

Team ID	PNT2022TMID27921
Project Name	Project - Real-Time River Water Quality Monitoring and Control System

USING WOWKI

SOURCE CODE:

```
#include <WiFi.h>//library for wifi
#include <PubSubClient.h>//library for MQTT
#include "DHT.h"// Library for dht11
#define DHTPIN 15
#define dhtpin 13    // what pin we're connected to
#define DHTTYPE DHT22 // define type of sensor DHT 11
#include <ESP32Servo.h>

DHT dht1 (DHTPIN, DHTTYPE);// creating the instance by passing pin and typr of dht
connected
DHT dht2 (dhtpin, DHTTYPE);
void callback(char* subscribetopic, byte* payload, unsigned int payloadLength);

//-----credentials of IBM Accounts-----

#define ORG "swz5ou"//IBM ORGANITION ID
#define DEVICE_TYPE "abcd"//Device type mentioned in ibm watson IOT Platform
#define DEVICE_ID "12"//Device ID mentioned in ibm watson IOT Platform
#define TOKEN "12345678" //Token
String data3;

float pH , t;
float tu,T;
int f;

//----- Customise the above values -----
char server[] = ORG ".messaging.internetofthings.ibmcloud.com";// Server Name
char publishTopic[] = "iot-2/evt/Data/fmt/json";// topic name and type of event perform
and format in which data to be send
char subscribetopic[] = "iot-2/cmd/command/fmt/String";// cmd REPRESENT command type AND
COMMAND IS TEST OF FORMAT STRING
char authMethod[] = "use-token-auth";// authentication method
char token[] = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;//client id
float dist,dur;
String data;

//-----
WiFiClient wifiClient; // creating the instance for wificlient
PubSubClient client(server, 1883, callback ,wifiClient); //calling the predefined client
id by passing parameter like server id,portand wificredential

void setup()// configureing the ESP32
```

```

{
    Serial.begin(115200);
    dht1.begin();
    dht2.begin();
    delay(10);
    Serial.println();
    wificonnect();
    mqttconnect();
}

void loop()// Recursive Function
{

    pH = dht1.readHumidity();
    t = dht1.readTemperature();
    tu = dht2.readHumidity();

    Serial.print("temp:");
    Serial.println(t);
    Serial.print("pH:");
    Serial.println(pH);
    Serial.print("turbid:");
    Serial.println(tu);

    PublishData(t,pH,tu);
    delay(1000);
    if (!client.loop()) {
        mqttconnect();
    }
}

/*.....retrieving to
Cloud.....*/

void PublishData(float temp, float pH, int turbid) {
    mqttconnect();//function call for connecting to ibm
    /*
        creating the String in in form JSon to update the data to ibm cloud
    */
    String payload = "{\"temp\":";
    payload += temp;
    payload += "," " \"pH\":";
    payload += pH;
    payload += "," " \"turbid\":";
    payload += turbid;
    payload += "}";

    Serial.print("Sending payload: ");
    Serial.println(payload);
}

```

```

    if (client.publish(publishTopic, (char*) payload.c_str())) {
        Serial.println("Publish ok");// if it successfully upload data on the cloud then it
will print publish ok in Serial monitor or else it will print publish failed
    } else {
        Serial.println("Publish failed");
    }
}

void PublishAlert() {
    mqttconnect();//function call for connecting to ibm
    /*
    creating the String in in form JSON to update the data to ibm cloud
    */
    String payload = "{\"alert\":\"";
    payload += 10000;
    payload += "\"}";

    Serial.print("Sending payload: ");
    Serial.println(payload);

    if (client.publish(publishTopic, (char*) payload.c_str())) {
        Serial.println("Publish ok");// if it successfully upload data on the cloud then it
will print publish ok in Serial monitor or else it will print publish failed
    } else {
        Serial.println("Publish failed");
    }
}

void mqttconnect() {
    if (!client.connected()) {
        Serial.print("Reconnecting client to ");
        Serial.println(server);
        while (!client.connect(clientId, authMethod, token)) {
            Serial.print(".");
            delay(500);
        }

        initManagedDevice();
        Serial.println();
    }
}

void wificonnect() //function definition for wificonnect
{
    Serial.println();
    Serial.print("Connecting to ");

    WiFi.begin("Wokwi-GUEST", "", 6);//passing the wifi credentials to establish the
connection
    while (WiFi.status() != WL_CONNECTED) {

```

```

    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void initManagedDevice() {
    if (client.subscribe(subscribetopic)) {
        Serial.println((subscribetopic));
        Serial.println("subscribe to cmd OK");
    } else {
        Serial.println("subscribe to cmd FAILED");
    }
}

void callback(char* subscribetopic, byte* payload, unsigned int payloadLength)
{
}

```

WOWKI SOURCE CODE-OUTPUT:

Simulation

DHT22

ESP32

DHT22

00:45.976

99%

Sending payload: {"temp":37.80,"pH":72.50,"turbid":51}

Publish ok

temp:37.80

pH:72.50

turbid:51.50

Sending payload: {"temp":37.80,"pH":72.50,"turbid":51}

Publish ok



IBM IOT WATSON:

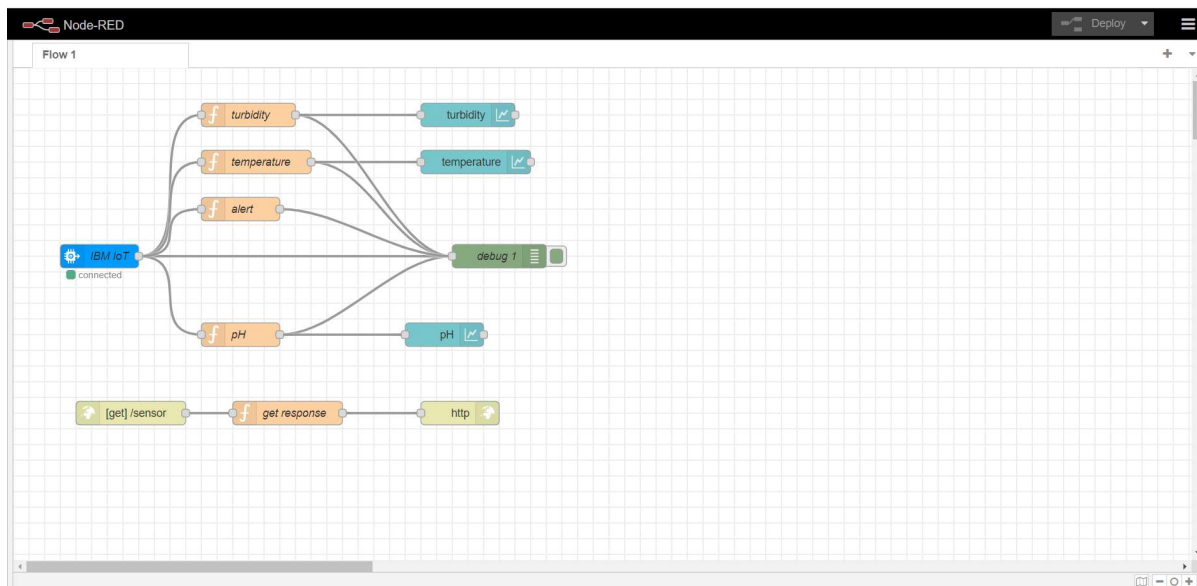
PUBLISHING DATA TO IBM IOT WATSON:

The screenshot displays the IBM Watson IoT Platform interface. At the top, the header shows the user's email (311519106066@smartinternz.com) and ID (swz5ou). The main navigation bar includes 'Browse', 'Action', 'Device Types', and 'Interfaces'. A sidebar on the left contains various icons for device management. The central panel shows a table of devices with columns: Device ID, Status, Device Type, Class ID, Date Added, and Descriptive Location. A device with ID 12 is selected, showing a status of 'Disconnected' and a device type of 'abcd'. Below the table, a 'Recent Events' tab is active, displaying a table of events. The events table has columns: Event, Value, Format, and Last Received. The events are all 'Data' events with a value of '{"temp":37.8,"pH":72.5,"turbid":51}' in 'json' format, received 'a few seconds ago'. A status bar at the bottom right indicates '0 Simulations running'.

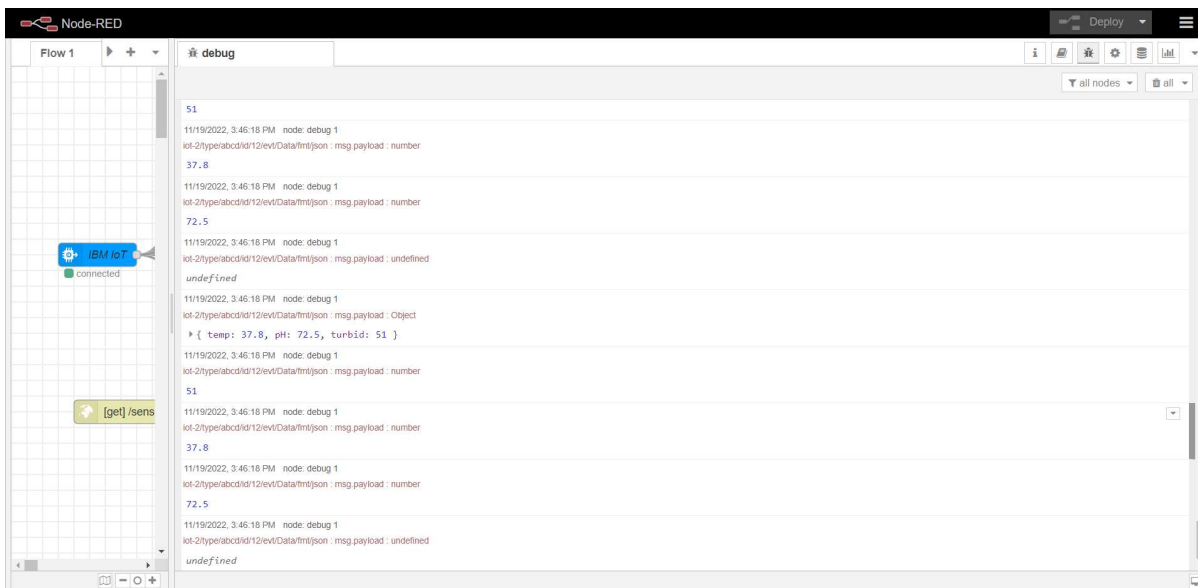
Event	Value	Format	Last Received
Data	{"temp":37.8,"pH":72.5,"turbid":51}	json	a few seconds ago
Data	{"temp":37.8,"pH":72.5,"turbid":51}	json	a few seconds ago
Data	{"temp":37.8,"pH":72.5,"turbid":51}	json	a few seconds ago
Data	{"temp":37.8,"pH":72.5,"turbid":51}	json	a few seconds ago
Data	{"temp":37.8,"pH":72.5,"turbid":51}	json	a few seconds ago

NODE-RED:

NODE-RED FLOW DIAGRAM:



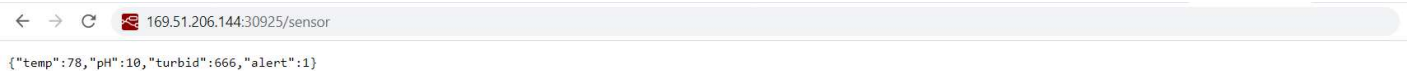
PUBLISHING DATA FROM IBM IOT WATSON TO NODE-RED:



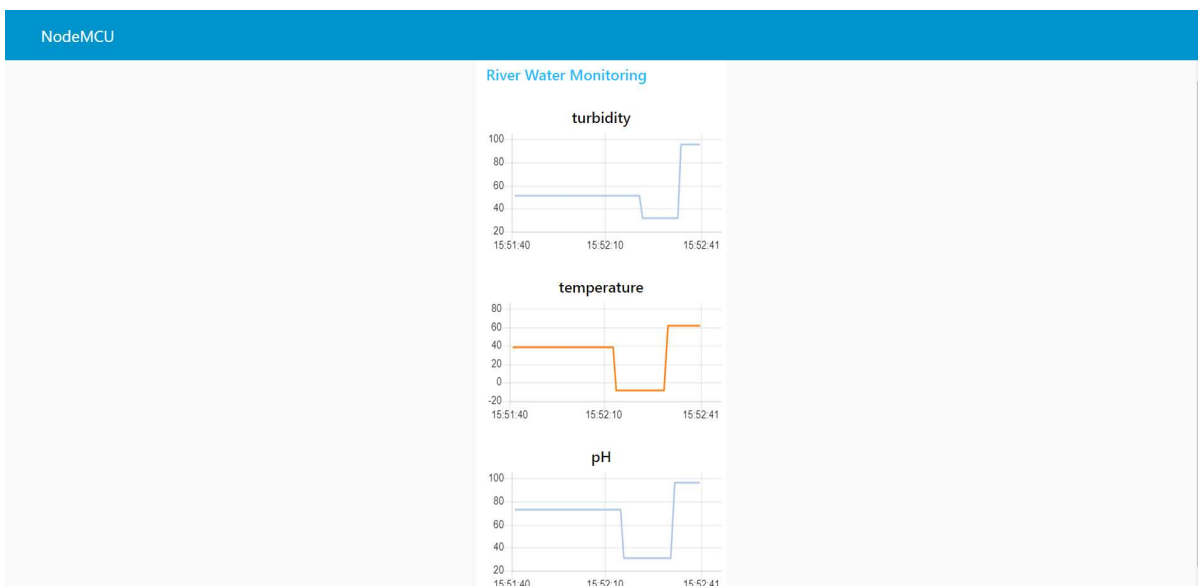
SOURCE CODE:

```
msg.payload = {"temp":global.get('t'),"pH":global.get('pH'),"turbid":global.get('tur'),"alert":global.get('a')}  
return msg;
```

HTTP REQUEST USING NODE RED:



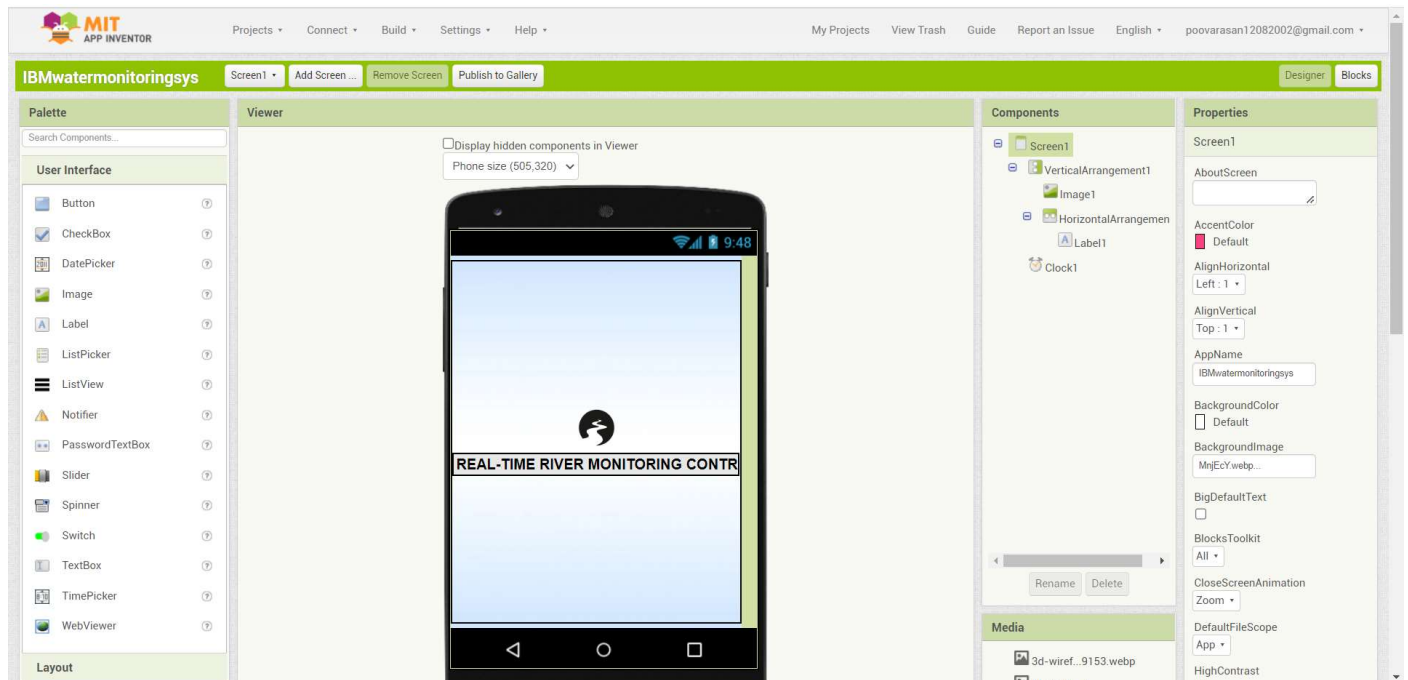
GENERATING THE OUTPUT FOR RECENT EVENTS:



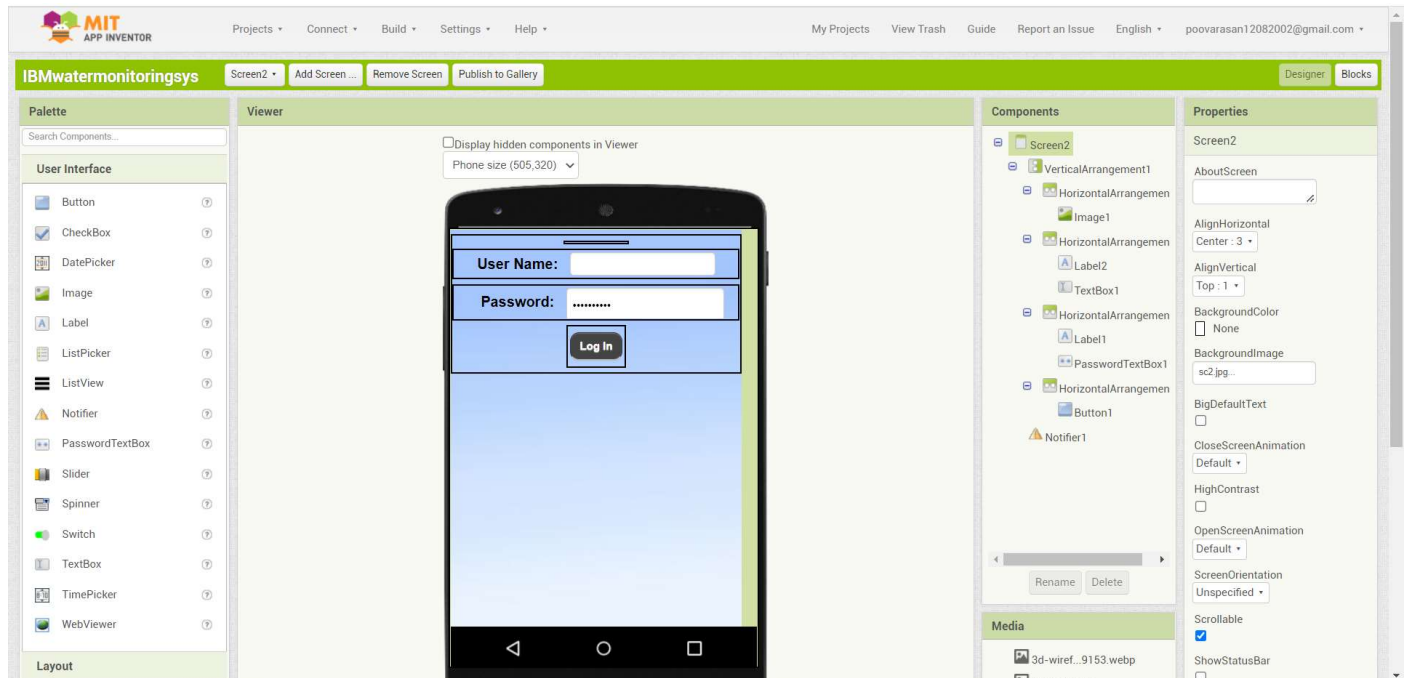
MIT APP INVENTOR:

FRONT END:

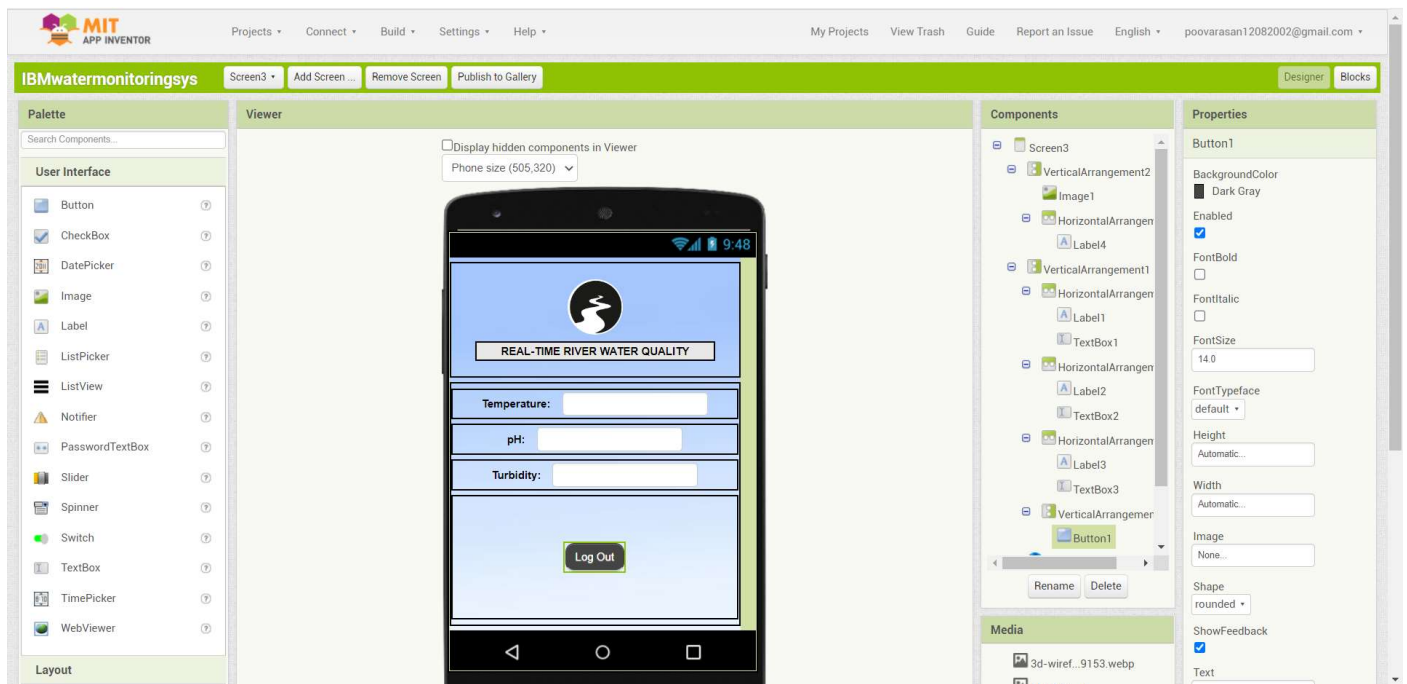
SCREEN-1:



SCREEN-2:

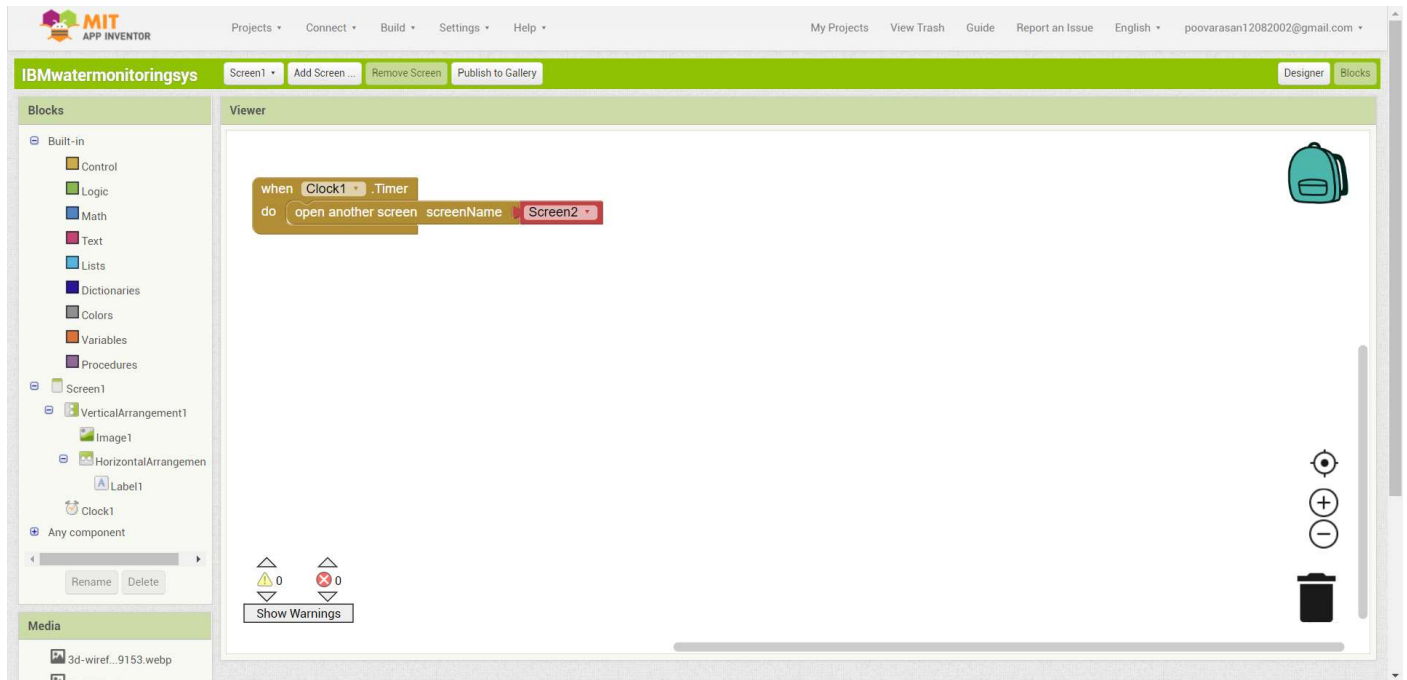


SCREEN-3:

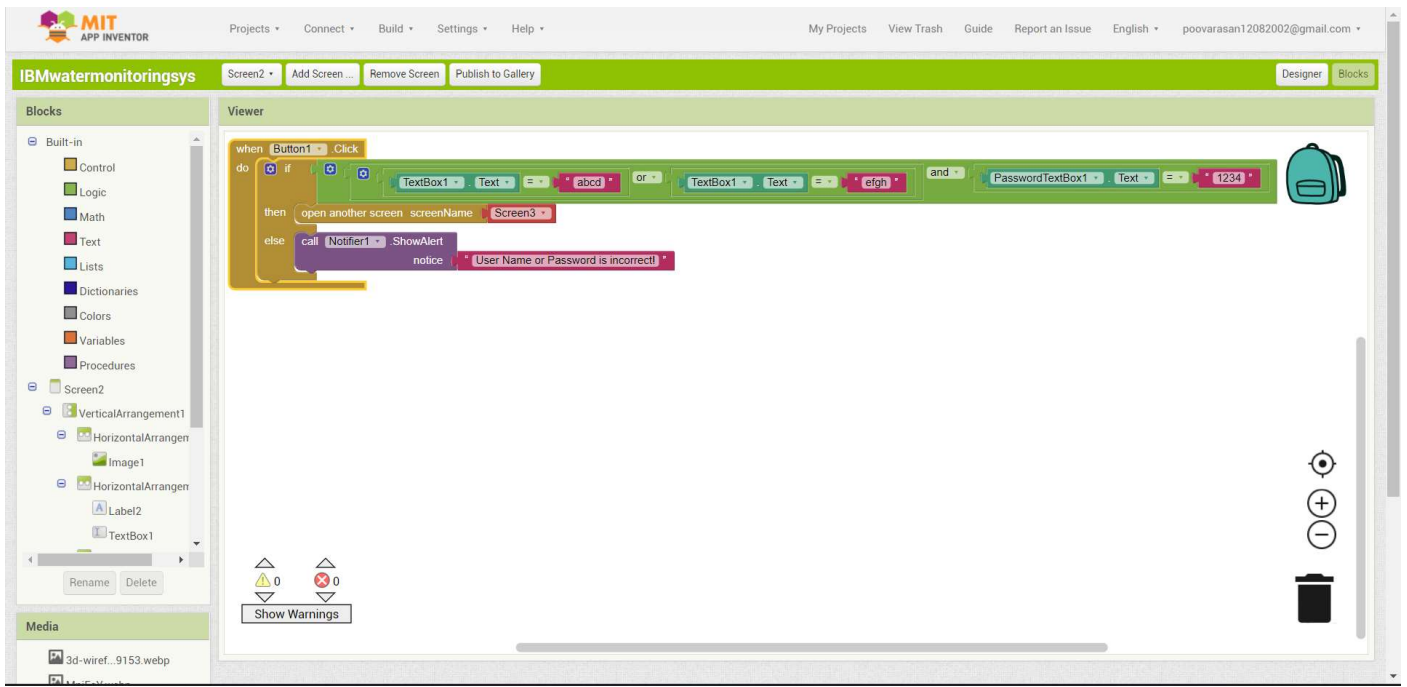


BACK END:

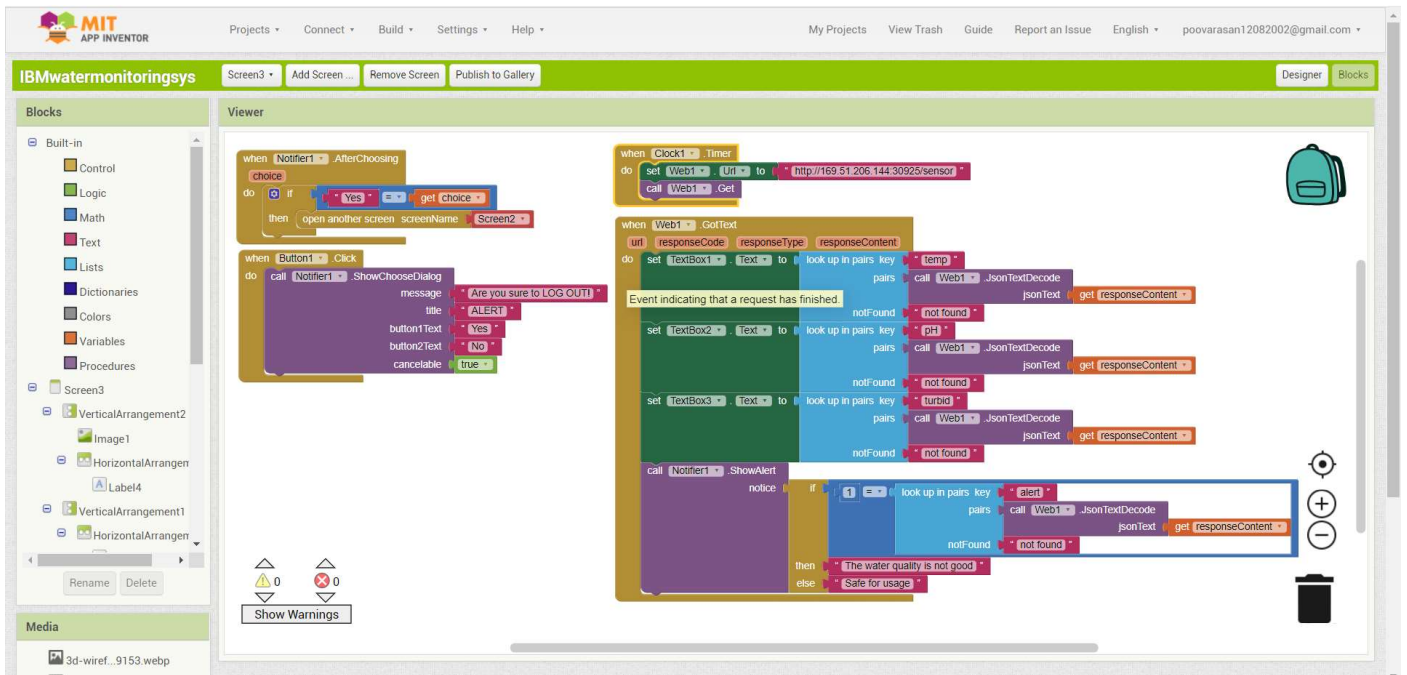
SCREEN-1:



SCREEN-2:



SCREEN-3:

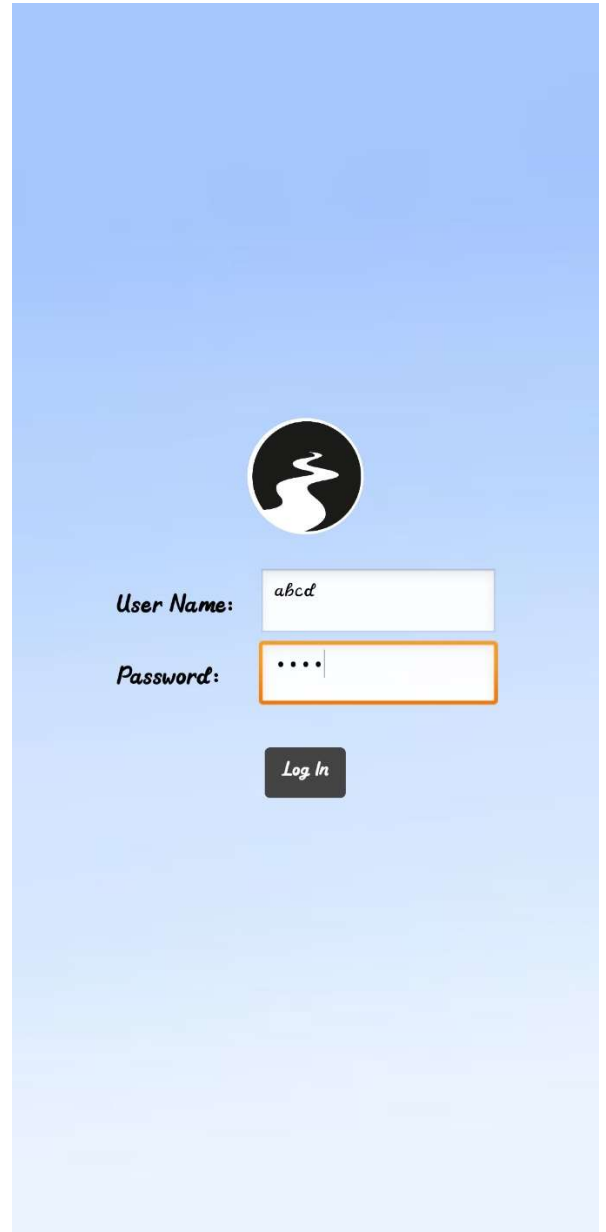


MIT APP INVERTOR OUTPUT-MOBILE PHONE:

SCREEN-1:



SCREEN-2(LOG IN PAGE)



SCREEN-3:



REAL-TIME RIVER WATER QUALITY MONITORING AND
CONTROL SYSTEM

Temperature:

61.2

pH:

96.5


Turbidity:

95

Safe for usage

Log Out

LOG OUT PAGE:



REAL-TIME RIVER WATER QUALITY MONITORING AND
CONTROL SYSTEM

Temperature:

22

pH:

8

ALERT

Are you sure to LOG OUT!

Yes

No

Cancel

Safe for usage

Log Out