Team Name: RGR

SMART FISH PONDS

ENABLED WITH

CLOUD APPLICATIONS

SIP PROJECT

Presented by:

Rasagjna

Gunashree

Ruchitha

MAIL ID 1: [mailtorasagjna201000@gmail.com](mailto:mailtorasagjna201000@gmail.com)

MAIL ID2: [ruchithagajawada@gmail.com](mailto:ruchithagajawada@gmail.com)

MAIL ID3: [gunashreevalmiki22@gmail.com](mailto:gunashreevalmiki22@gmail.com)

**ABSTRACT**

This project deals with working of “Smart Fish Ponds with Cloud Applications”. Fish have specific tolerant range of various parameters, thus fish farming of specific types of fish species requires certain conditions that have to be reached. Moreover, the people that work in the fish farming ponds have to be engaged in all day activities to maintain the living fish habitat. Therefore, monitoring and taking actions to maintain the habitat’s sustainable environment for certain fish species inside of fishing ponds over distributed machine to machine communication, which will shorten the time needed for some basic actions, is the main motivation for this project. In this project we present Internet of Things (IoT) system for monitoring fish ponds. The IoT system consists of various sensors that measure important factors of the water quality like temperature, turbidity, as well as small board computer that processes the data and sends notifications to the fish farming manager. The current system lacks the ability to process the data to the end-user via mobile platform. Due to remote distance of the fish farming ponds and their location dependence of clean fresh water, Modem allows user to communicate to the IoT system via Wi-Fi connection, all in one product. Later on, this module can be integrated with IoT for future work, we plan to expand not only the applicable services on different platforms, but also add more control modules and sensors to the existing IoT system for specific fish species. Through this project, we conclude that the need of smart fish ponds is increasing gradually and IoT has many more applications in every system.

**PROBLEM STATEMENT**

The aim of the project is to monitor water parameters like temperature and turbidity with the help of sensors which are linked to IBM cloud.Also, to reduce manual interference in maintaining the fish ponds.Through this project we can alert the specified person if the water parameters reach dangerous limits.Since the fishes need to be feed on time,if the person is not near the fish ponds , we can automate the feeding system to feed the fishes on time by keeping some motors.

**PROJECT WORKING PROCESS**

Firstly, a proper code is prepared to know the temperature and turbidity values so that when they reach the prescribed limits, a message notification should be sent to the mobile. For this the temperature sensor and turbidity sensors are linked with the IBM cloud through node-red. The notifications are built with the help of MSG91. We first take a temperature sensor and give the required connections using a node -mcu and a bread board.

The connections for temperature sensor are :

1.D2 pin of node-mcu is connected to yellow wire of temperature sensor.

2. Gnd of node-mcu is connected to black wire of temperature sensor.

3.3V3 of node-mcu is connected to red wire of temperature sensor.

We build a mobile application using MIT app inventor 2 to control the servo rotation. Whenever we switch on the servo, it should rotate and this command should be visible in the serial monitor. The connections are as follows:

1. D3 pin of node-mcu is connected to orange wire of servo motor.
2. Gnd pin of node-mcu is connected to brown wire of servo motor.
3. 3V3 pin of node-mcu is connected to red wire of servo motor.

The same process of temperature sensor is applied to the turbidity sensor. Whenever the values reach the prescribed limits a notification should be sent to the user. The connections are as follows:

1. Black wire of turbidity sensor is connected to Gnd of turbidity board.
2. Red wire of turbidity sensor is connected to VCC of turbidity board.
3. Brown wire of turbidity sensor is connected to A0 of turbidity board.
4. VCC and Gnd of turbidity board are connected to 3V3 and Gnd of node-mcu.

**ABOUT COMPONENTS USED**

1. **Temperature sensor:**

In this project, by Interfacing DS18B20 Temperature Sensor with ESP8266, we are extracting the temperature information from the sensor.

After making all the connections, configure the ESP8266 Module in Programming Mode (by connecting GPIO0 to GND and Resetting the Module). Now, you can upload the code to the ESP8266 Module using Arduino IDE.

NOTE: Select correct board and PORT in the Arduino IDE. Generic ESP8266 Module should be the Board.

Once the code is uploaded, configure the ESP8266 is normal mode by sliding GPIO0 from GND and resetting the module. Open the Serial Monitor in the Arduino IDE and select the baud rate as 115200. Also, do not forget to choose “Both NL & CR” option.

In the serial monitor, we can see the ESP8266 Module getting connected to the Internet through the Wi-Fi Network.

After confirming the Wi-Fi Connection, the ESP Module will start reading the Temperature information from the DS18B20 Sensor. This temperature information is displayed on the serial monitor according to the delay given.

As we also need to send message notifications to the user, the temperature values will be

uploaded to IBM cloud.



**2. Turbidity sensor:**

The gravity arduino turbidity sensor detects water quality by measuring the levels of turbidity. It uses light to detect suspended particles in water by measuring the light transmittance and scattering rate, which changes with the amount of total suspended solids (TSS) in water. As the TSS increases, the liquid turbidity level increases. Turbidity sensors are used to measure water quality in rivers and streams, wastewater and effluent measurements, control instrumentation for settling ponds, sediment transport research and laboratory measurements. This liquid sensor provides analog and digital signal output modes. The threshold is adjustable when in digital signal mode. We can select the mode according to our MCU.



**3.Servo motor:**

Nodemcu esp8266 works on 3.3 volts and servo motor requires 5 volts for its operations. So we need different power supplies for both the peripherals. Nodemcu is powered by the usb port of the computer and servo is powered with an another 5 volt adopter. GPIO-2 or D4 pin of nodemcu esp8266 12e is outputting pwm signal for servo motor rotation. Pwm signal output by the nodemcu is also in 3.3v TTL form. It must also be raised to 5v TTL before feeding the servo motor.

Project code is written in arduino ide. Luckily arduino has plenty of libraries for each function it offers. For interfacing servo with arduino or boards supported by arduino ide their is a pre defined library named "Servo.h". I included this library in my code and called its functions in code which are pretty easy to use. First the 'ESP8366WiFi.h' library us imported in code. This library contains functions used to initialize the nodemcu server and start the nodemcu esp8266 WiFi. Then 'Servo.h' library us imported.

Just make the circuit and upload the above code in your nodemcu WiFi module. Before uplaoding make sure the correct board is selected in the boards manage. After upload open the serial monitor of nodemcu. As soon as you open the serial monitor you will see nodemcu requesting your router for an IP allotment. After allotment of IP nodemcu will start its server and prints the server address on the serial monitor. This address is actually the address of the web page which has servo motor controls. You have to enter this address in your browser. If you see nothing in the serial monitor or just arbitrary and in complete characters look for the baud rate in serial monitor window. It must be set to 115200. Check for all possible lose connections.

Nodemcu esp8266 works on 3.3 volts and servo motor requires 5 volts for its operations. So we need different power supplies for both the peripherals. Nodemcu is powered by the usb port of the computer and servo is powered with an another 5 volt adopter. GPIO-2 or D4 pin of nodemcu esp8266 12e is outputting pwm signal for servo motor rotation. Pwm signal output by the nodemcu is also in 3.3v TTL form. It must also be raised to 5v TTL before feeding the servo motor.

Project code is written in arduino ide. Luckily arduino has plenty of libraries for each function it offers. For interfacing servo with arduino or boards supported by arduino ide their is a pre defined library named "Servo.h". I included this library in my code and called its functions in code which are pretty easy to use. First the 'ESP8366WiFi.h' library us imported in code. This library contains functions used to initialize the nodemcu server and start the nodemcu esp8266 WiFi. Then 'Servo.h' library us imported.

Just make the circuit and upload the above code in your nodemcu WiFi module. Before uplaoding make sure the correct board is selected in the boards manage. After upload open the serial monitor of nodemcu. As soon as you open the serial monitor you will see nodemcu requesting your router for an IP allotment. After allotment of IP nodemcu will start its server and prints the server address on the serial monitor. This address is actually the address of the web page which has servo motor controls. You have to enter this address in your browser. If you see nothing in the serial monitor or just arbitrary and in complete characters look for the baud rate in serial monitor window. It must be set to 115200. Check for all possible lose connections.



**4.Node mcu:**

NodeMCU is an open source IoT platform.It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266.The ESP-12E is a board created by AI-THINKER.The development board equips the ESP-12E module containing ESP8266 chip having Tensilica Xtensa® 32-bit LX106 RISC microprocessor which operates at 80 to 160 MHz adjustable clock frequency and supports RTOS.As the operating voltage range of ESP8266 is 3V to 3.6V, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as 80mA during RF transmissions. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

peripherals and i/o:

The ESP8266 NodeMCU has total 17 GPIO pins broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

ADC channel – A 10-bit ADC channel.

UART interface – UART interface is used to load code serially.

PWM outputs – PWM pins for dimming LEDs or controlling motors.

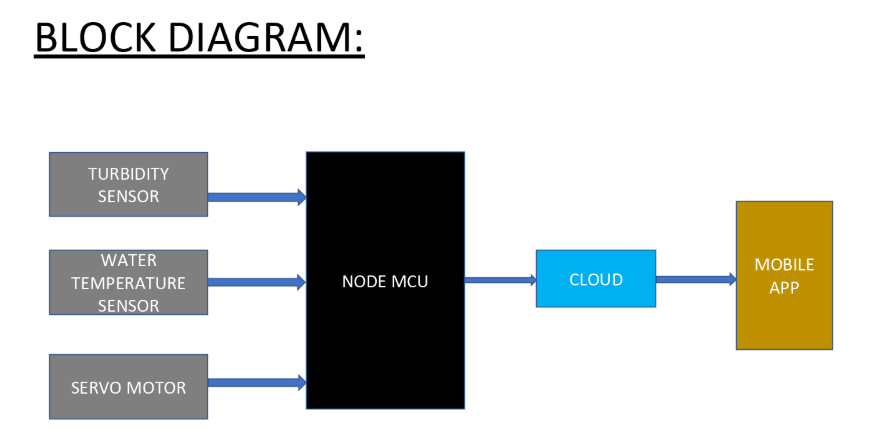
SPI, I2C & I2S interface – SPI and I2C interface to hook up all sorts of sensors and peripherals.

I2S interface – I2S interface if you want to add sound to your project.

on board switches and LED indicator:

The ESP8266 NodeMCU features two buttons. One marked as RST located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other FLASH button on the bottom left corner is the download button used while upgrading firmware.The board also has a LED indicator which is user programmable and is connected to the D0 pin of the board.

****

****

**CODE:**

#include <Servo.h>  
#include <ESP8266WiFi.h>  
#include <PubSubClient.h>  
#include <OneWire.h>  
#include <DallasTemperature.h>  
#include <SoftwareSerial.h>  
SoftwareSerial mySerial(D9, D10);   
// Data wire is plugged into port 2 on the Arduino  
#define ONE\_WIRE\_BUS D2  
  
// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)  
OneWire oneWire(ONE\_WIRE\_BUS);  
  
// Pass our oneWire reference to Dallas Temperature.   
DallasTemperature sensors(&oneWire);  
Servo myservo;  
  
  
// CHANGE TO YOUR WIFI CREDENTIALS  
const char\* ssid = "LG-X230I";  
const char\* password = "rasagjna201000";  
  
  
// CHANGE TO YOUR DEVICE CREDENTIALS AS PER IN IBM BLUMIX  
#define ORG "5s2w9y"  
#define DEVICE\_TYPE "Rasagjna20"  
#define DEVICE\_ID "100"  
#define TOKEN "9030204022"  //  Authentication Token OF THE DEVICE  
float temperature;  
String data3;  
String data="";  
String command;  
 float tempC;  
 float sensorValue;  
const char\* host = "[api.msg91.com](http://api.msg91.com/)";  
//-------- Customise the above values --------  
const char publishTopic[] = "iot-2/evt/Data/fmt/json";  
char server[] = ORG ".[messaging.internetofthings.ibmcloud.com](http://messaging.internetofthings.ibmcloud.com/)";  
char topic[] = "iot-2/cmd/home/fmt/String";// cmd  REPRESENT command type AND COMMAND IS TEST OF FORMAT STRING  
char authMethod[] = "use-token-auth";  
char token[] = TOKEN;  
char clientId[] = "d:" ORG ":" DEVICE\_TYPE ":" DEVICE\_ID;  
void callback(char\* topic, byte\* payload, unsigned int payloadLength);  
WiFiClient wifiClient;  
PubSubClient client(server, 1883, callback, wifiClient);  
  
int publishInterval = 5000; // 30 seconds  
long lastPublishMillis;  
void publishData();  
void wifiConnect() ;  
void setup() {  
  myservo.attach(D3);  
  Serial.begin(115200);  
  wifiConnect();  
  mqttConnect();  
  Serial.println("Dallas Temperature IC Control Library Demo");  
  
  // Start up the library  
  sensors.begin();  
}  
  
void loop() {  
  sensorValue = analogRead(A0);// read the input on analog pin 0:  
  Serial.println(sensorValue); // print out the value you read:  
  delay(500);  
  Serial.print("Requesting temperatures...");  
  sensors.requestTemperatures(); // Send the command to get temperatures  
  Serial.println("DONE");  
    
  tempC = sensors.getTempCByIndex(0);  
  
  // Check if reading was successful  
  if(tempC != DEVICE\_DISCONNECTED\_C)   
  {  
    Serial.print("Temperature for the device 1 (index 0) is: ");  
    Serial.println(tempC);  
  }   
  else  
  {  
    Serial.println("Error: Could not read temperature data");  
  }  
  
 if (millis() - lastPublishMillis > publishInterval)  
  {  
    publishData();  
    lastPublishMillis = millis();  
  }  
    
  if (!client.loop()) {  
    mqttConnect();  
  }  
  if (sensorValue>10)  
  {  
  msg();  
  Serial.println("turbidity value is exceeded");  
  }  
  else if(tempC<24 || tempC>27)  
  
  {  
   msg();  
    Serial.println("temperature value is inappropriate");  
  }  
  else{  
  }  
  delay(1000);  
  }  
  
void wifiConnect() {  
  Serial.print("Connecting to "); Serial.print(ssid);  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL\_CONNECTED) {  
    delay(500);  
    Serial.print(".");  
  }  
  Serial.print("nWiFi connected, IP address: "); Serial.println(WiFi.localIP());  
}  
  
void mqttConnect() {  
  if (!client.connected()) {  
    Serial.print("Reconnecting MQTT client to "); Serial.println(server);  
    while (!client.connect(clientId, authMethod, token)) {  
      Serial.print(".");  
      delay(500);  
    }  
    initManagedDevice();  
    Serial.println();  
  }  
}  
  
void initManagedDevice() {  
  if (client.subscribe(topic)) {  
   // Serial.println(client.subscribe(topic));  
    Serial.println("subscribe to cmd OK");  
  } else {  
    Serial.println("subscribe to cmd FAILED");  
  }  
}  
  
void callback(char\* topic, byte\* payload, unsigned int payloadLength) {  
    
  Serial.print("callback invoked for topic: ");  
  Serial.println(topic);  
  
  
  Serial.print("callback invoked for topic: "); Serial.println(topic);  
  
  for (int i = 0; i < payloadLength; i++) {  
    //Serial.println((char)payload[i]);  
    command += (char)payload[i];  
  }  
Serial.println(command);  
if(command == "servoon"){  
   myservo.write(180);  
  Serial.println("servo is Switched ON");  
}  
else if(command == "servooff"){  
  myservo.write(0);  
  Serial.println("servo is Switched OFF");  
}  
command ="";  
}  
  
  
void publishData()   
{  
    
   if (isnan(tempC)|| isnan(sensorValue)) {  
    Serial.println("Failed to read from DHT sensor!");  
    return;  
  }  
    
  String payload = "{\"d\":{\"temperature\":";  
  payload += tempC;  
   payload += ",""\"turbidity\":";  
  payload +=  sensorValue;  
    
  payload += "}}";  
  
  
  Serial.print("\n");  
  Serial.print("Sending payload: "); Serial.println(payload);  
  
  if (client.publish(publishTopic, (char\*) payload.c\_str())) {  
    Serial.println("Publish OK");  
  } else {  
    Serial.println("Publish FAILED");  
  }  
}  
void msg(){  
   Serial.print("connecting to ");  
  Serial.println(host);  
    
  // Use WiFiClient class to create TCP connections  
  WiFiClient client1;  
  const int httpPort = 80;  
  if (!client1.connect(host, httpPort)) {  
    Serial.println("connection failed");  
    return;  
  }  
    
  // We now create a URI for the request  
  String url = "/api/sendhttp.php?mobiles=9030204022&authkey=280139ABGUcvoPO5cfb6053&route=4&sender=TESTIN&message=limit Exceeded&country=91";  
  Serial.print("Requesting URL: ");  
  Serial.println(url);  
    
  // This will send the request to the server  
  client1.print(String("GET ") + url + " HTTP/1.1\r\n" +  
               "Host: " + host + "\r\n" +   
               "Connection: close\r\n\r\n");  
  delay(10);  
    
  // Read all the lines of the reply from server and print them to Serial  
  while(client1.available()){  
    String line = client1.readStringUntil('\r');  
    Serial.print(line);  
  }  
    
  Serial.println();  
  Serial.println("closing connection");  
}

CONCLUSIONS:

1. A message notification is sent when the temperature and turbidity values are exceeded to the mobile phone.
2. In order to supply food to fishes , a servo motor is made rotated through a mobilae application.