```
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>

// size of buffer used to capture HTTP requests
#define REQ_BUF_SZ   40

// MAC address from Ethernet shield sticker under board
byte mac[] = {0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF};
IPAddress ip(192, 168, 0, 105); // IP address, may need to change depending on network
EthernetServer server(80);  // create a server at port 80
File webFile;
char HTTP_req[REQ_BUF_SZ] = {0}; // buffered HTTP request stored as null terminated string
char req_index = 0;              // index into HTTP_req buffer

// Weather vane variables by DV
String readString;
String vaneReading;
int sensorNumber;

// Input Pin Assignment by DV
int pin2 = A1;   //Hall Effect Sensor 1
int pin3 = A2;   //Hall Effect Sensor 2
int pin4 = A3;   //Hall Effect Sensor 3
int pin5 = A4;   //Hall Effect Sensor 4
int pin6 = 6;   //Hall Effect Sensor 5
int pin7 = 7;   //Hall Effect Sensor 6
int pin8 = 8;   //Hall Effect Sensor 7
int pin9 = 9;   //Hall Effect Sensor 8

// Pin and reading variables by DV
int pinVal2;    //read pin and assign high or low
int pinVal3;    //read pin and assign high or low
int pinVal4;    //read pin and assign high or low
int pinVal5;    //read pin and assign high or low
int pinVal6;    //read pin and assign high or low
int pinVal7;    //read pin and assign high or low
int pinVal8;    //read pin and assign high or low
int pinVal9;    //read pin and assign high or low
int reading;    //total weight less active sensors

void setup()
{
   // disable Ethernet chip
```

```
    pinMode(10, OUTPUT);
    digitalWrite(10, HIGH);

    Serial.begin(9600);       // for debugging

    // initialize SD card
    Serial.println("Initializing SD card...");
    if (!SD.begin(4)) {
        Serial.println("ERROR - SD card initialization failed!");
        return;    // init failed
    }
    Serial.println("SUCCESS - SD card initialized.");
    // check for index.htm file
    if (!SD.exists("index.htm")) {
        Serial.println("ERROR - Can't find index.htm file!");
        return;  // can't find index file
    }
    Serial.println("SUCCESS - Found index.htm file.");
    pinMode(3, INPUT);        // switch is attached to Arduino pin 3

    Ethernet.begin(mac, ip);  // initialize Ethernet device
    server.begin();           // start to listen for clients

    //Pin mode setup by DV
    pinMode(pin2,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin3,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin4,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin5,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin6,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin7,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin8,INPUT_PULLUP); //pin will be high if sensor is not active
    pinMode(pin9,INPUT_PULLUP); //pin will be high if sensor is not active
}

void loop()
{
    EthernetClient client = server.available();  // try to get client

    if (client) {  // got client?
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {   // client data available to read
                char c = client.read(); // read 1 byte (character) from client
                // buffer first part of HTTP request in HTTP_req array (string)
                // leave last element in array as 0 to null terminate string (REQ_BUF_SZ - 1)
                if (req_index < (REQ_BUF_SZ - 1)) {
                    HTTP_req[req_index] = c;          // save HTTP request character
                    req_index++;
                }
                // last line of client request is blank and ends with \n
                // respond to client only after last line received
                if (c == '\n' && currentLineIsBlank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                     client.println("Connection: keep-alive");
                    client.println();
                    // Ajax request
                    if (StrContains(HTTP_req, "ajax_switch")) {
                        // read switch state and send appropriate paragraph text
                        GetSwitchState(client);
                    }
                    else {  // web page request
                        // send web page
                        webFile = SD.open("index.htm");        // open web page file
                        if (webFile) {
                            while(webFile.available()) {
                                client.write(webFile.read()); // send web page to client
                            }
                            webFile.close();
                        }
                    }
                    // display received HTTP request on serial port
                    Serial.println(HTTP_req);
                    // reset buffer index and all buffer elements to 0
                    req_index = 0;
                    StrClear(HTTP_req, REQ_BUF_SZ);
                    break;
                }
                // every line of text received from the client ends with \r\n
                if (c == '\n') {
                    // last character on line of received text
                    // starting new line with next character read
                    currentLineIsBlank = true;
                }
                else if (c != '\r') {
                    // a text character was received from client
                    currentLineIsBlank = false;
```

```
                }
            } // end if (client.available())
        } // end while (client.connected())
        delay(1);      // give the web browser time to receive the data
        client.stop(); // close the connection
    } // end if (client)
}

// send the state of the switch to the web browser
void GetSwitchState(EthernetClient cl)
{
    int pinWt2=1;   //assign binary pin weight
  int pinWt3=2;   //assign binary pin weight
  int pinWt4=4;   //assign binary pin weight
  int pinWt5=8;   //assign binary pin weight
  int pinWt6=16;  //assign binary pin weight
  int pinWt7=32;  //assign binary pin weight
  int pinWt8=64;  //assign binary pin weight
  int pinWt9=128; //assign binary pin weight



  pinVal2 = digitalRead(pin2);    //read pin
    if(pinVal2 == LOW){pinWt2=0;} //test if sensor is active low
  pinVal3 = digitalRead(pin3);    //read pin
    if(pinVal3 == LOW){pinWt3=0;} //test if sensor is active low
  pinVal4 = digitalRead(pin4);    //read pin
    if(pinVal4 == LOW){pinWt4=0;} //test if sensor is active low
  pinVal5 = digitalRead(pin5);    //read pin
    if(pinVal5 == LOW){pinWt5=0;} //test if sensor is active low
  pinVal6 = digitalRead(pin6);    //read pin
    if(pinVal6 == LOW){pinWt6=0;} //test if sensor is active low
  pinVal7 = digitalRead(pin7);    //read pin
    if(pinVal7 == LOW){pinWt7=0;} //test if sensor is active low
  pinVal8 = digitalRead(pin8);    //read pin
    if(pinVal8 == LOW){pinWt8=0;} //test if sensor is active low
  pinVal9 = digitalRead(pin9);    //read pin
    if(pinVal9 == LOW){pinWt9=0;} //test if sensor is active low

reading = (pinWt2+pinWt3+pinWt4+pinWt5+pinWt6+pinWt7+pinWt8+pinWt9);  //add
   //pin weights with active pins equal to 0
   cl.print("Wind Direction: ");
    if(reading==254){cl.println("N"),vaneReading="N";}  //if weight matches, print direction
```

```
     if(reading==252){cl.println("NNW"),vaneReading="NNW";}   //if weight matches, print
direction
      if(reading==253){cl.println("NW"),vaneReading="NW";}   //if weight matches, print direction
       if(reading==249){cl.println("WNW"),vaneReading="WNW";}   //if weight matches, print
direction
        if(reading==251){cl.println("W"),vaneReading="W";}    //if weight matches, print direction
         if(reading==243){cl.println("WSW"),vaneReading="WSW";}   //if weight matches, print
direction
          if(reading==247){cl.println("SW"),vaneReading="SW";}   //if weight matches, print direction
           if(reading==231){cl.println("SSW"),vaneReading="SSW";}   //if weight matches, print
direction
            if(reading==239){cl.println("S"),vaneReading="S";}    //if weight matches, print direction
             if(reading==207){cl.println("SSE"),vaneReading="SSE";}   //if weight matches, print
direction
              if(reading==223){cl.println("SE"),vaneReading="SE";}   //if weight matches, print
direction
               if(reading==159){cl.println("ESE"),vaneReading="ESE";}   //if weight matches, print
direction
                if(reading==191){cl.println("E"),vaneReading="E";}    //if weight matches, print
direction
                 if(reading==63){cl.println("ENE"),vaneReading="ENE";}    //if weight matches, print
direction
                  if(reading==127){cl.println("NE"),vaneReading="NE";}   //if weight matches, print
direction
                   if(reading==126){cl.println("NNE"),vaneReading="NNE";}   //if weight matches, print
direction

}

// sets every element of str to 0 (clears array)
void StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}

// searches for the string sfind in the string str
// returns 1 if string found
// returns 0 if string not found
char StrContains(char *str, char *sfind)
{
    char found = 0;
    char index = 0;
    char len;

    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }
    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
            }
        }
        else {
            found = 0;
        }
        index++;
    }

    return 0;
}
```