Detector\Detector.ino

```
/*
This is an implementation of a home automation system that uses an ESP8266
 microcontroller to monitor and control sensors and actuators.
 The system uses the painlessMesh library to create a mesh network of nodes
 that can communicate with each other.
 The mesh network is used to send and receive data from various sensors and
 actuators, such as a soil moisture sensor, a temperature and humidity sensor,
 and a buzzer.

It starts by including the necessary libraries, such as painlessMesh, PageBuilder,
 PageStream, DHT_U, DHT, and ArduinoJson. It then defines some constants,
 such as the mesh prefix, password, and port, as well as the pins for the
 various sensors and actuators. T
 he code also initializes the DHT11 component and allocates memory for the
 JSON document.

The setup function initializes the serial communication for debugging,
initializes the mesh network, sets up the callbacks for receiving messages,
new connections, changed connections, and node time adjustments, and adds a
task to send messages periodically. The setup function also connects to the
WiFi network, sets the LED pin as an output, and starts the web server.
Finally, the setup function initializes the DHT component.

The loop function updates the mesh network, handles client requests, reads the
water level value, turns on or off the pump based on the water level value,
reads the detector switch value, sets the LED brightness based on the water
level value, sets the buzzer frequency based on the LED brightness, and sounds
the buzzer if the pump is on and no one is close. The loop function also reads
the temperature and humidity values, creates an HTML page with the current
values, and sends the HTML page to the client.

The code also includes some helper functions, such as detectorOn, readTempHum,
jsonDetectorSensor, get_index, get_json, and shouldTurnOnPump.
The detectorOn function reads the switch pin value and returns true or false
based on the status. The readTempHum function reads the temperature and
humidity values and prints them to the serial monitor. The jsonDetectorSensor
function adds JSON request data to the document. The get_index function creates
an HTML page with the current values and sends it to the client. The get_json
function creates JSON data with the current values and sends it to the client.
The shouldTurnOnPump function checks the water level value and returns true or
false based on the threshold.


*/


#include "painlessMesh.h"

#define MESH_PREFIX "homeIOT"
#define MESH_PASSWORD "phyComIOT"
#define MESH_PORT 5555
#include <PageBuilder.h>
```

```
#include <PageStream.h>
#include <DHT_U.h>
#include <DHT.h>
#include <ArduinoJson.h>

// #include <TinyDHT.h>
// #include "DHT.h"

// Define the pins for the soil moisture sensor and LED
const int waterLevelPin = A0; // Analog pin for soil moisture sensor
const int ledPin = D2;        // Digital pin for LED
const int buzzerPin = D3;     // Digital pin for the buzzer
const int switchPin = D1;
const int tempHumPin = D4;

// Initialise the DHT11 component
DHT dht(tempHumPin, DHT11);

// Allocate the JSON document
// Allows to allocated memory to the document dinamically.
DynamicJsonDocument doc(1024);

// Set the PORT for the web server
ESP8266WebServer server(80);

// The WiFi details
// const char *ssid = "Oluseed";
// const char *password = "mic12345";

int switch_value;
int waterLevelValue = 1024; // Highest value for the sensor
int ledBrightness = 0;
// Initialise variables to store the temperature and humidity values
int temperature = 0;
int humidity = 0;
int buzzerFrequency = 0;

int noWaterLevel = 900; // Minimum water level in the pot
String pump = "OFF";
String someoneClose = "NO";

Scheduler userScheduler; // to control your personal task
painlessMesh mesh;

// User stub
void sendMessage(); // Prototype so PlatformIO doesn't complain

Task taskSendMessage(TASK_SECOND * 1, TASK_FOREVER, &sendMessage);

void sendMessage()
{
  jsonDetectorSensor();

  // Make JSON data ready for the http request
  String jsonStr;
  serializeJson(doc, jsonStr); // The function is from the ArduinoJson library no need for pretty
  mesh.sendBroadcast(jsonStr);
  Serial.println("Detector sending message: " + jsonStr);
  taskSendMessage.setInterval(random(TASK_SECOND * 1, TASK_SECOND * 5));
}

void handleJsonMessage(const char *json)
{
  StaticJsonDocument<1024> doc;
  DeserializationError error = deserializeJson(doc, json);
  if (error)
  {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return;
  }
  int nodeId = doc["nodeId"];
  String message = doc["message"];
  Serial.printf("Received message from node %d: %s\n", nodeId, message.c_str());
  someoneClose = doc["someoneClose"].as<String>();

  // Display other data
}

// Needed for painless library
void receivedCallback(uint32_t from, String &msg)
{
  Serial.printf("Received from %u msg=%s\n", from, msg.c_str());

  handleJsonMessage(msg.c_str());
}

void newConnectionCallback(uint32_t nodeId)
{
  Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}

void changedConnectionCallback()
{
  Serial.printf("Changed connections\n");
}

void nodeTimeAdjustedCallback(int32_t offset)
{
  Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(), offset);
}
void setup()
{
  // Initialize Serial communication for debugging
  Serial.begin(115200);

  // mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC | COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
  mesh.setDebugMsgTypes(ERROR | STARTUP); // set before init() so that you can see startup messages

  mesh.init(MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT);
  mesh.onReceive(&receivedCallback);
  mesh.onNewConnection(&newConnectionCallback);
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

  userScheduler.addTask(taskSendMessage);
  taskSendMessage.enable();

  // Connect to the WiFi network
  // WiFi.begin(ssid, password);
  // Set the LED pin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  pinMode(switchPin, INPUT);

  // // Wait for connection
  // while (WiFi.status() != WL_CONNECTED)
  // {
  //   delay(500);
  //   Serial.println("Waiting to connect... to: " + String(ssid));
  // }

  // // Print the board IP address
  // Serial.print("IP address: ");
  // Serial.println(WiFi.localIP());

  server.on("/", get_index);    // Get the index page on root route
  server.on("/json", get_json); // Get the json data on the '/json' route

  server.begin(); // Start the server
  Serial.println("Server listening");

  // Start the dht component reading
  dht.begin();
}

// The loop function runs continuously
void loop()
{

  // Update the mesh network
  mesh.update();

  // Handle incoming client requests
  server.handleClient();

  // Read the water level sensor value
  waterLevelValue = analogRead(waterLevelPin);

  // Check if the pump should be turned on based on the water level
  if (shouldTurnOnPump(waterLevelValue, noWaterLevel))
  {
    Serial.print("Should turn on pump");

    // Turn the relay ON (close the contacts)
    // delay(5000); // Wait for 5 second
    // digitalWrite(relayPin, LOW);

    // Set the pump status to ON
    pump = "ON";

    // delay(1000); // Wait for 1 second

    // // Turn the relay OFF (open the contacts)
    // digitalWrite(relayPin, LOW);
    // delay(1000); // Wait for 1 second
  }
  else
  {
    Serial.print("Should turn off pump");

    // Turn the relay OFF (open the contacts)
    // digitalWrite(relayPin, HIGH);

    // Set the pump status to OFF
    pump = "OFF";

    // delay(1000); // Wait for 1 second
  }

  // Check if the motion detector is on
  if (detectorOn())
  {
    Serial.print("Detector is on ");

    // Set the LED brightness to maximum
    ledBrightness = 255;
  }
  else
  {
    Serial.print("Detector is off");

    // Set the LED brightness to minimum
    ledBrightness = 0;
  }

  // Map the water level sensor value to the LED brightness
  ledBrightness = map(waterLevelValue, 500, 1023, 255, 0);

  // Set the LED brightness
  analogWrite(ledPin, ledBrightness);

  // buzzerFrequency = map(ledBrightness, 0, 255, 2000, 500); // Adjust frequency range as needed

  // Serial.print("Buzzer Frequency: ");
  // Serial.print(buzzerFrequency);

  // // Set the buzzer frequency and duration
  // tone(buzzerPin, buzzerFrequency);
  // delay(50); // Adjust delay for buzzer tone duration

  // If the pump is on and no one is close, sound the buzzer and print a message
  Serial.println("Pump: ");
  Serial.print(pump);
  Serial.println("Someone Close: ");
  Serial.println(someoneClose);
  if (pump == "ON" && someoneClose == "NO")
  {
    tone(buzzerPin, 2000);
    Serial.println("Send Message");
  }
  else
  {
    noTone(buzzerPin);
  }

  // Print the water level sensor value and LED brightness to the Serial Monitor
  Serial.print("Soil Moisture: ");
  Serial.print(waterLevelValue);
  Serial.print(" | LED Brightness: ");
  Serial.println(ledBrightness);

  // Add a delay to avoid rapid updates
  delay(1000); // 1 second delay

  // Read the temperature and humidity
  readTempHum();
}

bool detectorOn()
{

  // read the switch pin value
  switch_value = digitalRead(switchPin);

  // Log switch value
  Serial.print("Switch value: ");
  Serial.println(switch_value);

  // check the status and return either true or false
  if (switch_value == 0)
  {
    return false;
  }
  return true;
}

// Read the temperature and humidity values
void readTempHum()
{

  temperature = dht.readTemperature();
  humidity = dht.readHumidity();
  Serial.println(temperature);
  Serial.println(humidity);

  Serial.println("Temperature: " + String(temperature) + " C");
  Serial.println("Humidity: " + String(humidity) + " %");
}

void get_index()
{
  // Read the temperature and humidity values
  readTempHum();

  // Read the water level value
  waterLevelValue = analogRead(waterLevelPin);

  // Create the HTML page with the current values
  String html = "<html><head><title>Dashboard</title></head><body>";
  html += "<h1>Detector</h1>";
  html += "<p>Water Level: " + String(waterLevelValue) + "</p>";
  html += "<p>Temperature: " + String(temperature) + " C</p>";
  html += "<p>Humidity: " + String(humidity) + " %</p>";
  // Include buzzer frequency
  html += "<p>Buzzer Frequency: " + String(buzzerFrequency) + " Hz</p>";
  html += "</body></html>";

  // Send the HTML page to the client
  server.send(200, "text/html", html);
}
// Check water level
// if water level is low, turn on the pump
// if water level is high, turn off the pump
void jsonDetectorSensor()
{

  // Add JSON request data
  doc["Content-Type"] = "application/json";
  doc["Status"] = 200;
  doc["nodeId"] = mesh.getNodeId();
  doc["message"] = "Message from node Detector";

  // Set flags
  doc["pump"] = pump;

  // Add water level sensor JSON object data
  JsonObject waterLevel = doc.createNestedObject("WaterLevel");
  waterLevel["description"] = "Water Level";
  waterLevel["value"] = waterLevelValue;

  // Add temperature and humidity sensor JSON object data
  JsonObject tempHumSensor = doc.createNestedObject("TempHum");
  tempHumSensor["description"] = "Temperature and Humidity Sensor";
  tempHumSensor["temperature"] = temperature;
  tempHumSensor["humidity"] = humidity;

  // Add buzzer frequency JSON object data
  JsonObject buzzer = doc.createNestedObject("Buzzer");
  buzzer["description"] = "Sound";
  buzzer["frequency"] = buzzerFrequency;
}
void get_json()
```

```
{
  // Create JSON data
  jsonDetectorSensor(); // This adds some data to doc
  // Make JSON data ready for the http request
  String jsonStr;
  serializeJsonPretty(doc, jsonStr); // The function is from the ArduinoJson library
  // Send the JSON data
  server.send(200, "application/json", jsonStr);
}

bool shouldTurnOnPump(int value, int threshold)
{

  if (value > threshold)
  {
    // Turn the relay ON (close the contacts)
    // digitalWrite(relayPin, HIGH);
    return true;
  }
  else
  {
    // Turn the relay OFF (open the contacts)
    // digitalWrite(relayPin, LOW);
    return false;
  }
}

```