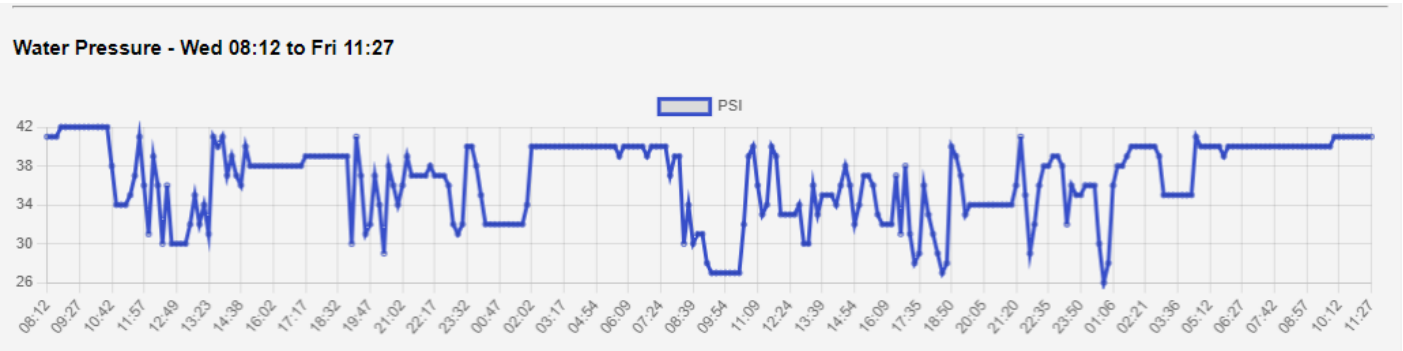
Water System Monitoring

For this project I needed a way to confirm the water pump at a remote location was functioning properly over time. The goal was to ensure the pressure is kept at a consistent 40 psi.

The solution was relatively inexpensive ($15 in parts) for an easily assembled unit using an automotive pressure sensor. The pressure sensor is wired to a ADS1115 module which then converts to voltage back to a pressure reading. The pressure reading is sent every 5 minutes to a remote server which logs the data and makes the readings available through a web page with line charts.

The server is an HP Elitedesk Pro mini pc with linux running through cloudflare (also free).

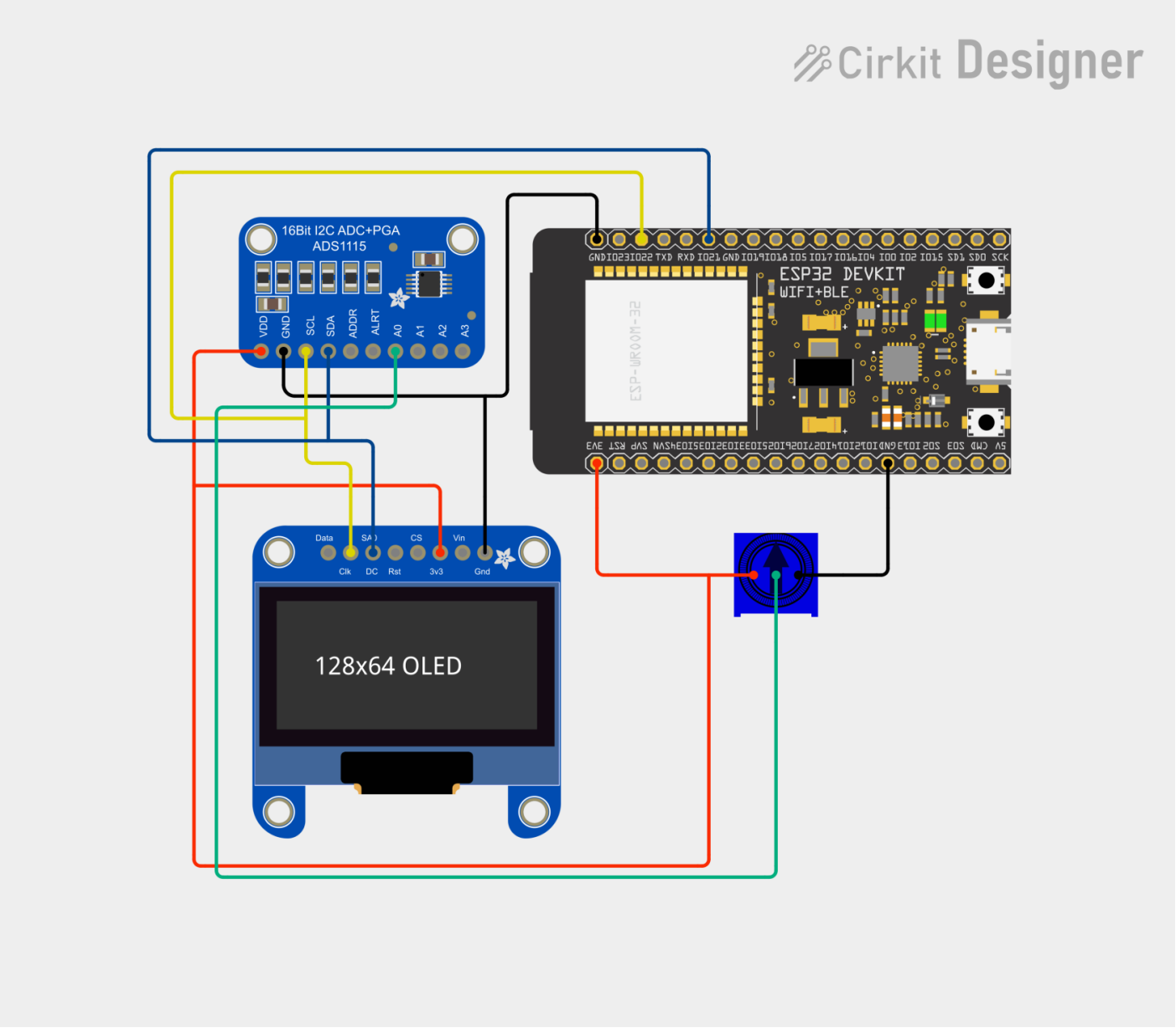
Chart on server showing pressure readings over last 2 days



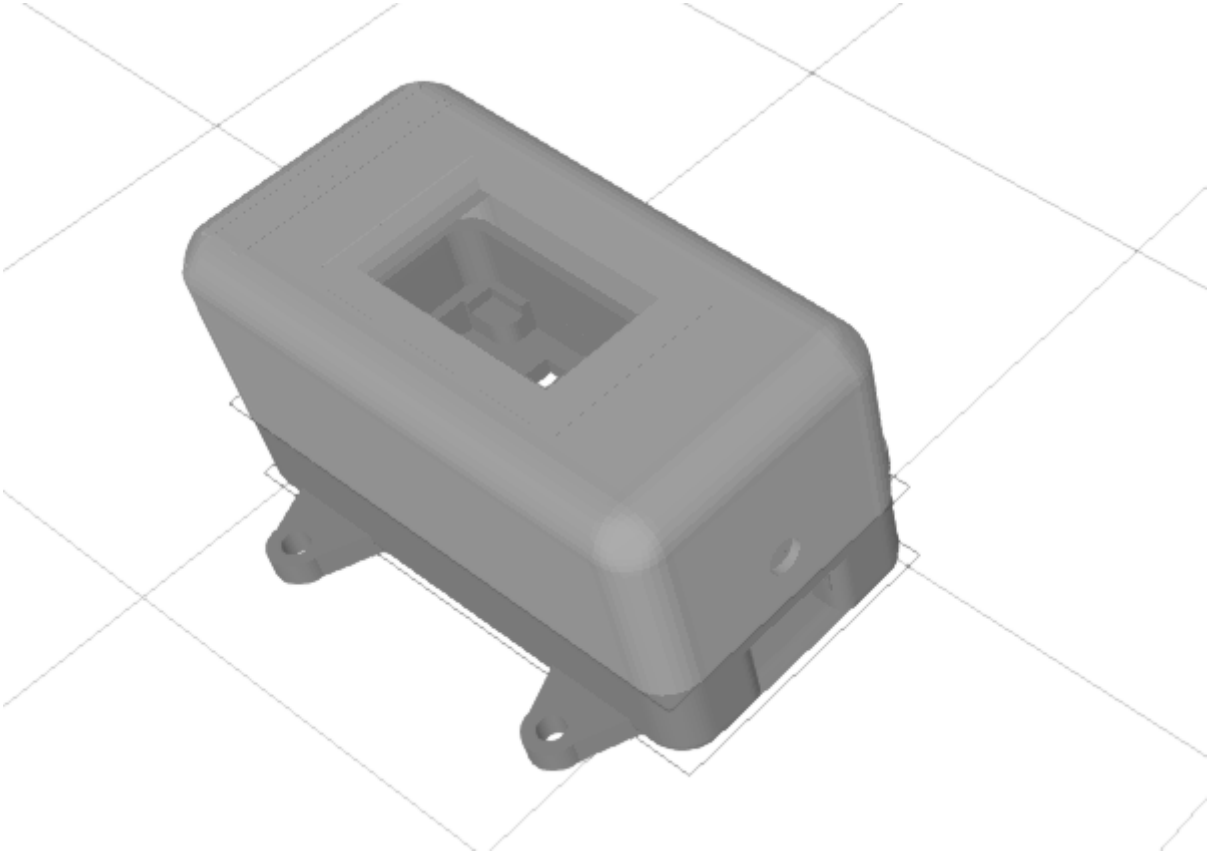
Components:

- ESP32 WROOM ($4.76 CAD) - https://www.aliexpress.com/item/4000471022528.html
- 5V 1/8NPT Pressure Transducer ($6.15 CAD) - https://www.aliexpress.com/item/1005005255271180.html
- 16 Bit I2C ADS1115 Module ($2.79 CAD) - https://www.aliexpress.com/item/32817162654.html

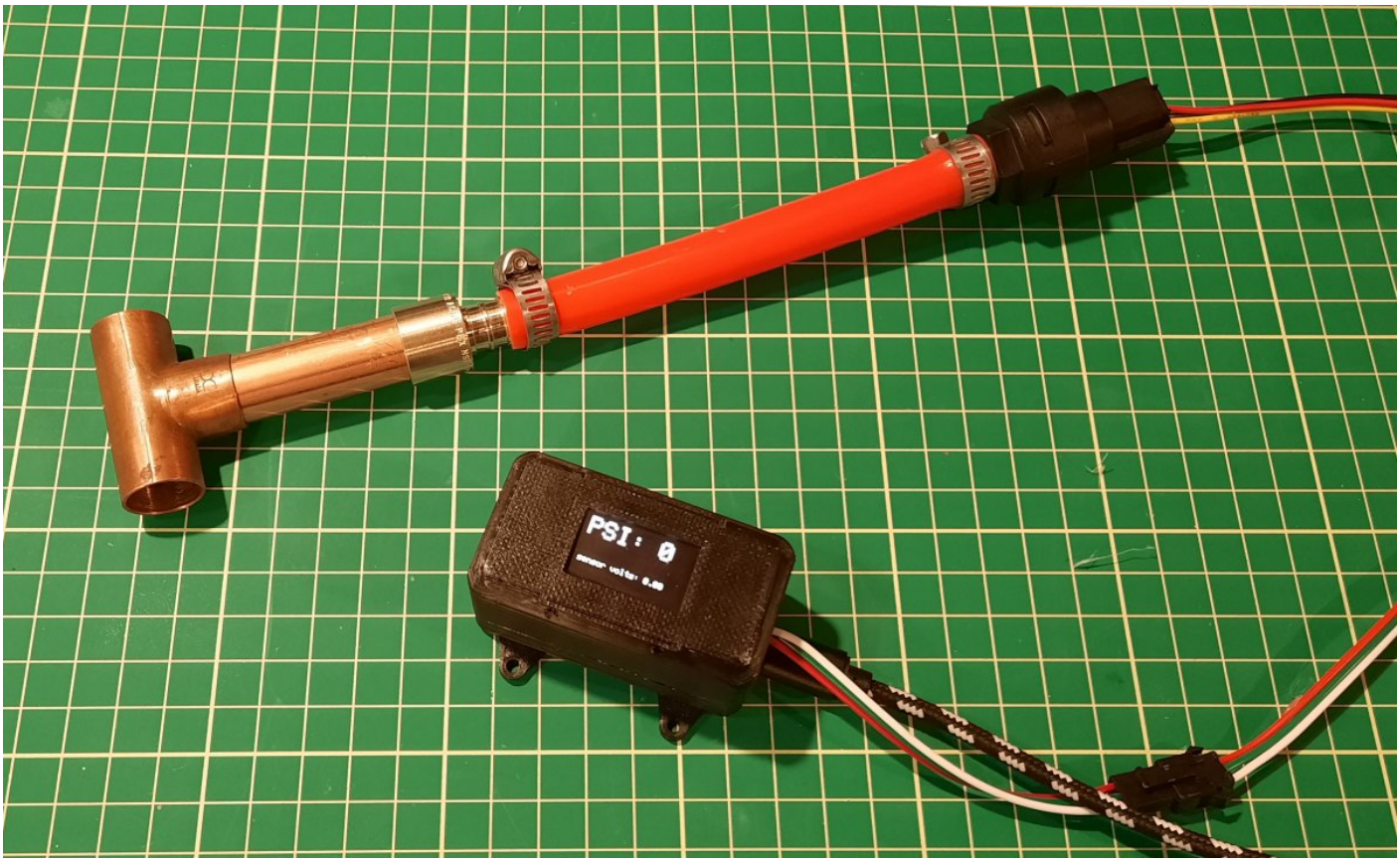
- OLED SSD1306 ($1.53 CAD) - https://www.aliexpress.com/item/32643950109.html

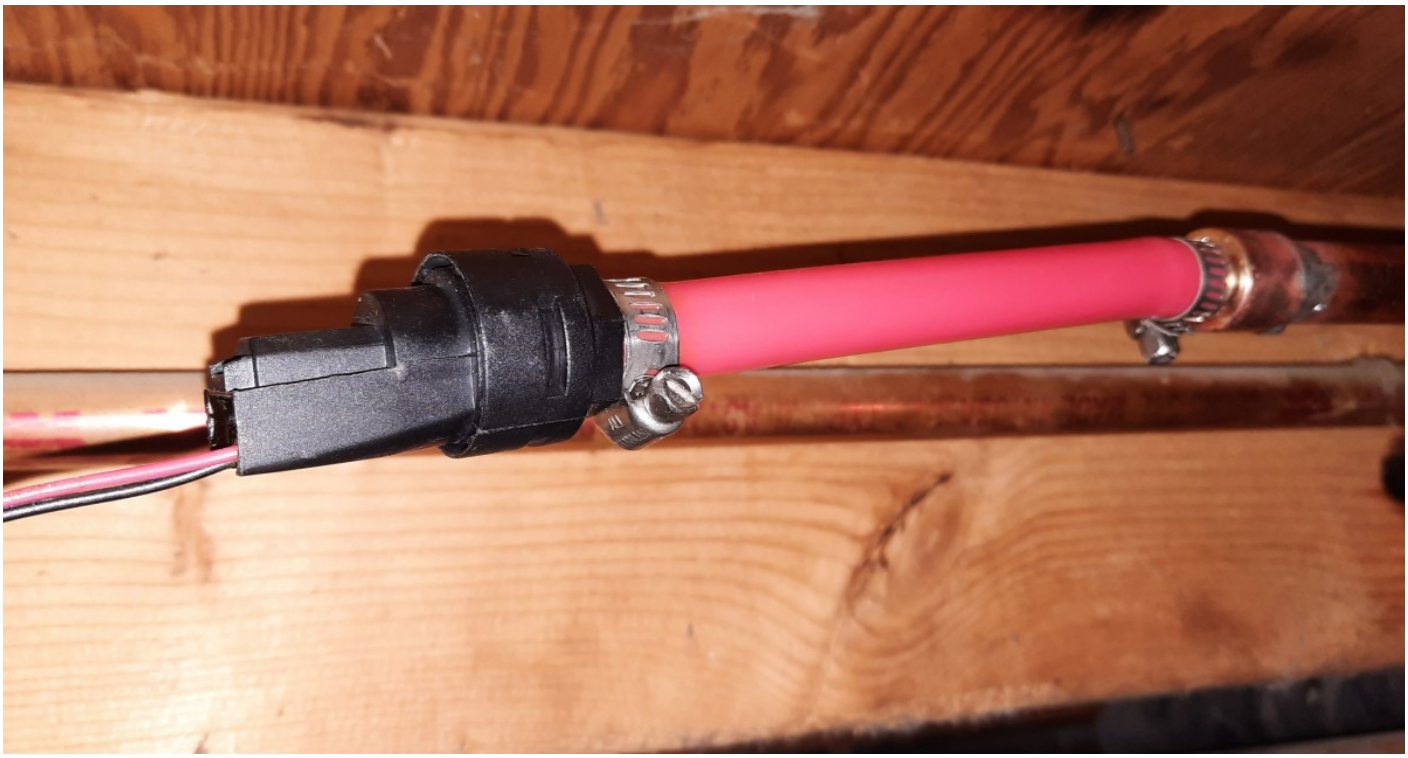


3d model for case designed for free with selfcad (www.selfcad.com)



https://www.thingiverse.com/thing:6481134





ESP32 code written in Arduino IDE (https://www.arduino.cc/en/software)

```
#include <SPI.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_ADS1X15.h>
#include <EEPROM.h>
#include "anniebitmap.h"

int activeConnection = 1;

// hansen
const String ssid1 = "xxx1";
const String password1 = "xxxxx";

// silo
const String ssid2 = "xxx2";
const String password2 = "xxxxx";

const String heartbeatUrl = "https://test.mysite.ca/silopower/heartbeat.php";
const String pressureUrl = "https://test.mysite.ca/silopower/set_water_pressure.php?data=";

#define EEPROM_SIZE 4
int eepromPingsAddress = 0;
float totalServerPings = 0;
int eepromFailedPingsAddress = 1;
float totalServerPingFails = 0;
int eepromActiveConnection = 1;

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define LOGO_HEIGHT 128
#define LOGO_WIDTH 64

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET - 1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, & Wire, OLED_RESET);

Adafruit_ADS1115 ads;
```

```
long secondsOfHour = 0;

const int secondsPerHour = 3600; // set to 60 for debugging
const int pulseRate = 1000; // loop runs once per second

const int serverSendInterval = 10; // 10 minutes between sending a voltage update to the server
const int samplesPerReading = 60 * serverSendInterval; // every x minutes send a sample to the server

int loopCount = 0;

String payload = "";
int httpCode = 0;
bool wifiConnected = false;
bool wifiPaused = false;
int wifiPausedTick = 0;
bool wifiSleeping = false;
bool debug = false; // when true server does not update
bool serverFailed = false;
int wifiConnectionAttempts = 0;

float basePressureVoltage = 0.46;
float totalledAveragePressure = 0;
float averagePressure = 0;

HTTPClient http;

void setup() {

  Serial.begin(115200);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }

  if (!ads.begin()) {
    Serial.println("Failed to initialize ADS.");
    while (1);
```

```
}

display.clearDisplay();
display.drawBitmap(0, 0, epd_bitmap_annie, 128, 64, 1);
display.display();
delay(3000);

//Init EEPROM
EEPROM.begin(EEPROM_SIZE);

activeConnection = EEPROM.read(eepromActiveConnection);

Serial.print("eepromActiveConnection :");
Serial.println(activeConnection);

if (isnan(activeConnection)) {
  activeConnection = 1;
}

if (activeConnection == 0) {
  activeConnection = 1;
}

Serial.print("activeConnection: ");
Serial.println(activeConnection);

float pingData = EEPROM.readFloat(eepromPingsAddress);
if (isnan(pingData)) {
  pingData = 0;
}
totalServerPings = pingData;
EEPROM.end();
EEPROM.begin(EEPROM_SIZE);
float pingFailData = EEPROM.readFloat(eepromFailedPingsAddress);
if (isnan(pingFailData)) {
  pingFailData = 0;
}
totalServerPingFails = pingFailData;
EEPROM.end();
```

```
  if (!connectToWiFi()) {
    delay(2000);
    WiFi.disconnect();
    delay(1000);
    if (activeConnection == 1) {
      activeConnection = 2;
    } else {
      activeConnection = 1;
    }
    connectToWiFi();
  }

  delay(2000); // Pause for 2 seconds

}

void loop() {

  secondsOfHour++;

  // after 24 hours reset this integer
  if (secondsOfHour > 86400) {
    secondsOfHour = 1;
    // Serial.println("restarting :");
    ESP.restart();
  }

  int16_t adc0;

  adc0 = ads.readADC_SingleEnded(0);
  float voltage = (adc0 * 0.1875) / 1000; //- basePressureVoltage);

  float normalizedVolatage = voltage - basePressureVoltage;

  if (normalizedVolatage < 0) {
    normalizedVolatage = 0;
  }

  float pressure = normalizedVolatage * (30 - (normalizedVolatage * 3));
```

```
if (pressure < 0) {

  pressure = 0;
}

totalledAveragePressure += pressure;

averagePressure = totalledAveragePressure / loopCount;



Serial.println("averagePressure: ");
Serial.println(averagePressure);



display.clearDisplay();
display.setTextSize(3);
display.setTextColor(WHITE);
display.setCursor(0, 0);
display.print("PSI: ");
display.println(pressure, 0);
display.println("");
display.setTextSize(1);
display.print("sensor volts: ");
display.println(normalizedVolatage, 2);
display.display();

//////////////////////
// SERVER RELAY

if (!debug && (loopCount >= samplesPerReading) || serverFailed) {

  loopCount = 0;
  totalledAveragePressure = 0;

  setWifiSleepMode(false);

  delay(2000);

  // do a heartbeat check to see if we are online...
  http.begin(heartbeatUrl);
```

```
    httpCode = http.GET();
    if (!httpCode > 0) {
      // wifi may not be alive yet so wait 3 seconds
      delay(3000);
    }

    String recordedPressure = String(averagePressure, 1);

    recordedPressure.trim();

    loopCount = 0;

    http.begin(pressureUrl + recordedPressure);
    httpCode = http.GET();
    if (httpCode > 0) {
      payload = http.getString();
      Serial.println("HTTP Response: " + payload);
      recordPingSucces();
    } else {
      recordPingFailure();
    }
    http.end();

    setWifiSleepMode(true);
  }

  loopCount++;

  delay(1000);

}

bool connectToWiFi() {

  String activeSsid = "";
  String activePassword = "";

  if (activeConnection == 1) {
    activeSsid = ssid1;
    activePassword = password1;
```

```
} else if (activeConnection == 2) {
  activeSsid = ssid2;
  activePassword = password2;
}

Serial.print("Connecting to WiFi: ");
Serial.println(activeSsid);

WiFi.begin(activeSsid, activePassword);

while (WiFi.status() != WL_CONNECTED && wifiConnectionAttempts < 20) {
  delay(500);
  Serial.print(".");
  wifiConnectionAttempts++;
}

wifiConnectionAttempts = 0;

if (WiFi.status() == WL_CONNECTED) {
  Serial.println("\nConnected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  wifiConnected = true;

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.write(eepromActiveConnection, activeConnection);
  EEPROM.commit();
  EEPROM.end();

  Serial.print("set activeConnection to : ");
  Serial.println(activeConnection);

  return true;

} else {
  Serial.print("Connection to ");
  Serial.print(activeSsid);
  Serial.println(" failed. Trying alternative");
  return false;
}
```

```
}

/**
 * set wifi sleep mode between data relays to conserve energy
 * @param sleepMode - if true set wifi card to sleep to conserve energy
 */
void setWifiSleepMode(bool sleepMode) {

  wifiSleeping = sleepMode;

  if (sleepMode) {
    WiFi.disconnect();
    WiFi.setSleep(true);
    delay(1000);
    Serial.print("sleep wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
  } else {
    WiFi.setSleep(false);
    WiFi.reconnect();
    delay(1000);
    Serial.print("awaken wifi status: ");
    Serial.println(wl_status_to_string(WiFi.status()));
    // Check if the connection is still active. if not trigger wait for it to come back online
    if (WiFi.status() != WL_CONNECTED && !wifiPaused) {
      Serial.println("Connection lost. Attempting to reconnect in 1 minute ...");
      WiFi.disconnect();
      wifiPaused = true;
      wifiConnected = false;
      connectToWiFi();
    }
  }
}

/**
 * record server ping success in long term memory
 */
void recordPingSucces() {
  totalServerPings++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromPingsAddress, totalServerPings);
```

```
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = true;
  serverFailed = false;
}

/**
 * record server ping fails in long term memory
 */
void recordPingFailure() {
  totalServerPingFails++;
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.writeFloat(eepromFailedPingsAddress, totalServerPingFails);
  EEPROM.commit();
  EEPROM.end();
  wifiConnected = false;
  serverFailed = true;
}

/**
 * ESP32 wifi card statuses
 * @param status
 * @return string
 */
String wl_status_to_string(wl_status_t status) {

  String response = "";

  switch (status) {
  case WL_NO_SHIELD:
    response = "WL_NO_SHIELD";
    break;
  case WL_IDLE_STATUS:
    response = "WL_IDLE_STATUS";
    break;
  case WL_NO_SSID_AVAIL:
    response = "WL_NO_SSID_AVAIL";
    break;
  case WL_SCAN_COMPLETED:
    response = "WL_SCAN_COMPLETED";
```

```
    break;
  case WL_CONNECTED:
    response = "WL_CONNECTED";
    break;
  case WL_CONNECT_FAILED:
    response = "WL_CONNECT_FAILED";
    break;
  case WL_CONNECTION_LOST:
    response = "WL_CONNECTION_LOST";
    break;
  case WL_DISCONNECTED:
    response = "WL_DISCONNECTED";
    break;
  }

  return response;
}
```

Revision #38
Created 10 February 2024 00:50:38 by peter drinnan
Updated 23 June 2024 16:21:18 by peterd