```
 /* This code is made by Khuong Huynh
  *
  *
 * READ THIS TEXT BEFORE UPLOADING TO YOUR MICROCONTROLLER!!!
 *
 *
 *  In this code the microcontroller is going to communicate with BLYNK via Wifi
 * The microcontroller starts by connecting to the BLYNK server and the local internett.
 * It then test the servomotor by going side to side and calibrate the photoresistor.
 * This program uses timer intervals to call different functiong that reads the sensorvalues.
 *
 * In the BLYNK app you can decide which of the sensors you want to start reading values from.
 * When you start up the microcontroller all of these are normally on
 * The microcontroller are saving the sensorvalues into arrays and visualising them in a SuperChart, Gauges, Labeled Displays and LED's
 * in live time on the BLYNK app. The arrays can save 50 values, after that it starts replacing the older values.
 *
 * After a set timer, the microcontroller are going to calculate the average value. There's a Slider in the BLYNK app that lets you
 * choose how many of the last readings you want to use (2-50). This is set on 10 when the program starts.
 *
 * Every 30 seconds the ESP32 goes through the arrays and plots the min and max values of each sensor.
 * If one of the criteria for the alarm goes , the program goes into ALARM modus.
 *
 * When in ALARM modus: The ALARM led in BLYNK is blinking.
 *                  Servo motor turns from end position to end position
 *                  The buzzer start buzzing at an annoying frequency.
 *
 * To reset the alarm: The values that created the alarm has to go back to "Normal".
 * This sets the servomotor back into "normal" modus and goes all the way to the right after the first alarm,
 * and all the way to left after the next alarm. This alternates after each alarm.
 *
 * On the BLYNK app theres an "Test" button that test the servomotor by rotating from end to end while it's pressed.
 * This is going to simulate an safety vaulve to check if its stuck or not.
 * Theres also an "Reset" button. This sends the servo to one of the end position. Also alternating after each press.
 *
 * There's an terminal in the BLYNK app that informs you about the status as the microcontroller is running.
 *
 * As an additional feature the microcontroller are programmed to set up an local website that features the sensorvalues
 * Theres also and status indicating if its in alarm og normal mode.
 * This website refreshes automatically every 10 seconds.
 *
 *
 * List of usage of Virtual Pins on BLYNK
 * V0 = Green LED for Tiltsensor
 * V1 = Red LED for TiltSensor
 * V2 = Red LED for Alarm
 * V3 =
 * V4 =
 * V5 =
 * V6 =
 * V7 = Reset button for Servo
 * V8 = Slider for choosing how many "points" to calculate average values
```

* V9 = Terminal

* V10 = Temperature Labeled Value

* V11 = Temperature Gauge

* V12 = Temperature Live Value All Chart

* V13 = Temperatue Average Chart

* V14 = Temperature Max Value Chart

* V15 = Temperature Min Vakue chart

* V16 = Temperature On/Off Menu

* V17 = Temperature Live Value Chart

* V20 = Photoresistor Labeled Value

* V21 = Photoresistor Gauge

* V22 = Photoresistor Live All Value Chart

* V23 = Photoresistor Average Chart

* V24 = Photoresistor Max Value Chart

* V25 = Photoresistor Min Value Chart

* V26 = Photoresistor On/Off Menu

* V27 = Photoresistor Live Value Chart

* V30 =

* V31 =

* V32 = Tilt Live Value All Chart

* V33 =

* V34 = Tilt Max/Min Value Chart

* V35

* V36 = Tilt On/Off Menu

* V37 = Tilt Live Value Chart

* V40 = Distance Labeled Value

* V41 = Distance Gauge

* V42 = Distance Live Value All Chart

* V43 = Distance Average Chart

```
 * V44 = Distance Max Value Chart
 * V45 = Distance Min Value Chart
 * V46 = Distance On/Off Menu
 * V47 = Distance Live Value Chart
 * V99 = Test Button For ServoMotor
 *
 *
 */
 //Importing libraries
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WebServer.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32_Servo.h>
#include <analogWrite.h>


//Defining blynk modules
BlynkTimer timer;
WidgetLED led1 (V0);
WidgetLED led2 (V1);
WidgetLED led3 (V2);
WidgetTerminal terminal(V9);
Servo myservo;
WebServer server(80);

//Auth code from BLYNK app
char auth[] = "4c6Lu1mnZD0FbMNK-95dckX85k7OEhlB";
```

```
//Wifi credentials
char ssid[] = "Khuongs10";
char pass[] = "Kake1234";

//Defining pinnumber and global variables


//Pins
const int pinTemp = 32;
const int pinPhoto = 33;
const int pinTilt = 14;
const int pinTrig = 0;
const int pinEcho = 4;

//Raw values from sensors
bool valTilt;
float valTemp;
int numReadings = 10;
float tempC;
int valPhoto;
float valDistance;

//Boolean values for sensor if they have been "turned" on from BLYNK
bool onTemp = true;
bool onPhoto = true;
bool onTilt = true;
bool onDistance = true;

//Values for calibrating Photoresistor
```

```
int minPhoto = 1023;

int maxPhoto = 0;


//Time variables

unsigned long calTime = 5000;

unsigned long averageTime = 30000;

unsigned long startTime;

unsigned long blinkTime = 1000;

unsigned long lastBlink;


//Values for WebServer and arrays

int averagePhoto;

float averageTemp;

float averageDistance;



//Array & alarm values

float readingsTemp[50];

int readingsPhoto[50];

int readingsTilt[50];

float readingsDistance[50];

int readIndexTemp = 0;

int readIndexPhoto = 0;

int readIndexTilt = 0;

int readIndexDistance = 0;

int testButton = 0;

int servoEndPos = 0;

int servoPos = 0;

int alarmNumb = 0;
int freq = 1000;
int channel = 2;
int resolution = 8;

//Alarm limits
int tempAlarm_Limit = 30;
int photoAlarm_Limit = 100;
int tiltAlarm_Limit = 1;
int distanceAlarm_Limit = 200;
int servoAlarmVal = 180;
int servoResetVal = 0;
int lastAlarm = 0;
int alarmCount = 0;




void myTimerEvent1()
{
  //Reading temp value and writing to Blynk
  if( !testButton && onTemp ){
  valTemp = analogRead(pinTemp);
  float volt = (valTemp / 1023.0);
  tempC = (volt - 0.5) * 100; // converting into Celsius
  readingsTemp[readIndexTemp] = tempC; //Saving the temp into an array
  Blynk.virtualWrite(V10, tempC); //Labeled value Temp
  Blynk.virtualWrite(V11, tempC); // Gauge value Temp
  Blynk.virtualWrite(V12, tempC); //All Chart value Temp
  Blynk.virtualWrite(V17, tempC); //Chart value Temp
```

```
 readIndexTemp += 1;
 if (readIndexTemp > 50) readIndexTemp = 0; // Resetting readIndexTemp
 }
}

void myTimerEvent2() {
 if( !testButton && onPhoto ){
 //Reads, maps and constrains Fhotoresistor value
 valPhoto = analogRead(pinPhoto);
 valPhoto = map(valPhoto, minPhoto, maxPhoto, 0, 255);
 valPhoto = constrain(valPhoto, 0, 255);
 readingsPhoto[readIndexPhoto] = valPhoto; //Saving valPhoto into an array
 Blynk.virtualWrite(V20, valPhoto); //Labeled value Photoresistor
 Blynk.virtualWrite(V21, valPhoto); //Gauge value Photoresistor
 Blynk.virtualWrite(V22, valPhoto); //All Chart value Photoresistor
 Blynk.virtualWrite(V27, valPhoto); //Chart value Photoresistor
 readIndexPhoto +=1;
 if(readIndexPhoto > 50) readIndexPhoto = 0; //Resetting readIndexPhto
 }
}

void myTimerEvent3() {
 if( !testButton && onTilt ){
 //Reading value of tiltsensor
 valTilt = digitalRead(pinTilt);
 if (valTilt == true) {
  led1.on();// Green led on
  led2.off();// Red led off
  Blynk.virtualWrite(V32, 0); //Chart tiltsensor "off"
```

```
    readingsTilt[readIndexTilt] = 0; //Tiltsensor has tilted, an "true" value been added
  }
  else {
    led1.off();//Green led off
    led2.on();//Red led on
    Blynk.virtualWrite(V32, 1); //Chart tiltsensor "on"
    readingsTilt[readIndexTilt] = 1;//Tilt sensor not tilted, an false value been added
  }
  readIndexTilt +=1;
  if(readIndexTilt >50) readIndexTilt = 0; //Resetting readIndexTilt
  }
}

void myTimerEvent4(){
  //Calculate the distace
  if( !testButton && onDistance ){
  valDistance = ultraSonic();
  valDistance = constrain(valDistance, 2, 400); //Constrain for min and max value
  readingsDistance[readIndexDistance] = valDistance; //Saving the Distance into an array
  Blynk.virtualWrite(V40, valDistance); //Labeled value Distance
  Blynk.virtualWrite(V41, valDistance); // Gauge value Distance
  Blynk.virtualWrite(V42, valDistance); //All Chart value Distance
  Blynk.virtualWrite(V47, valDistance); //Chart value Distance
  readIndexDistance += 1;
  if (readIndexDistance > 50) readIndexDistance = 0; // Resetting readIndexDistance
  }
}

void myTimerEvent5() {
```

```
//Function for calculating average temp, distance and photoRes
//The system has to run for an amount of time before this function to run
if( !testButton){
if ((millis() - startTime) >= averageTime ) {
 //Creating variables internally
 float totalTemp = 0;
 float totalDistance = 0;
 int totalPhoto = 0;
 int k = 0;
 int h = 0;
 int t = 0;

 //Calculating total values from last readIndex
 for (int i = 0; i < numReadings; i++) {
  int readIndexTemp_Average = (readIndexTemp -1) - i;
  int readIndexPhoto_Average = (readIndexPhoto -1) - i;
  int readIndexDistance_Average = (readIndexPhoto-1)-i;

  //If the index go below 0, it starts counting from 50
  if( readIndexTemp_Average < 0){
   readIndexTemp_Average = 49 - k;
   k+=1;
  }
   if(readIndexPhoto_Average < 0){
   readIndexPhoto_Average = 49 - h;
   h+=1;
  }
   if(readIndexDistance_Average < 0){
   readIndexDistance_Average = 49 - t;
```

```
    t+=1;

   }

   totalTemp = totalTemp + readingsTemp[readIndexTemp_Average];

   totalPhoto = totalPhoto + readingsPhoto[readIndexPhoto_Average];

   totalDistance = totalDistance + readingsDistance[readIndexDistance_Average];


  }


  averagePhoto = (totalPhoto / numReadings); //Calculating average Photo

  averageTemp = (totalTemp / numReadings); //Calculating average Temp

  averageDistance = (totalDistance /numReadings); //Calculating average Distance

  //Printing to terminal

  terminal.print("The average temp is: ");

  terminal.println(averageTemp);

  terminal.print("The average photoval is: ");

  terminal.println(averagePhoto);

  terminal.print("The avrg dist is: ");

  terminal.println(averageDistance);

  terminal.flush();

  Blynk.virtualWrite(V13, averageTemp); //Chart average value Temp

  Blynk.virtualWrite(V23, averagePhoto); //Chart average value Photo

  Blynk.virtualWrite(V43, averageDistance); //Chart average value Distance

 }

 }

}


void myTimerEvent6(){

 //Function for finding MIN and MAX values

 //The system has to run for an amount of time before this function to run
```

```
if ((millis() - startTime) >= averageTime ) {
 if( !testButton){
  //Creating variables to find MIN and MAX values
  int lowTemp = 125;
  int highTemp = -50;
  int lowPhoto = 255;
  int highPhoto = 0;
  int highDistance = 2;
  int lowDistance = 400;
  int boolTilt = 0;

  for(int i=0; i <= 49; i++){
   //Reading the values from arrays
   int arrayTemp = readingsTemp[i];
   int arrayPhoto = readingsPhoto[i];
   int arrayDistance = readingsDistance[i];
   int arrayTilt = readingsTilt[i];
   //Replace the MIN and MAX values
   if( arrayTemp > highTemp) highTemp = arrayTemp;
   if( arrayTemp < lowTemp) lowTemp = arrayTemp;
   if( arrayPhoto > highPhoto) highPhoto = arrayPhoto;
   if( arrayPhoto < lowPhoto) lowPhoto = arrayPhoto;
   if( arrayTilt == 1) boolTilt = 1;
   if( arrayDistance < lowDistance) lowDistance = arrayDistance;
   if( arrayDistance > highDistance) highDistance = arrayDistance;
  }
  //Charting the MIN and MAX values
  Blynk.virtualWrite(V14, highTemp);
  Blynk.virtualWrite(V15, lowTemp);
```

```
   Blynk.virtualWrite(V24, highPhoto);
   Blynk.virtualWrite(V25, lowPhoto);
   Blynk.virtualWrite(V34, boolTilt);
   Blynk.virtualWrite(V44, highDistance);
   Blynk.virtualWrite(V45, lowDistance);
   //Sends values to alarmfunction
   alarmFunction(highTemp, lowPhoto, boolTilt, highDistance);
  }
 }
}

void myTimerEvent7(){
 //Testing servomotor
 //When testing the servo, sensors do not read values
 //Testing does not work if theres an alarm active
 if ( alarmNumb == 0 && testButton == 1){
  servoMove();
 }
}

void myTimerEvent8(){
 //Alarm timer function where the Alarm LED blinks, Buzzer tones, and makes servo go from end to end
 if ( alarmNumb != 0){
  lastAlarm = alarmNumb; //Saves what the last alarm numb was
  if ( (millis()-lastBlink) >= blinkTime){ //Timer for blink, tonechange and servoswipe
   if (led3.getValue()){
    led3.off();                    //Led off
    ledcWriteTone(channel, 1000);   //Buzzer 1000Hz
    myservo.write(servoResetVal);   //Servo end position
   servoEndPos = 0;
   servoPos = servoResetVal;       //Indicating servo position
   lastBlink = millis();
  }
  else{
   led3.on();                     //Led on
   ledcWriteTone(channel, 2000);   //Buzzer 2000Hz
   myservo.write(servoAlarmVal);   //Servo end position
   servoEndPos = 1;
   servoPos = servoAlarmVal;       //Indicating servo position
   lastBlink = millis();
  }
 }
}
//If theres been an alarm and is now reset the servo will go to one of the end position,
//depending on how many alarm theres been
else if ( lastAlarm != 0 && alarmNumb == 0){
 alarmCount +=1;                // Counts alarms
 if ( alarmCount % 2 == 0){      // If theres an even number of alarms the servo will go to the right
  myservo.write(servoResetVal);
  servoPos = servoResetVal;     // Storing the position of servo
  servoEndPos = 0;             // Storing the end pos of servo
  lastAlarm = alarmNumb;       // Indicating last alarm was "no Alarm"
 }
 else{
  myservo.write(servoAlarmVal); //If theres an odd number of alarms the servo will go to the left
  servoPos = servoAlarmVal;     // Storing the position of servo
  servoEndPos = 1;             //Storing the end pos of servo
  lastAlarm = alarmNumb;       //Indicating last alarm was "no Alarm"
  }
 }
 //If none of the criterias above, turns of led and buzzer
 else{
  led3.off();
  ledcWriteTone(channel, 0);
 }
}


/*The virtual pins V16, V26, V36 and V46 are menu options on BLYNK that lets you
 * choose if the sensors are active or inactive
*/

//Temperature menu option
BLYNK_WRITE(V16){
 switch ( param.asInt())
 {
  case 1:
   onTemp = true;
   terminal.print("Temperature readings ON");
   terminal.flush();
   break;
  case 2:
   onTemp = false;
   terminal.print("Temperature readings OFF");
   terminal.flush();
   break;
 }
```

```
}
//Photoresistor menu option
BLYNK_WRITE(V26){
  switch ( param.asInt())
  {
    case 1:
      onPhoto = true;
      terminal.print("Photoresistor readings ON");
      terminal.flush();
      break;
    case 2:
      onPhoto = false;
      terminal.print("Photoresistor readings OFF");
      terminal.flush();
      break;
  }
}
//Tiltsensor menu option
BLYNK_WRITE(V36){
  switch ( param.asInt())
  {
    case 1:
      onTilt = true;
      terminal.print("Tilt readings ON");
      terminal.flush();
      break;
    case 2:
      onTilt = false;
      terminal.print("Tilt readings OFF");
      terminal.flush();
      break;
  }
}
//Distance sensor menu option
BLYNK_WRITE(V46){
  switch ( param.asInt())
  {
    case 1:
      onDistance = true;
      terminal.print("Distance readings ON");
      terminal.flush();
      break;
    case 2:
      onDistance = false;
      terminal.print("Distance readings OFF");
      terminal.flush();
      break;
  }
}

BLYNK_WRITE(V7) {
  //Resetting buzzer and LED alarms
  if( param.asInt()){
    if ( servoEndPos == 1){
      myservo.write(servoResetVal);   //If the servo last end pos was right it goes to the left
      servoEndPos = 0;                //Stores that last end pos was left
      servoPos = servoResetVal;       //Stores servo position
    }
  else{
   myservo.write(servoAlarmVal);   //If the servo last en pos was left it goes to the right
   servoEndPos = 1;                //Stores last end pos was right
   servoPos = servoAlarmVal;       //Store servo position
  }

  Serial.println("Alarm reset");
 }
}

BLYNK_WRITE(V8) {
 //Slider for choosing how many values to use for calculating the average
 numReadings = param.asInt();
 //Writing to terminal how many values are used
 terminal.print("V8 Slider value is: ");
 terminal.println(numReadings);
 terminal.flush();
}

BLYNK_WRITE(V99){
 //Test button for servomotr
 testButton  = param.asInt();
}



void setup()
{
 // Debug console & connecting to wifi
```

```
Serial.begin(115200);
delay(100);
Serial.println("Connecting to ");
Serial.println(ssid);
//Connecting to the local wi-fi
WiFi.begin(ssid, pass);

//Controls if connectiong is made and prints the IP adress
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected..!");
Serial.print("Got IP: ");  Serial.println(WiFi.localIP());
server.on("/", handle_OnConnect);
server.onNotFound(handle_NotFound);
server.begin();
Serial.println("HTTP server started");

//Setting pinMode
pinMode(pinTemp, INPUT);
pinMode(pinPhoto, INPUT);
pinMode(pinTilt, INPUT);
pinMode(pinTrig, OUTPUT);
pinMode(pinEcho, INPUT);

//Blynk.begin(auth, ssid, pass);
// You can also specify server:
```

```
//Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 80);
Blynk.begin(auth, ssid, pass, IPAddress(91, 192, 221, 40), 8080);

//Attach pin to buzzer
ledcSetup(channel, freq, resolution);
ledcAttachPin(12, channel);

//Attach pin to servomotor, and test the motor
myservo.attach(25);
for ( int i = 0; i <=180; i++){
  myservo.write(i);
  servoPos = i;
  delay(10);
}
for ( int i = 180; i >=0; i--){
  myservo.write(i);
  servoPos = i;
  delay(10);
}

//Resetting the arrays for temp and photo
for (int thisReading = 0; thisReading <= 50; thisReading++) {
  readingsTemp[thisReading] = 0;
  readingsPhoto[thisReading] = 0;
  readingsTilt[thisReading] = 0;
  readingsDistance[thisReading] = 0;
}

//Turning on led signalising calibration an sets numReading to 10
```

```
led1.on();

led2.on();

Blynk.virtualWrite(V8, 10);


//Clearing the terminal

terminal.clear();


//Calibrate for 5 secs

unsigned long timeNow = millis();

while ((millis()-timeNow) < calTime) {

  valPhoto = analogRead(pinPhoto);

  if (valPhoto < minPhoto) minPhoto = valPhoto;//Record the maximum sensor value

  if (valPhoto > maxPhoto) maxPhoto = valPhoto; //Record the minimum sensor value


}


//Turning off led signalising calibration completed

led1.off();

led2.off();


//Writing max and min value for Photoresistor

terminal.print("Max value for Photoresistor:");

terminal.println(maxPhoto);

terminal.print("Min photovalue for Photoresistor: ");

terminal.print(minPhoto);

terminal.flush();


//Setting the interval for the timers

timer.setInterval(2000L, myTimerEvent1);//Temp sensor every 5 sec
```

```
  timer.setInterval(1000L, myTimerEvent2);//Photo resistor every 2 sec
  timer.setInterval(500L, myTimerEvent3);//Tilt sensor every sec
  timer.setInterval(750L, myTimerEvent4);//HC-SR04 sensor every sec
  timer.setInterval(10000L, myTimerEvent5);//Calculating average every 10sec
  timer.setInterval(30000L, myTimerEvent6);//Max & Min calue every 30 sec
  timer.setInterval(20L, myTimerEvent7); //Servo test every 20 ms
  timer.setInterval(10L, myTimerEvent8);//Alarm LED blink and buzzer tone change
  //Start time set
  startTime = millis();
}


//This section of code contains different function that are used by the timer intervals

float ultraSonic(){
  //Function for the SR-HC04 sensor
  float duration = 0;
  float distance = 0;

  digitalWrite(pinTrig, LOW);
  delayMicroseconds(2);
  digitalWrite(pinTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(pinTrig, LOW);

  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
  duration = pulseIn(pinEcho, HIGH);
  // Convert the time into a distance
  distance = duration/58.2;
  return distance;
}

//Function for alarms
void alarmFunction(int temp, int photo, int tilt, int distance){
  //Creating variables used in the function
  bool alarmTemp = false;
  bool alarmPhoto = false;
  bool alarmTilt = false;
  bool alarmDistance = false;
  //Checks if values are over/under limit values
  if ( temp > tempAlarm_Limit) alarmTemp = true;
  if ( photo < photoAlarm_Limit) alarmPhoto = true;
  if ( tilt == tiltAlarm_Limit) alarmTilt = true;
  if ( distance > distanceAlarm_Limit) alarmDistance = true;
  //Sets the alarmcodes
  if ( alarmTemp == true && alarmPhoto == true) alarmNumb = 1;
  else if ( alarmTemp == true && alarmDistance == true) alarmNumb = 2;
  else if ( alarmDistance == true && alarmPhoto == true) alarmNumb = 3;
  else if ( alarmTilt == true) alarmNumb = 4;
  else { alarmNumb = 0;}


  alarmSwitch(alarmNumb);

}
```

```
//Switch/case function for alarm number
void alarmSwitch(int alarmNumb){

  switch( alarmNumb ){
    case 0:
      //No Alarm
      break;
      //Temp and Photo alarm
    case 1:
      Serial.println("ALARM! To high Temp and to low Photo");
      terminal.println("ALARM! To high Temp and to low Photo");
      terminal.flush();
      break;
      //Temp and Distance alarm
    case 2:
      Serial.println("ALARM! To high Temp and to big Distance");
      terminal.println("ALARM! To high Temp and to big Distance");
      terminal.flush();
      break;
      //Distance and Photo alarm
    case 3:
      Serial.println("ALARM! To high Distance and to low Photo");
      terminal.println("ALARM! To high Distance and to low Photo");
      terminal.flush();
      break;
      //Tilt alarm
    case 4:
      Serial.println("ALARM! Door is open!");
      terminal.println("ALARM! Door is open!");
      terminal.flush();
      break;
  }
}

void servoMove(){
  //Makes servo move from side to side
  if( servoEndPos == 0){
      myservo.write(servoPos);
      servoPos += 5;
      if ( servoPos == 180) servoEndPos = 1;
    }
     else if (servoEndPos == 1){
      myservo.write(servoPos);
      servoPos -= 5;
      if (servoPos == 0) servoEndPos = 0;
     }
}

//Section of code for creating the local website

void handle_OnConnect() {
  //If connection is made, sends values to create HTML webserver
  float Temperature = averageTemp; // Gets the values of the temperature
  int PhotoRes = averagePhoto; // Gets the values of the humidity
  float Distance = averageDistance; //Gets the value of SR-HC04
  String alarmText;
  if ( alarmNumb == 0) alarmText = "Normal";
  else{ alarmText = "Alarm";}
  //Sends in sensor values to HTML creating function
  server.send(200, "text/html", SendHTML(Temperature,PhotoRes, Distance, alarmText));
}

void handle_NotFound(){
  //If no connection you get an error page
  server.send(404, "text/plain", "Not found");
}
//HTML creator function
String SendHTML(float Temperature,float PhotoRes, float Distance, String Text){
  String axl = "<!DOCTYPE html> <html>\n";
  axl +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
  axl +="<title>ESP32 Sensors Readings</title>\n";
  axl +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: left;}\n";
  axl +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
  axl +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
  axl +="</style>\n";
  axl +="</head>\n";
  axl +="<body>\n";
  axl +="<div id=\"webpage\">\n";
  axl +="<h1>ESP32 Sensors Readings</h1>\n";

  axl +="<p>Temperature: ";
  axl +=(int)Temperature;
  axl +="C";
  axl +="<p>PhotoResistor: ";
  axl +=(int)PhotoRes;
  axl +="<p>Distance: ";
  axl +=(float)Distance;
  axl +=" cm";
  axl +="<p>Status: ";
  axl +=(String)Text;
  axl +="<br> ";
  axl +="<br> ";
  axl +="<br> ";
  axl +="<br> ";
  axl +="<br> ";
  axl +="<br> ";
  axl +="<br> ";
  axl +="<img src=http://www.agdervent.no/images/logo/bil-agderventilasjon-72dpi-farge.jpg>";//Image
  axl +="<br> ";
  axl +="<br> ";
  axl +="Made by Khuong Huynh";
  axl +="</p>";
  axl +="<meta http-equiv=refresh content=10>";//Refreshes every 10 sec automatically

  axl +="</div>\n";
  axl +="</body>\n";
  axl +="</html>\n";
  return axl;
}



void loop()
{
  Blynk.run();

  timer.run(); // Initiates BlynkTimer

  server.handleClient();//Runs the html webpage


}
```