Remedy\Remedy.ino

```
/*
************************************************************
Thiis is the remedy node code that also uses the painlessMesh library to send
and receive messages between dectector and itself on a mesh network. The sketch also includes
code to control a relay, an ultrasonic sensor, and a servo motor. The relay is
used to turn a water pump on and off, the ultrasonic sensor is used to detect
if someone is close, and the servo motor is used to move a sensor.

The sketch starts by including the painlessMesh library and defining some
constants for the mesh network, such as the prefix, password, and port.
It also includes libraries for the servo motor, ultrasonic sensor,
and ArduinoJson, which is used to create and parse JSON data.

The setup function initializes the serial communication and the mesh network.
It also sets the pin modes for the relay, ultrasonic sensor, and servo motor.
The loop function updates the mesh network and checks if the water pump should
be turned on or off based on the value of a variable called "pump".
It also checks if someone is close using the ultrasonic sensor and turns off a
buzzer if someone is detected.

The sketch also includes functions for sending and receiving messages over the
mesh network, handling JSON data, and controlling the servo motor. The
jsonDetectorSensor function creates a JSON object with data from the ultrasonic
sensor, water pump, and servo motor. The handleJsonMessage function parses
incoming JSON data and updates variables accordingly.
The servoMovement function moves the servo motor to different positions.

************************************************************
*/
#include "painlessMesh.h"

#define MESH_PREFIX "homeIOT"
#define MESH_PASSWORD "phyComIOT"
#define MESH_PORT 5555

#include <Servo.h>
#include <DHT_U.h>
#include <DHT.h>
#include <ArduinoJson.h>

const int relayPin = D4; // Digital pin for the relay coil
// Trigger Pin of Ultrasonic Sensor and  Echo Pin of Ultrasonic Sensor
const int trigPin = D2;
const int echoPin = D1;
const int servoPin = D3; // Digital pin for the servo motor

// Initialise the DHT11 component

// 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";

// Duration and distance variables
long duration = 0;
int distance = 0;

int waterLevelValue = 0;

// bool pump = false;
String pump = "OFF";

// Close distance in cm
const int closeDistance = 30;

// create a servo object
Servo myservo;

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("Remedy sending message: " + jsonStr);
  taskSendMessage.setInterval(random(TASK_SECOND * 1, TASK_SECOND * 2));
}

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());
```

```
  // Display other data
  JsonObject waterLevelSensor = doc["WaterLevelSensor"];
  // Serial.printf("Water Level Sensor: %s\n", waterLevelSensor["value"].as<char *>());
  waterLevelValue = waterLevelSensor["value"];
  Serial.printf("Water Level Sensor: %d\n", waterLevelValue);

  // cast pump to string
  pump = doc["pump"].as<String>();
  // pump = doc["pump"];
  // Serial.printf("Pump: %s\n", pump);
}

// 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);
}

// This function is called once at startup to initialize the program
void setup()
{
  Serial.begin(115200); // Initialize the serial communication at a baud rate of 115200

  // Set the debug message types for the mesh network
  // 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

  // Initialize the mesh network with the specified prefix, password, scheduler, and port
  mesh.init(MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT);

  // Set the callback functions for when a message is received, a new connection is made, connections are changed, and node time is adjusted
  mesh.onReceive(&receivedCallback);
  mesh.onNewConnection(&newConnectionCallback);
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

  // Add a task to the user scheduler and enable it
  userScheduler.addTask(taskSendMessage);
```

```
  taskSendMessage.enable();

  // Set the pin modes for the relay, trigger, and echo pins
  pinMode(relayPin, OUTPUT);
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);  // Sets the echoPin as an Input

  // Connect to the WiFi network
  // WiFi.begin(ssid, password);

  // 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");

  // Attach the servo to the specified pin
  myservo.attach(servoPin);
}

void loop()
{
  // it will run the user scheduler as well
  mesh.update();

  // server.handleClient();

  distanceCentimeter();

  // servoMovement();

  Serial.print("Water Pump: ");
  Serial.println(digitalRead(relayPin));
  // put your main code here, to run repeatedly:
  if (pump == "ON")
  {
    Serial.print("Should turn on pump");
    // Turn the relay ON (close the contacts)
    // delay(5000); // Wait for 5 second
    digitalWrite(relayPin, LOW);
    servoMovement();
    // 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);
    // delay(1000); // Wait for 1 second
  }

  const int buzzerPin = D3; // Digital pin for the buzzer

  // if someone is close turn off buzzer
  if (isSomeoneClose(distance))
  {
    noTone(buzzerPin);
  }
}

void servoMovement()
{

  // // Create an int variable called mappedValue
  // // The variable should contain a range of 0 - 180 mapped to the ultrosound distance
  // int mappedValue = map(distance, 0, 40, 0, 180);

  // // Use the mapped value to control the servo
  // myservo.write(mappedValue);

  // Set the servo to 0 degrees
  myservo.write(0);
  delay(1000);

  // Set the servo to 90 degrees
  myservo.write(90);
  delay(1000);
}

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;
  }

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

```
274
275   // // Set the LED brightness
276   // analogWrite(ledPin, ledBrightness);
277
278   // // Print the sensor value and LED brightness to the Serial Monitor
279   // Serial.print("Water Level: ");
280   // Serial.print(waterLevelValue);
281   // Serial.print(" | LED Brightness: ");
282   // Serial.println(ledBrightness);
283
284   // // Add a delay to avoid rapid updates
285   // delay(1000); // 1 second delay
286 }
287
288 void distanceCentimeter()
289 {
290
291   // Clears the trigPin
292   digitalWrite(trigPin, LOW);
293   delayMicroseconds(2);
294
295   // Sets the trigPin on HIGH state for 10 micro seconds
296   digitalWrite(trigPin, HIGH);
297   delayMicroseconds(10);
298
299   // Clears the trigPin
300   digitalWrite(trigPin, LOW);
301
302   // Reads the echoPin, returns the sound wave travel time in microseconds
303   duration = pulseIn(echoPin, HIGH);
304
305   // Calculating the distance in cm
306   distance = (duration * 0.034) / 2;
307
308   // Prints distance to Serial Monitor
309   Serial.print(distance);
310   Serial.println(": Centimeters");
311 }
312
313 // Check if someone is close
314 bool isSomeoneClose(int distance)
315 {
316   if (distance < closeDistance)
317   {
318     return true;
319   }
320   else
321   {
322     return false;
323   }
324 }
325
326 void get_index()
327 {
328
329   distanceCentimeter();
```

```
330
331   // Create the HTML page with the current values
332   String html = "<html><head><title>Dashboard</title></head><body>";
333   html += "<h1>Remedy</h1>";
334   // Display on or off for water pump
335   html += "<p>Water Pump: " + String(digitalRead(relayPin) != 0 ? "OFF" : "ON") + "</p>";
336   // Water Pump Value
337   // Check if someone is close
338   html += "<p>Someone is close: " + String(isSomeoneClose(distance) ? "YES" : "NO") + "</p>";
339   // Closeness value
340   // Servo motor position
341   html += "<p>Servo Motor Position: " + String(myservo.read()) + "</p>";
342   html += "</body></html>";
343
344   // Send the HTML page to the client
345   // server.send(200, "text/html", html);
346 }
347
348 // if water level is high, turn off the pump
349 void jsonDetectorSensor()
350 {
351
352   distanceCentimeter();
353
354   // Add JSON request data
355   doc["Content-Type"] = "application/json";
356   doc["Status"] = 200;
357   doc["nodeId"] = mesh.getNodeId();
358   doc["message"] = "Message from node Remedy";
359
360   doc["someoneClose"] = isSomeoneClose(distance) ? "YES" : "NO";
361
362   // // Add water level sensor JSON object data
363   // JsonObject waterLevelSensor  = doc.createNestedObject("WaterLevelSensor");
364   // waterLevelSensor["sensorName"] = "Water Level";
365   // waterLevelSensor["sensorValue"] = waterLevelValue;
366
367   // Add water pump status
368   JsonObject waterPump = doc.createNestedObject("WaterPump");
369   waterPump["description"] = "Water Pump";
370   waterPump["status"] = digitalRead(relayPin) != 0 ? "OFF" : "ON";
371   waterPump["value"] = digitalRead(relayPin);
372
373   // Check closeness
374   JsonObject closeness = doc.createNestedObject("Distance");
375   closeness["description"] = "Ultrasound";
376   closeness["value"] = distance;
377   closeness["someoneClose"] = isSomeoneClose(distance) ? "YES" : "NO";
378
379   // Add servo motor position
380   JsonObject servoMotor = doc.createNestedObject("ServoMotor");
381   servoMotor["description"] = "Servo Motor";
382   servoMotor["position"] = myservo.read();
383 }
384
385 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);
}

```