**Flood Monitoring and Early Warning System.**

**Phase-5**

Here, we document our project and prepare it for submission.

Tasks:

Documentation:

● Describe the project's objectives, IoT sensor deployment, platform development, and code implementation.

● Include diagrams, schematics, and screenshots of the IoT sensors, transit information platform, and real-time data display.

● Explain how the real-time transit information system can improve the Early Warning system and managing the water level.

Submission:

● Share the GitHub repository link containing the project's code and files.

● Provide instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate those using Python.

● Include example outputs of IoT sensor data transmission, platform UI, and real-time data updates.

**Flood Monitoring and Early Warning System.**

1. **Overview:**

**Introduction:**

The “Flood Monitoring and Early Warning” project aims to revolutionize the difficulties that are faced by the public during floods by using the power of IoT (Internet of Things) technology. IoT sensors such as ultrasonic sensor and other sensor that operates on RADAR technologies are used to monitor the water level. This data is stored and processed to predict the future water levels. This project seeks to improve the safety, reliability, and overall wellbeing of public for the residences leaving near flood-prone areas. The key concept of this project is to create an efficient and comprehensive flood monitoring and early warning system to protect flood-prone areas and to reduce the risks to residents and infrastructure.

**Technical Approach:**

* **IoT Sensor System Design:**

Design and deploy IoT sensors (Ultrasonic Sensors) on dams and other water bodies like river and lake. The results from the sensor is sent for data processing.

* **Data Processing and Analysis:**

Develop data processing algorithms to clean, aggregate, and analyse the collected data. Utilize machine learning techniques to predict the water levels.

* **Real-Time Transit Information Platform:**

The water level is continuously sent to the public through SMS alerts. Call alerts are sent when there are high possibility of flood.

**2.Project Components**

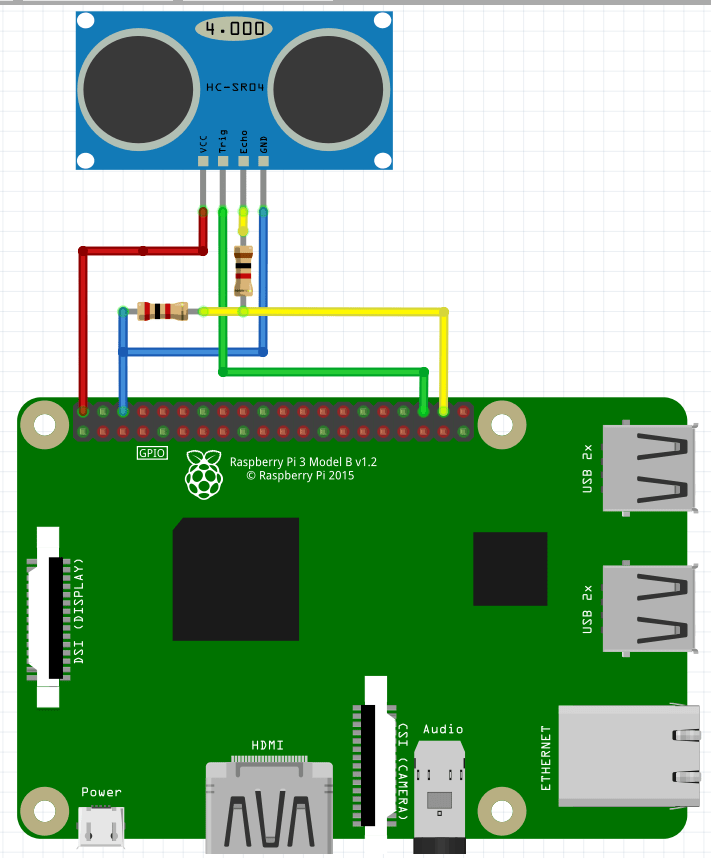
**IOT Sensors**

* ESP8266 NodeMCU
* Ultrasonic Sensor
* LEDs (Red & Green)
* Breadboard
* Jumpers

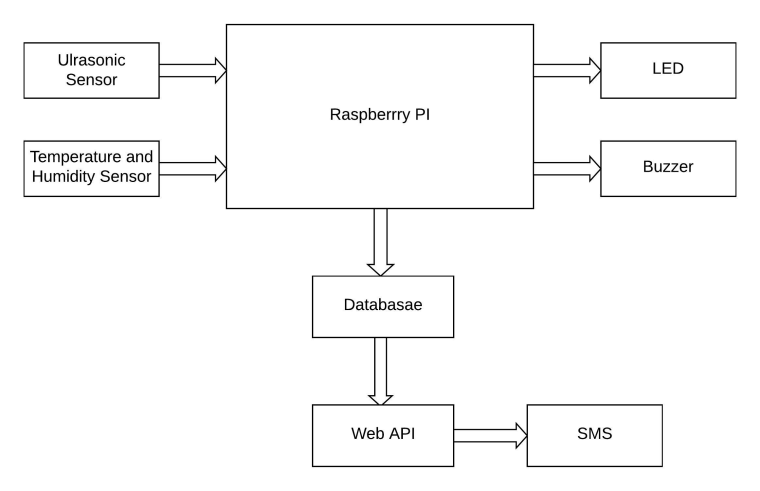
Arduino Module:

The Arduino module is responsible for collecting the data from the ultrasonic sensors and transmit the data to the IoT Platform.

**3.IoT Device setup:**



**4.Schematic Diagram:**



5.Flow chart:

**5.Code Implementation:**

* DHT11

import time

import RPi

class DHT11Result:

'DHT11 sensor result returned by DHT11.read() method'

ERR\_NO\_ERROR = 0

ERR\_MISSING\_DATA = 1

ERR\_CRC = 2

error\_code = ERR\_NO\_ERROR

temperature = -1

humidity = -1

def \_\_init\_\_(self, error\_code, temperature, humidity):

self.error\_code = error\_code

self.temperature = temperature

self.humidity = humidity

def is\_valid(self):

return self.error\_code == DHT11Result.ERR\_NO\_ERROR

class DHT11:

'DHT11 sensor reader class for Raspberry'

\_\_pin = 0

def \_\_init\_\_(self, pin):

self.\_\_pin = pin

def read(self):

RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.OUT)

# send initial high

self.\_\_send\_and\_sleep(RPi.GPIO.HIGH, 0.05)

# pull down to low

self.\_\_send\_and\_sleep(RPi.GPIO.LOW, 0.02)

# change to input using pull up

RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.IN, RPi.GPIO.PUD\_UP)

# collect data into an array

data = self.\_\_collect\_input()

# parse lengths of all data pull up periods

pull\_up\_lengths = self.\_\_parse\_data\_pull\_up\_lengths(data)

# if bit count mismatch, return error (4 byte data + 1 byte checksum)

if len(pull\_up\_lengths) != 40:

return DHT11Result(DHT11Result.ERR\_MISSING\_DATA, 0, 0)

# calculate bits from lengths of the pull up periods

bits = self.\_\_calculate\_bits(pull\_up\_lengths)

# we have the bits, calculate bytes

the\_bytes = self.\_\_bits\_to\_bytes(bits)

# calculate checksum and check

checksum = self.\_\_calculate\_checksum(the\_bytes)

if the\_bytes[4] != checksum:

return DHT11Result(DHT11Result.ERR\_CRC, 0, 0)

# ok, we have valid data

# The meaning of the return sensor values

# the\_bytes[0]: humidity int

# the\_bytes[1]: humidity decimal

# the\_bytes[2]: temperature int

# the\_bytes[3]: temperature decimal

temperature = the\_bytes[2] + float(the\_bytes[3]) / 10

humidity = the\_bytes[0] + float(the\_bytes[1]) / 10

return DHT11Result(DHT11Result.ERR\_NO\_ERROR, temperature, humidity)

def \_\_send\_and\_sleep(self, output, sleep):

RPi.GPIO.output(self.\_\_pin, output)

time.sleep(sleep)

def \_\_collect\_input(self):

# collect the data while unchanged found

unchanged\_count = 0

# this is used to determine where is the end of the data

max\_unchanged\_count = 100

last = -1

data = []

while True:

current = RPi.GPIO.input(self.\_\_pin)

data.append(current)

if last != current:

unchanged\_count = 0

last = current

else:

unchanged\_count += 1

if unchanged\_count > max\_unchanged\_count:

break

return data

def \_\_parse\_data\_pull\_up\_lengths(self, data):

STATE\_INIT\_PULL\_DOWN = 1

STATE\_INIT\_PULL\_UP = 2

STATE\_DATA\_FIRST\_PULL\_DOWN = 3

STATE\_DATA\_PULL\_UP = 4

STATE\_DATA\_PULL\_DOWN = 5

state = STATE\_INIT\_PULL\_DOWN

lengths = [] # will contain the lengths of data pull up periods

current\_length = 0 # will contain the length of the previous period

for i in range(len(data)):

current = data[i]

current\_length += 1

if state == STATE\_INIT\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# ok, we got the initial pull down

state = STATE\_INIT\_PULL\_UP

continue

else:

continue

if state == STATE\_INIT\_PULL\_UP:

if current == RPi.GPIO.HIGH:

# ok, we got the initial pull up

state = STATE\_DATA\_FIRST\_PULL\_DOWN

continue

else:

continue

if state == STATE\_DATA\_FIRST\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# we have the initial pull down, the next will be the data pull up

state = STATE\_DATA\_PULL\_UP

continue

else:

continue

if state == STATE\_DATA\_PULL\_UP:

if current == RPi.GPIO.HIGH:

# data pulled up, the length of this pull up will determine whether it is 0 or 1

current\_length = 0

state = STATE\_DATA\_PULL\_DOWN

continue

else:

continue

if state == STATE\_DATA\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# pulled down, we store the length of the previous pull up period

lengths.append(current\_length)

state = STATE\_DATA\_PULL\_UP

continue

else:

continue

return lengths

def \_\_calculate\_bits(self, pull\_up\_lengths):

# find shortest and longest period

shortest\_pull\_up = 1000

longest\_pull\_up = 0

for i in range(0, len(pull\_up\_lengths)):

length = pull\_up\_lengths[i]

if length < shortest\_pull\_up:

shortest\_pull\_up = length

if length > longest\_pull\_up:

longest\_pull\_up = length

# use the halfway to determine whether the period it is long or short

halfway = shortest\_pull\_up + (longest\_pull\_up - shortest\_pull\_up) / 2

bits = []

for i in range(0, len(pull\_up\_lengths)):

bit = False

if pull\_up\_lengths[i] > halfway:

bit = True

bits.append(bit)

return bits

def \_\_bits\_to\_bytes(self, bits):

the\_bytes = []

byte = 0

for i in range(0, len(bits)):

byte = byte << 1

if (bits[i]):

byte = byte | 1

else:

byte = byte | 0

if ((i + 1) % 8 == 0):

the\_bytes.append(byte)

byte = 0

return the\_bytes

def \_\_calculate\_checksum(self, the\_bytes):

return the\_bytes[0] + the\_bytes[1] + the\_bytes[2] + the\_bytes[3] & 255

Main:

import RPi.GPIO as GPIO

import dht11

import time

import datetime

# initialize GPIO

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

# read data using pin 14

instance = dht11.DHT11(pin=14)

GPIO.setup(16,GPIO.OUT)

def THRESHOLD():

return 30

class ultraSound:

TRIG = 20

ECHO = 21

#LED=16

def \_\_init\_\_(self):

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(self.TRIG,GPIO.OUT)

GPIO.setup(self.ECHO,GPIO.IN)

#GPIO.setup(self.LED,GPIO.OUT)

def readData(self):

GPIO.output(self.TRIG, False)

#print "Waitng For Sensor To Settle"

time.sleep(2)

GPIO.output(self.TRIG, True)

time.sleep(0.00001)

GPIO.output(self.TRIG, False)

pulse\_start=0;

pulse\_end=0;

while GPIO.input(self.ECHO)==0:

pulse\_start = time.time()

while GPIO.input(self.ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

time.sleep(0.4)

return distance;

sensor=ultraSound();

while True:

result = instance.read()

if result.is\_valid():

print("Last valid input: " + str(datetime.datetime.now()))

print("Temperature: %-3.1f C" % result.temperature)

print("Humidity: %-3.1f %%" % result.humidity)

time.sleep(3)

print"----------------------------------------------------"

sensor.readData();

if(sensor.readData()<=THRESHOLD()):

GPIO.output(16,GPIO.HIGH)

print "WARNING THE LEVEL OF WATER HAS RAISED BEYOND THE THRESHOLD LEVEL"

else:

GPIO.output(16,GPIO.LOW)

print"----------------------------------------------------"

Database**:**

import RPi.GPIO as GPIO

import dht11

import time

import datetime

import MySQLdb

# initialize GPIO

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

# read data using pin 14

instance = dht11.DHT11(pin=14)

GPIO.setup(16,GPIO.OUT)

def THRESHOLD():

return 20

class ultraSound:

TRIG = 20

ECHO = 21

#LED=16

def \_\_init\_\_(self):

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(self.TRIG,GPIO.OUT)

GPIO.setup(self.ECHO,GPIO.IN)

#GPIO.setup(self.LED,GPIO.OUT)

def readData(self):

GPIO.output(self.TRIG, False)

#print "Waitng For Sensor To Settle"

time.sleep(2)

GPIO.output(self.TRIG, True)

time.sleep(0.00001)

GPIO.output(self.TRIG, False)

pulse\_start=0;

pulse\_end=0;

while GPIO.input(self.ECHO)==0:

pulse\_start = time.time()

while GPIO.input(self.ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

time.sleep(0.4)

return distance;

sensor=ultraSound();

while True:

remarks\_val="Nothing to Worry"

result = instance.read()

distance=sensor.readData();

dateandtime=datetime.datetime.now()

if result.is\_valid():

print("Time: " + str(datetime.datetime.now()))

print("Temperature: %-3.1f C" % result.temperature)

print("Humidity: %-3.1f %%" % result.humidity)

print "Distance:",distance,"cm"

time.sleep(3)

print"----------------------------------------------------"

if(distance<=THRESHOLD()):

GPIO.output(16,GPIO.HIGH)

print "WARNING THE LEVEL OF WATER HAS RAISED BEYOND THE THRESHOLD LEVEL"

remarks\_val="WARNING THE LEVEL OF WATER HAS RAISED BEYOND THE THRESHOLD LEVEL"

else:

GPIO.output(16,GPIO.LOW)

print"----------------------------------------------------"

conn=MySQLdb.connect(host="localhost",user="root",passwd="shrestha5555",db="minor")

cur=conn.cursor()

sql\_query="""INSERT INTO Data (DateTime,Temperature,Humidity,Distance,Remarks) VALUES (%s,%s,%s,%s,%s)"""

cur.execute(sql\_query,(dateandtime,result.temperature,result.humidity,distance,remarks\_val))

conn.commit()

conn.close()

Ultrasonic**:**

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(16,GPIO.OUT)

def THRESHOLD():

return 30

class ultraSound:

TRIG = 20

ECHO = 21

#LED=16

def \_\_init\_\_(self):

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(self.TRIG,GPIO.OUT)

GPIO.setup(self.ECHO,GPIO.IN)

#GPIO.setup(self.LED,GPIO.OUT)

def readData(self):

GPIO.output(self.TRIG, False)

#print "Waitng For Sensor To Settle"

time.sleep(2)

GPIO.output(self.TRIG, True)

time.sleep(0.00001)

GPIO.output(self.TRIG, False)

pulse\_start=0;

pulse\_end=0;

while GPIO.input(self.ECHO)==0:

pulse\_start = time.time()

while GPIO.input(self.ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

time.sleep(0.4)

return distance;

sensor=ultraSound();

while True:

sensor.readData();

if(sensor.readData()<=THRESHOLD()):

GPIO.output(16,GPIO.HIGH)

print "WARNING THE LEVEL OF WATER HAS RAISED BEYOND THE THRESHOLD LEVEL"

else:

GPIO.output(16,GPIO.LOW)

**6.Hardware interfacing with Cayenne IoT platform**

Cayenne is an IoT project builder platform that allows you to create IoT applications and projects by connecting various sensors and devices to the Cayenne platform. Here's a step-by-step guide to configuring Cayenne for a flood management and early warning system using sensors and devices:

Step 1: Create a Cayenne Account

Go to the Cayenne website (https://cayenne.mydevices.com) and create a new account if you don't already have one.

Step 2: Add a New Device

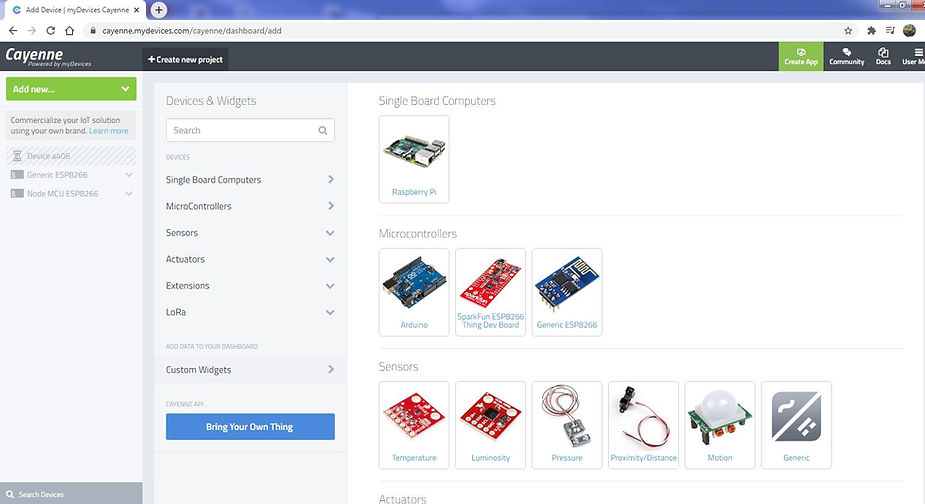
After logging in, click on the "Add New..." button and select "Device."

Add New Device here and select Generic ESP8266 for in this project.

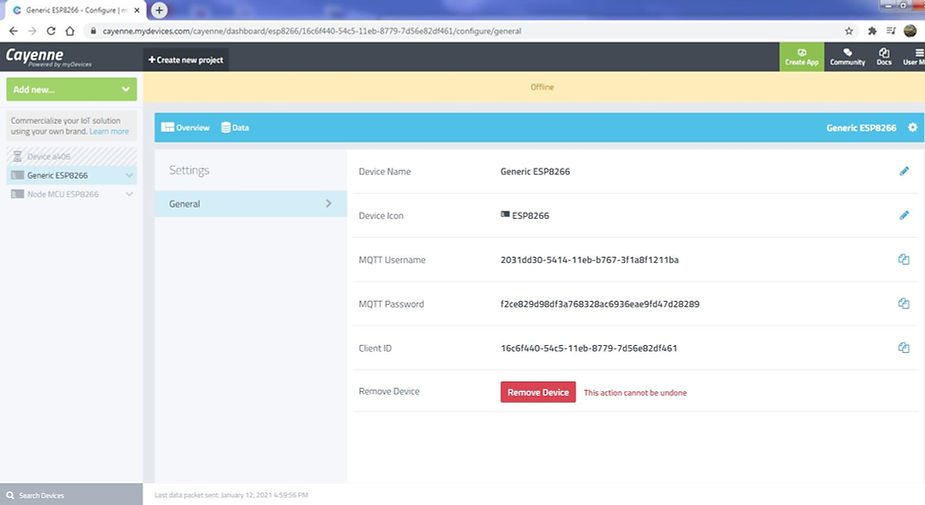
Step 3: Create a New Project

In your Cayenne dashboard, click on the "Create New Project" button.

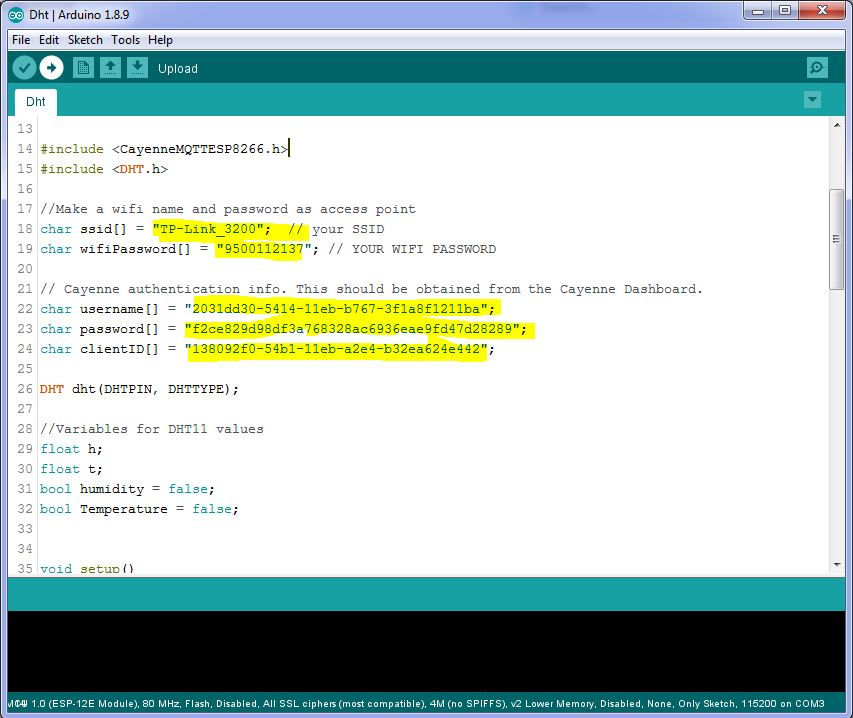
Give your project a name related to flood management and early warning.



Step 4: Configure device Generic ESP8266, MQTT username, password and client ID from Create App.



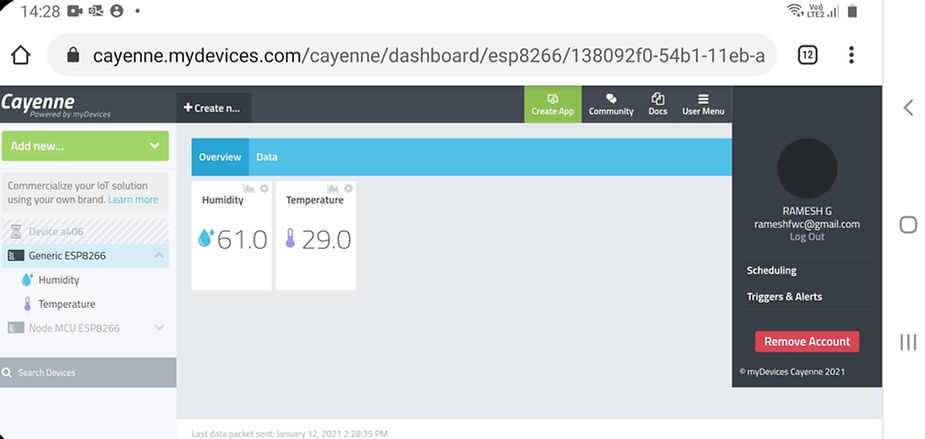
Step 5: Paste these respective details under username, password and client ID in Arduino source code, along with your Wi-Fi name and password.



After successfully compiling and uploading the code to NodeMCU, You will see ESP8266 connected to Wi-Fi. After the connection is established, the previous page is automatically updated on Cayenne. A new dashboard opens in the browser. Cayenne generates an ID and a device icon for your device.

Click on Custom Widgets and then value, and populate all fields . The channel number should be 1. (Make sure the channel number is same as in code.) Now, click on Add Widget.

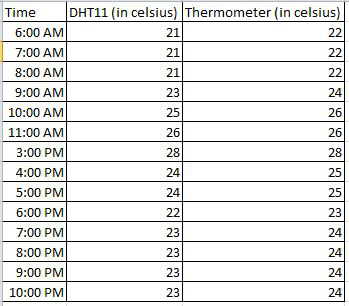
When a connection is made, sensor data gets uploaded to Cayenne. Temperature and Humidity data on Cayenne.

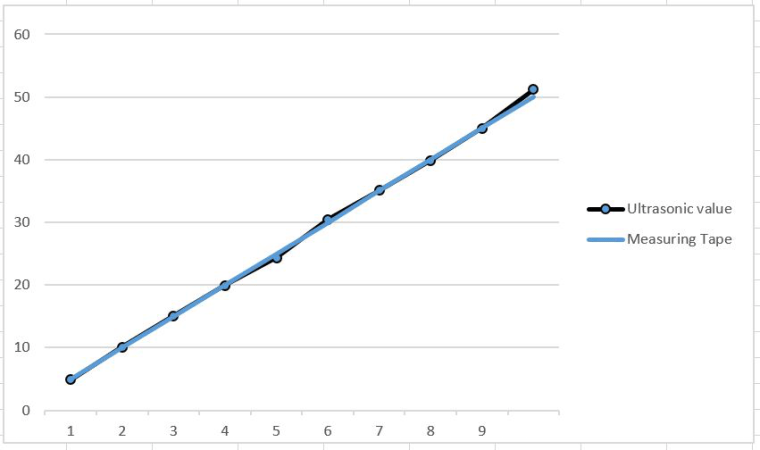


The Data is received from the Sensors and are displayed.

**7.RESULT:**

After all the complete connections of the system were made successfully along with the required software, the system was ready for testing. Individual models were tested at the beginning of the project. The system was tested for analyzing the various parameters such as temperature, humidity and level of water.

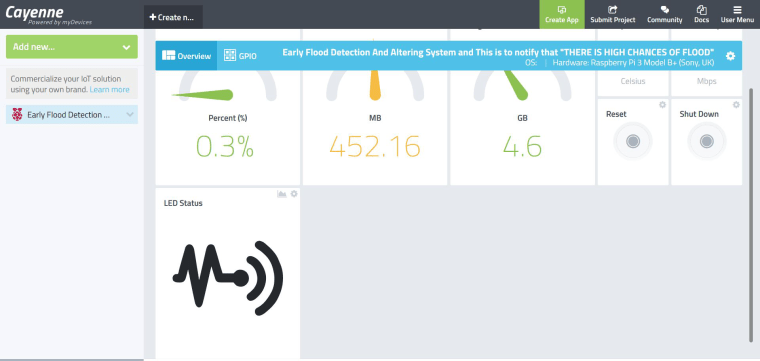




The value of ultrasonic sensor gives the distance between the sensor and the river which is normally calculated as the level of water. The value of distance was measured by varying the position of ultrasonic sensor and the data come out to be not much different than the distance measured on the measuring tape. This signifies that the ultrasonic sensor is feasible for the use in system. The above table shows the distance calculated from the measuring tape and ultrasonic sensor.

**8.Sample Output:**

The results from the Sensor is sent to Cayenne IoT platform and the data is displayed at real time.



When the water level increases then Alert is sent through SMS.

