Project code explanation

28/01/2022

Contents

ESP32 Code (C++ variant)

```
//all includes (libaries)
#include <DHT.h>
#include <DHT_U.h>
#include <Adafruit_Sensor.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <LSM303.h>

// defines LSM303 as "compass"
LSM303 compass;

// defines wifi credentials
const char* ssid = "Laptop";
const char* password = "password";



//Server (website) name and address to send request to (used to store data)
String serverName = "http://med.clockr.net/storedata.php?";

// Set up loop (runs on start up)
```

This is the first section of code before any loops, this is basically just used for defining libraries and variables. The libraries are as follows DHT is not required however it make the output data easier to read, the adafruit sensor library is a requirement for LSM303 which deals with the output for the sensor. WIFI and httpclient are used for connecting to the wifi and then sending data via Http request (more info on http: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview)

Wire I am not entirely sure what it does to be honest, as far as I can see it is just required for LSM303 as it has something to do with the I2C data transfer protocol (I2C info: https://en.wikipedia.org/wiki/I%C2%B2C)

LSM303 is then defined as “compass” which is used further down in the program

Then variables are declared SSID and password are for network connection (currently showing my laptop hotspot details)

Server name is the basic name of the server, this is where HTTP requests with the data are sent. This will be more apparent further in the code but for now the data is sent to a webpage (storedata.php) which then moves it to the database

```
// Set up loop (runs on start up)
void setup() {
  // starts serial monitor
  Serial.begin(115200);


  //begins wifi (tries to connect with credentials above)
  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  // prints local IP address
  Serial.println(WiFi.localIP());

  //begins wire (tbh idk what this does just that its a depdency)
  Wire.begin();
  // starts compass
  compass.init();
  compass.enableDefault();
}
```

This is the setup loop, it runs once on start-up of the ESP, firstly it begins the serial monitor at a baud rate of 115200 baud rate means it can send 115,200 bits of data per second, this is required because of how fast the data is sent we have to have a high “refresh rate” in order to send data at a frequency high enough to display data correctly

Next the Wi-Fi is set up, it tries to connect with the credentials listed in the previous section it then runs a loop to see if WIFI status is not equal to connected, if not it prints “connecting…..”. when the WIFI does connect, it says connected to Wi-Fi and displays the local IP address of the device (192.168.X.X) wire then begins which deals with I2C and the compass in initialised and then the default parameters are stated (listed in the sensor and code library datasheet)

```
void loop() {


  //Check WiFi connection status
  if(WiFi.status()== WL_CONNECTED){

    HTTPClient http;
    // reads compass values
    compass.read();
    // DEBUG: Serial.println(compass.a.x);
    //prepares server query
    String serverPath = serverName + "x=" + compass.a.x + "&y=" + compass.a.y + "&z=" + compass.a.z;

    //sends data to server
    http.begin(serverPath);

    // Send HTTP GET request
    int httpResponseCode = http.GET();

    if (httpResponseCode>0) {
      // checks if response was valid (200 OK)
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      String payload = http.getString();
      Serial.println(payload);
    }
    else {
      //if theres an error, report the error code (403/500/401)
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }
    // Free resources
    //closes HTTP request
    http.end();
  }
  else {
    //prints disconected if wifi drops
    Serial.println("WiFi Disconnected");
  }
  // delays for 100ms and reruns
  delay(100);
}
```

This is the main loop of the code; this loops over and over until the device is either reset or turned off.

Firstly, the code checks that it still has a WIFI signal, it then defines the HTTPClient function as http just for ease of programming, the compass is then read, this gets a “screenshot” of the current values of the sensor at the time executed. The server path is then defined as the server’s name (on the first section) + the components being sent to the server (the X,Y and Z cords of the sensor)

Compass.a.x is only one of the options, a is for acceleration there is also the option for compass.m.x which get the magnitude of the data, this is easily implemented in the future if required.

The HTTP client then initialises with the server path, it then sends a GET request (part of HTTP in terms of requests more info can be found at: https://www.w3schools.com/tags/ref_httpmethods.asp) we are not using GET requests in the way they were designed but it will make sense when getting onto the server code.

The ESP then checks the data was sent correctly and prints either 200 which is the OK code saying the data was received or it prints the error code (404/403/500 ect) depending on what the error is.

At the bottom there is delay(100), this means It sleeps for 100 ms before running again giving us a range of 10Hz

Server code (PHP)

Storedata.php

```php
<?php
//inclides db config page
include "db.php";


// defines $x, $y, $z as the variables sent by esp
$x = $_GET['x'];
$y = $_GET['y'];
$z = $_GET['z'];


//gets time in millisecs
$time = time() * 1000;




//stores data in database
$conn->query("INSERT INTO `data` (`id`, `unix`, `x`, `y`, `z`) VALUES ('', '{$time}', '{$x}', '{$y}', '{$z}')");


?>
```

This is the storedata.php file that was in the server path of the ESP. firstly, it includes a file called “db.php” which includes all of the connection information for the database” it looks for requests where X,Y and Z are sent as GET requests and stores there values as $x, $y and $z. it also get the current time in ms to attach a time value to the co ordinates as it makes it much easier to display the data as you know the amount of time between two sets of data.

The server then sends a “query” to the SQL database of “INSERT INTO” (https://www.w3schools.com/sql/sql_insert.asp) this inserts the data received into a table called “data” with the the following values

ID is just the unique identifier for each row and is auto generated by the database so you can have multiple sets of data and they wont clash as they all have different IDs. UNIX refers to the amount of milliseconds since the unix epoch (1/1/1970) X,Y and Z are just the data received from the ESP

Getdata.php

```php
//includes db config
include "db.php";


//asks db for any row where time is in the past 60 secs
$sql = $conn->query("SELECT * FROM `data` WHERE 1 ORDER BY `id` DESC");

//defines counter as 0, used for array
$counter = 0;

//sets a loop to go through the associative array
while ($r = $sql->fetch_assoc()){

  //defines arrays for the data in the database
  $dataX[] = $r['x'];
  $dataY[] = $r['y'];
  $dataZ[] = $r['z'];
  $dataAvg[] = ($r['z'] + $r['y'] + $r['z']) / 3;
  $dataTime[] = $counter;
  $counter = $counter + 1;
  //sets a counter to make sure there are only 60 results
  if ($counter > 100){
    break;
  }

}
// sets and array of arrays assoiated to the axsis
$array = array("x" => $dataX, "y" => $dataY, "z" => $dataZ, "avg" => $dataAvg, "count" => $dataTime);

//returns the data as JSON standard
echo json_encode($array);
```

Getdata.php is what the webpage calls to display data, this is quite a bit more complicated, starting from the beginning the database is queried to return all data in the “data” table. It does this by auto generated ID in descending order (newest data first) next a counter is defined at 0 this is used for only showing the 100 newest rows of data. Next there is a loop set up. This in simple terms loops through each row of called data. It stores the data as an associative array which means all values have key and you can search by key so if you need x from every row you can make a loop with $r[‘x’] and it returns all of the X values. While this loop executes it makes 5 arrays, one each for X,Y and Z and then it creates and average and how many datavalues are stored by way of the counter. The counter is important as if there is only 50 values the chart of the webpage would still be trying to display the fault 100 and would break. The counter is then increased by one and checked if the counter is more than 100, if so the loop break if not it repeats until there are 100 records in the arrays. A new associative array is the created with the values of the data arrays. (https://en.wikipedia.org/wiki/Associative_array) and printed “echoed” in a JSON encoded format (https://en.wikipedia.org/wiki/JSON#:~:text=JSON%20(JavaScript%20Object%20Notation%2C%20pronounced,(or%20other%20serializable%20values).)

This is then used by the web page to generate required charts.

Web page (JavaScript)

Charts.js

```
const labels = ["1","2","3","4","5","6","7","8","9","10",];

const data = {
  labels: labels,
  datasets: [{
    label: 'X',
    backgroundColor: 'rgb(255, 99, 132, 0.25)',
    borderColor: 'rgb(255, 99, 132, 0.4)',
    data: [0,0,0,0,0,0,0,0,0,0],
    tension: 0.1,
    borderWidth: 1
  },
  {
    label: 'Y',
    backgroundColor: 'rgb(30,144,255, 0.25)',
    borderColor: 'rgb(30,144,255, 0.4)',
    data: [0,0,0,0,0,0,0,0,0,0],
    tension: 0.1,
    borderWidth: 1
  },{
    label: 'Z',
    backgroundColor: 'rgb(124,252,0, 0.25)',
    borderColor: 'rgb(124,252,0, 0.4)',
    data: [0,0,0,0,0,0,0,0,0,0],
    tension: 0.1,
    borderWidth: 1
  },{
    label: 'Average',
    backgroundColor: 'rgb(0,0,0)',
    borderColor: 'rgb(0,0,0)',
    data: [0,0,0,0,0,0,0,0,0,0],
    tension: 0.1,
    borderWidth: 1
  },
```

```
  const config = {
    type: 'line',
    data: data,
    options: {}
  };

  const myChart = new Chart(
    document.getElementById('myChart'),
    config
  );



function updatechart(){

fetch("http://med.clockr.net/getdata.php")
.then(x => x.json())
.then(f => {
  let x = f.x;
  let y= f.y;
  let z = f.z;
  let avg = f.avg;
  let count = f.count;
  myChart.config.data.labels = count;
  myChart.config.data.datasets[0].data = x;
  myChart.config.data.datasets[1].data = y;
  myChart.config.data.datasets[2].data = z;
  myChart.config.data.datasets[3].data = avg;
  myChart.update();
})
```

The image on the left defines the labels for the chart (default is 10 because I didn't want to type out 100 values) it then defines the data sets X, Y, Z and Avg. it defines the colours for each line, the "tension" (how rounded the points are 0 is no roundness) and the "borderwidth" (thickness of the line)

The image on the right defines the config data for the chart, sets it as a line chart and to use the data set on the left. The function updatechart is responsible for the chart updates, it sends a request to getdata.php, declares the data as JSON (which is how it was exported) (X and F are just variable name values used to declare the data set received) it then declare X, Y, Z, Avg and count as the items from the associative JSON array and lets counts as labels and then X – Avg as the datasets (javascript uses 0 as the first values instead of 1)