Water System Monitoring

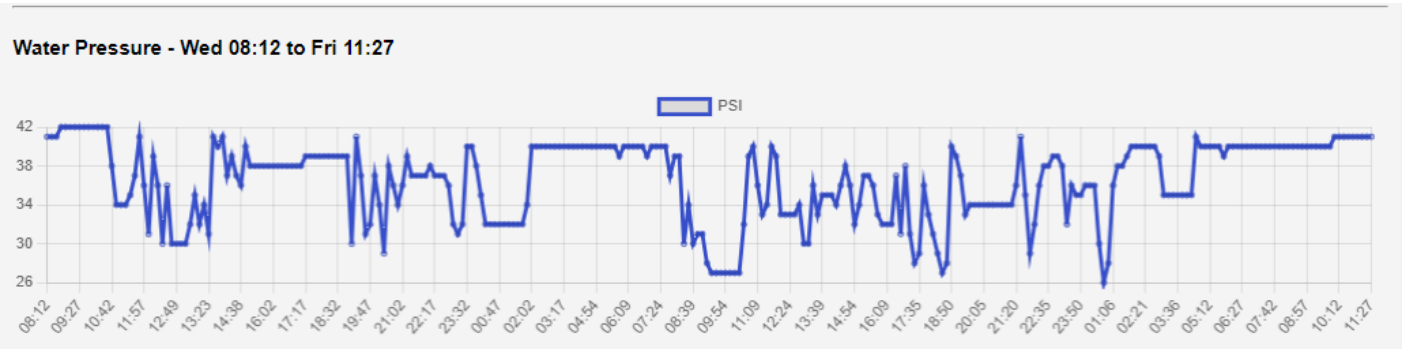


For this project I needed a way to confirm the water pump at a remote location was functioning properly over time. The goal was to ensure the pressure is kept at a consistent 40 psi.

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).

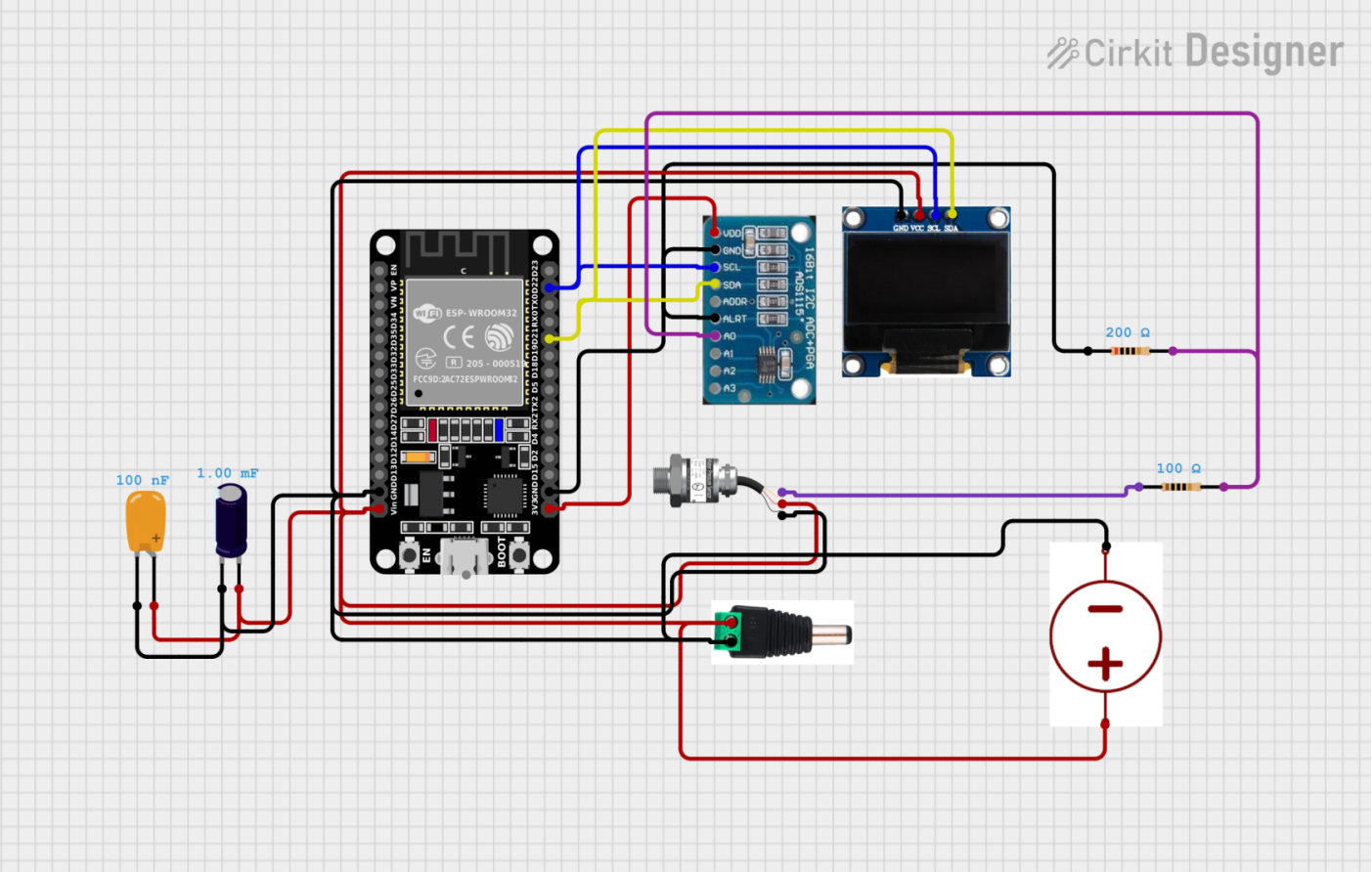
Chart on server showing pressure readings over last 2 days

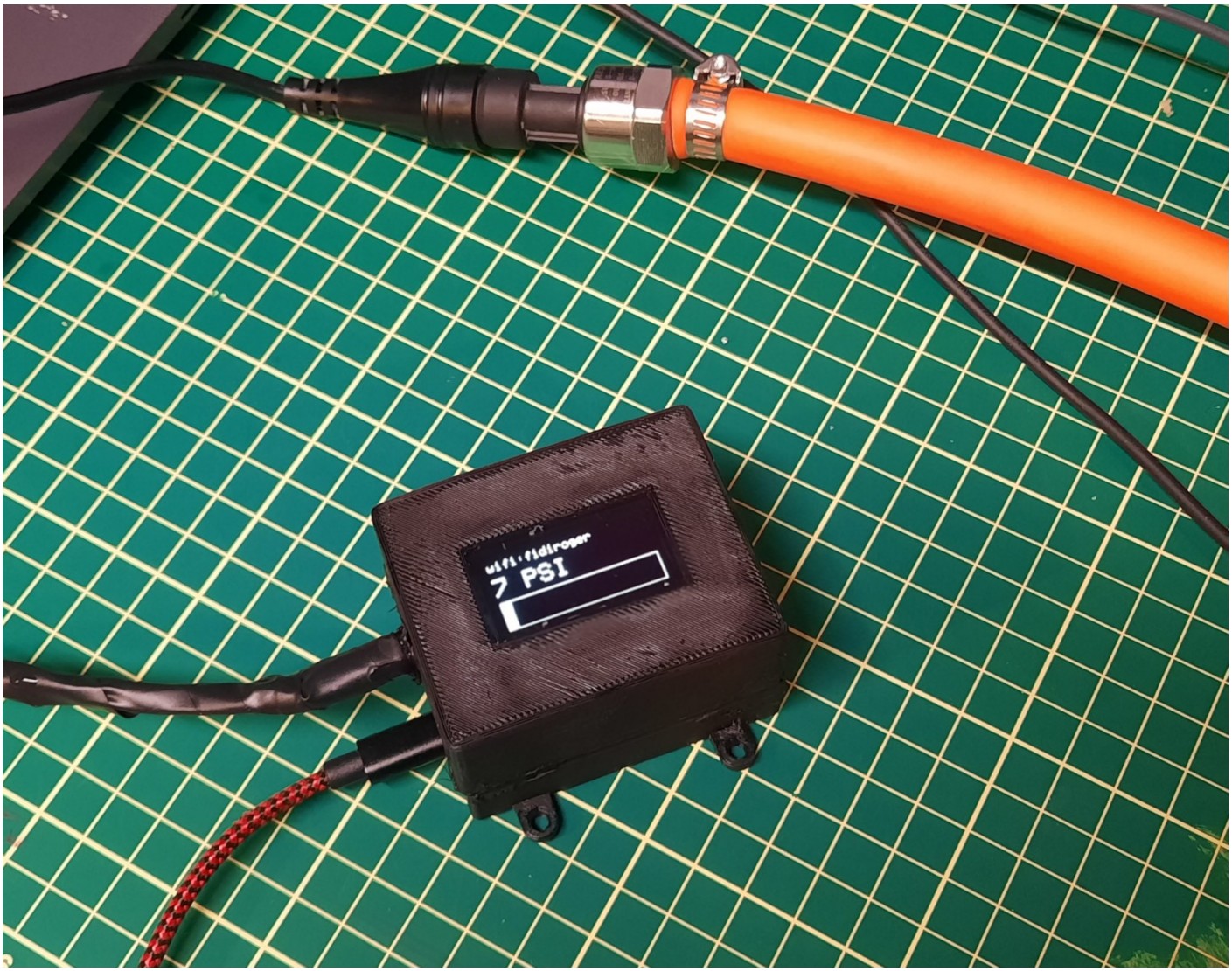


Components:

- 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





ESP32 code written in Platform IO

```
#include <Wire.h>
#include <Adafruit_ADS1X15.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#include <SPI.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>
#include <EEPROM.h>

int activeConnection = 1;
String activeSSID = "";
```

```
// Alternate display info
unsigned long lastInfoSwitch = 0;
bool showWifiInfo = true;

// hansen
const String ssid1 = "xxxx";
const String password1 = "xxx";

// silo
const String ssid2 = "xxxx";
const String password2 = "xxxx";

const String heartbeatUrl = "https://xxxx.xxxx.ca/silopower/heartbeat.php";
const String pressureUrl = "https://xxxx.xxxx.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;

const int pulseRate = 1000; // loop runs once per second

const int serverSendInterval = 10;                    // 10 minutes between sending a
pressure 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;
```

```
HTTPClient https;
WiFiClientSecure client;

// --- Display setup ---
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SH1106G display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// --- ADS1115 setup ---
Adafruit_ADS1115 ads;

// --- Voltage divider ratio ---
// You said 5V → 3.4V at A0, so gain = 3.4/5.0 = 0.68
const float dividerGain = 3.4f / 5.0f; // ≈0.68

// Sensor parameters (0.5–4.5 V = 0–100 psi)
const float sensorMinV = 0.5;
const float sensorMaxV = 4.5;
const float psiMax = 100.0;

// Burst sample settings
const int NUM_SAMPLES = 50;
const float TRIM_FRACTION = 0.20f;
const float ALPHA = 0.2f; // EMA smoothing

float psiFiltered = 0.0f;

float readPsiTrimmed();

void setWifiSleepMode(bool sleepMode);

bool connectToWiFi();

String wl_status_to_string(wl_status_t status);

// --- Zero calibration ---
float zeroOffset = 0.0f;
```

```
void calibrateZero()
{
  float psiNow = readPsiTrimmed();
  zeroOffset = psiNow; // store offset
  Serial.print("Zero calibrated at: ");
  Serial.println(zeroOffset, 2);
}

float psiCorrected(float psiRaw)
{
  float psi = psiRaw - zeroOffset;
  if (psi < 0)
    psi = 0;
  if (psi > psiMax)
    psi = psiMax;
  return psi;
}

// --- Setup ---
void setup()
{
  Serial.begin(115200);
  Wire.begin(21, 22);

  delay(5000);

  Serial.println("Starting PSI Sensor...");
  // Init display
  if (!display.begin(0x3C, true))
  {
    Serial.println("SH1106G not found");
    while (1)
      ;
  }
  display.clearDisplay();
  display.setTextColor(SH110X_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 20);
  display.println("PSI Sensor Init...");
  display.display();
```

```
// Init ADS1115
if (!ads.begin(0x48))
{
  Serial.println("ADS1115 not found");
  while (1)
    ;
}
ads.setGain(GAIN_ONE); // ±4.096 V range
ads.setDataRate(RATE_ADS1115_128SPS);

display.setCursor(0, 40);
display.println("ADS1115 Init...");
display.display();

delay(1000);

// Auto-zero on startup (sensor vented to atmosphere)
calibrateZero();

// 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;
}

if (activeConnection > 2)
{
```

```
  activeConnection = 2;
}
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;
  }
  // Try a second time
  if (!connectToWiFi()) {
    delay(2000);
    WiFi.disconnect();
    delay(1000);
    // Optionally, print a message or handle failure here
    Serial.println("WiFi connection failed after two attempts.");
  }
}
```

```
  delay(2000); // Pause for 2 seconds
}

// --- Conversion helpers ---
float countsToVolts(int16_t counts) { return ads.computeVolts(counts); }

float voltsToPsi(float v_ads)
{
  float v_sensor = v_ads / dividerGain; // undo divider (÷0.68)
  float psi = (v_sensor - sensorMinV) * (psiMax / (sensorMaxV - sensorMinV));
  if (psi < 0)
    psi = 0;
  if (psi > psiMax)
    psi = psiMax;
  return psi;
}

// --- Robust read with trimmed mean ---
float readPsiTrimmed()
{
  static int16_t buf[NUM_SAMPLES];

  ads.setDataRate(RATE_ADS1115_860SPS); // fast burst

  for (int i = 0; i < NUM_SAMPLES; i++)
  {
    buf[i] = ads.readADC_SingleEnded(0);
    delayMicroseconds(400);
  }

  ads.setDataRate(RATE_ADS1115_128SPS);

  // Sort samples
  for (int i = 1; i < NUM_SAMPLES; i++)
  {
    int16_t key = buf[i];
    int j = i - 1;
    while (j >= 0 && buf[j] > key)
    {
      buf[j + 1] = buf[j];
```

```
      j--;
    }
    buf[j + 1] = key;
  }

  // Trim extremes
  int trim = (int)(NUM_SAMPLES * TRIM_FRACTION);
  if (trim * 2 >= NUM_SAMPLES)
    trim = (NUM_SAMPLES - 1) / 2;
  long sum = 0;
  int count = 0;
  for (int i = trim; i < NUM_SAMPLES - trim; i++)
  {
    sum += buf[i];
    count++;
  }
  float avgCounts = (float)sum / (float)count;

  float v_ads = countsToVolts((int16_t)avgCounts);
  return voltsToPsi(v_ads);
}

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);

  activeSSID = activeSsid;

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

  activeSSID = "";
  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;
}

// --- Main loop ---
void loop()
{

  int16_t adc0 = ads.readADC_SingleEnded(0);

  // Convert to volts using library helper
  float volts = ads.computeVolts(adc0);

  // Serial.print("A0 Counts: ");
  // Serial.print(adc0);
  // Serial.print(" | Voltage: ");
```

```
// Serial.print(volts, 4);
// Serial.println(" V");

float psiRaw = readPsiTrimmed();
float psi = psiCorrected(psiRaw);

// EMA smoothing
psiFiltered = ALPHA * psi + (1.0f - ALPHA) * psiFiltered;

// Serial
// Serial.print("Raw PSI: "); Serial.print(psiRaw, 2);
// Serial.print(" | Corrected PSI: "); Serial.print(psi, 2);
// Serial.print(" | Smoothed: "); Serial.println(psiFiltered, 2);

// OLED
display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 6);

// Alternate info every 5 seconds
if (millis() - lastInfoSwitch > 5000)
{
  showWifiInfo = !showWifiInfo;
  lastInfoSwitch = millis();
}

// Alternate info every 10 seconds
if ((loopCount / 10) % 2 == 0)
{
  display.println("wifi:" + activeSSID);
}
else
{
  display.println("total pings:" + String((int)totalServerPings));
}
display.setTextSize(2);
display.setCursor(0, 20);
display.print((int)(psiFiltered + 0.5f));
display.print(" PSI");
```

```
// Bar graph
int barW = map((int)(psiFiltered + 0.5f), 0, (int)psiMax, 0, 126);
display.drawRect(0, 40, 126, 24, SH110X_WHITE);
display.fillRect(0, 40, barW, 24, SH110X_WHITE);

display.display();

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

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

  loopCount = 0;

  setWifiSleepMode(false);

  delay(2000);

  // do a heartbeat check to see if we are online...
  https.begin(client, heartbeatUrl);
  httpCode = https.GET();
  if (!httpCode > 0)
  {
    // wifi may not be alive yet so wait 3 seconds
    delay(3000);
  }

  String recordedPressure = String(psiFiltered, 1);

  recordedPressure.trim();

  loopCount = 0;

  client.setInsecure();

  https.begin(client, pressureUrl + recordedPressure);
  httpCode = https.GET();
  if (httpCode > 0)
  {
```

```
      payload = https.getString();
      Serial.println("HTTP Response: " + payload);
      recordPingSucces();

      display.clearDisplay();
      display.setTextSize(1);
      display.setCursor(0, 10);
      display.println("Pressure relayed");
      display.display();
    }
    else
    {
      recordPingFailure();
    }
    https.end();

    setWifiSleepMode(true);
  }

  loopCount++;

  delay(1000);
}
```

Revision #39
Created 2024-02-10 00:50:38 UTC by peter drinnan
Updated 2025-10-11 11:23:33 UTC by peterd