("GET", url, headers=headers, params=querystring)
6. print(response.text)

**CODE FOR TAKING DATA FROM OPEN WEATHER** :

1. import requests
2. api\_address='http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID=403dec552ad1128a08f340f6d83f93d3'
3. city=input("city name :")
4. url = api\_address + city
5. json\_data = requests.get(url).json()
6. print(json\_data)

**CODE FOR TAKING THE DATA FROM OPEN WEATHER AND SENDING IT TO THE USER USING FAST2SMS :**

1. import requests, json
2. api\_key = "403dec552ad1128a08f340f6d83f93d3"
3. base\_url = "http://api.openweathermap.org/data/2.5/weather?"
4. url = "https://www.fast2sms.com/dev/bulk"#fast2sms
5. city\_name = input("Enter city name : ")
6. complete\_url = base\_url + "appid=" + api\_key + "&q=" + city\_name
7. response = requests.get(complete\_url)
8. x = response.json()
9. if x["cod"] != "404":
10. y = x['main']
11. current\_temperature = y["temp"]
12. current\_humidity = y["moisture"]
13. print(" Temperature (in kelvin unit) = " +
14. str(current\_temperature) +
15. "\n humidity (in percentage) = " +
16. str(current\_moisture) +
17. querystring = {"authorization":"FPYnw1x9CrHcIk6R3AqTpOeZzvi85lQyjLa4dDJgMGVmf02suKFjdvKuCfeWxa8oY59y6Xw1b3R7UlJH","sender\_id":"FSTSMS","message":"Humidity: "+str(current\_humidiy)+"Temperature: "+str(current\_temperature),"language":"english","route":"p","numbers":"9154023236"}
18. headers={
19. 'cache-control': "no-cache"
20. }
21. response = requests.request("GET", url, headers=headers, params=querystring)
22. print(response.text)
23. else:
24. print(" City Not Found ")

**CODE FOR INTEGRATION OF OPEN WEATHER, FAST2SMS AND IBMIOT PLATFORM :**

1. import time
2. import sys
3. import ibmiotf.application
4. import ibmiotf.device
5. import random
6. import requests
7. organization = "z2iz6p"
8. deviceType = "raspberrypi"
9. deviceId = "123456"
10. authMethod = "token"
11. authToken = "12345678"
12. def myCommandCallback(cmd):
13. print("Command received: %s" % cmd.data)
14. print(type(cmd.data))
15. i=cmd.data('command')
16. if i=='motoron':
17. print("motor is on")
18. elif i=='motoroff':
19. print("motor is off")
20. try:
21. deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-token": authToken}
22. deviceCli = ibmiotf.device.Client(deviceOptions)
24. except Exception as e:
25. print("Caught exception connecting device: %s" % str(e))
26. sys.exit()
27. deviceCli.connect()
28. while True:
29. hum=random.randint(40, 90)
30. temp =random.randint(30, 45)
31. data = { 'Temperature' : temp, 'Moisture': hum }
32. if hum<=50:
33. r=requests.get('https://www.fast2sms.com/dev/bulk?authorization=FPYnw1x9CrHcIk6R3AqTpOeZzvi85lQyjLa4dDJgMGVmf02suKFjdvKuCfeWxa8oY59y6Xw1b3R7UlJH&sender\_id=FSTSMS&message=turnmotoron&language=english&route=p&number=9154023236')
34. print(r.status\_code)
35. def myOnPublishCallback():
36. print ("Published Temperature = %s C" % temp, "Moisture = %s %%" % hum, "to IBM Watson")
37. success = deviceCli.publishEvent("DHT11", "json", data, qos=0, on\_publish=myOnPublishCallback)
38. if not success:
39. print("Not connected to IoTF")
40. time.sleep(2)
41. deviceCli.commandCallback = myCommandCallback
42. deviceCli.disconnect()

**B. UI output**

