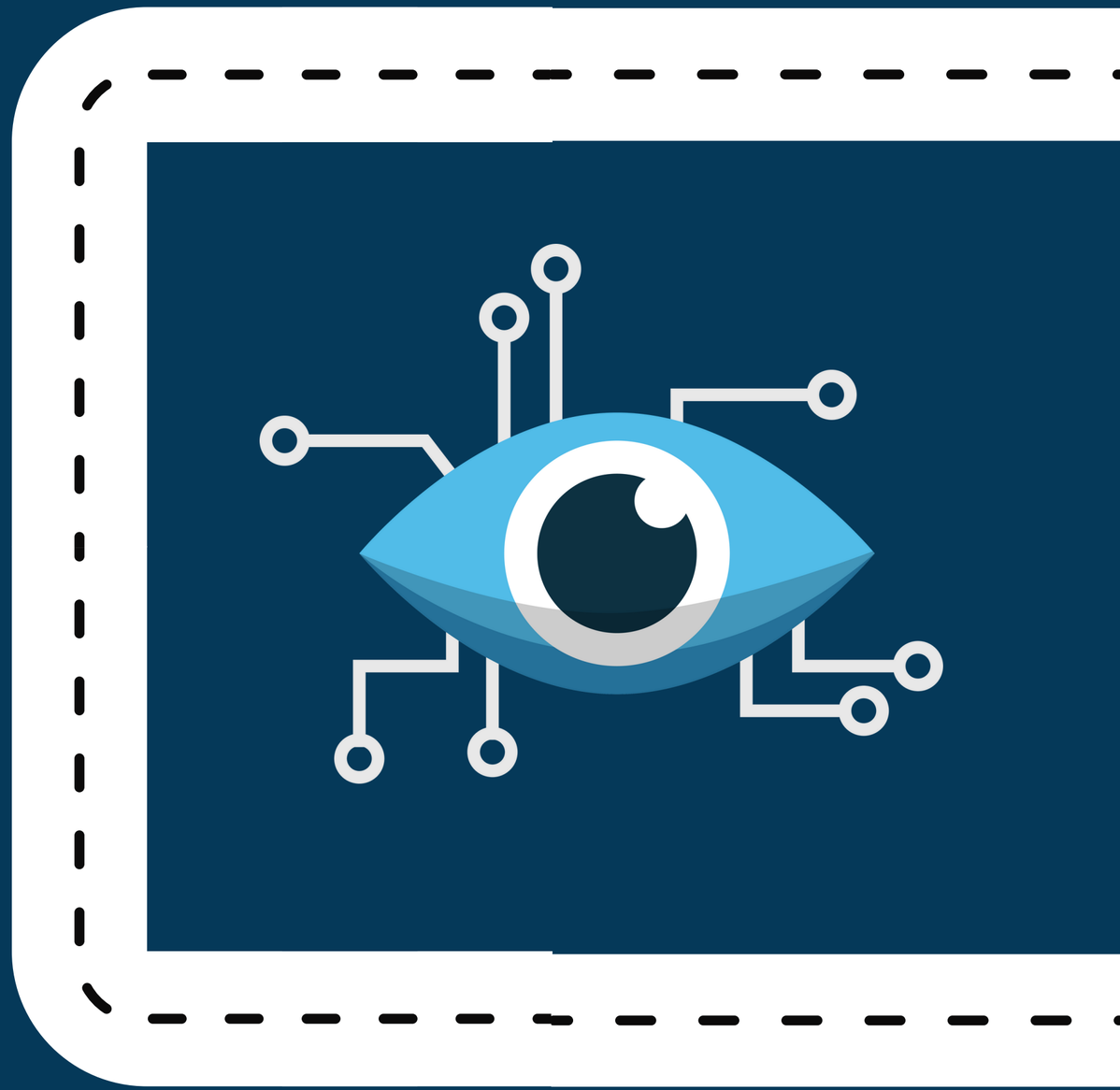
# Real-Time

# Air Quality Monitoring System

Tech Stack : C# · JSON · TCP/IP Socket · Multi-threading



신선호

# 목차


1 프로젝트 개요

2 Flowchart

3 Protocol

4 UI 및 주요코드

5 실행결과

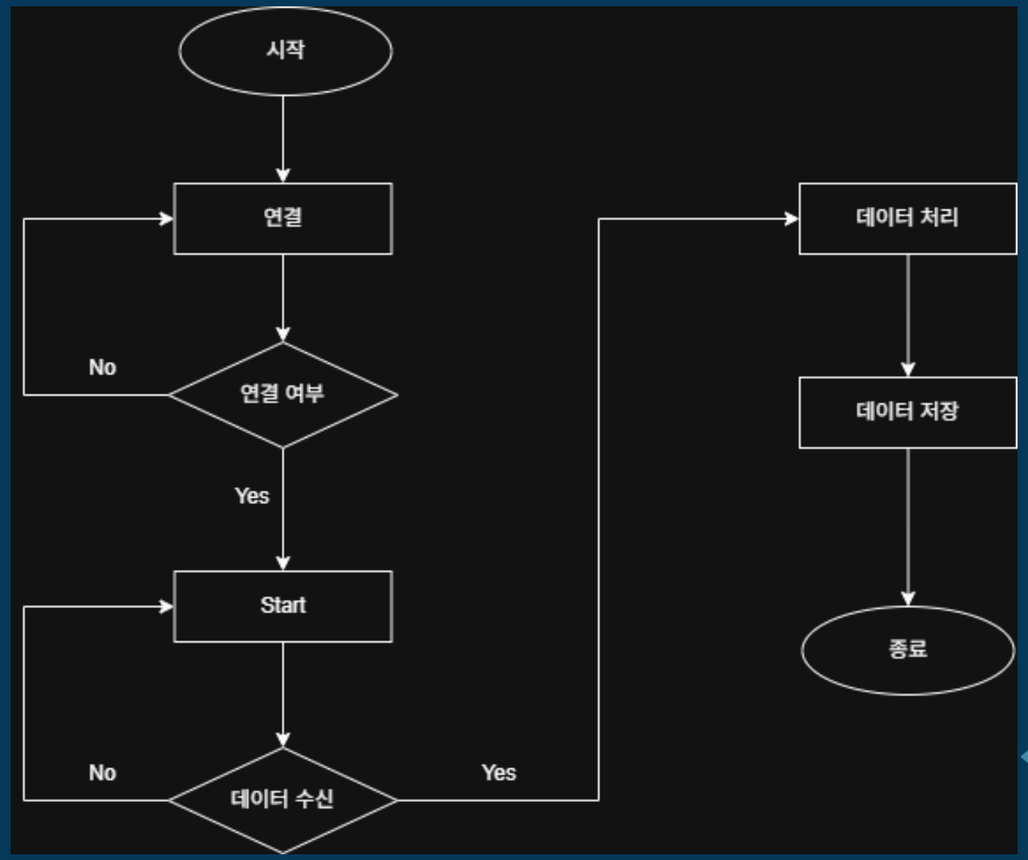
# 1. 프로젝트 개요

본 프로젝트는 실제 센서(H/W) 대신
서버에서 생성한 더미 데이터를
TCP/IP로 수신하여
실시간 데이터 처리/모니터링 구조를
구현한 프로젝트입니다.

수신된 온도/습도/산소/이산화탄소
미세먼지/초미세먼지 데이터를
1분 단위 평균값으로 처리하여
UI에 표시하고 CSV 파일로 자동 저장합니다.

멀티스레딩을 적용하여 데이터
수신과 UI 처리를 분리해
실시간성과 안정성을 확보했습니다.

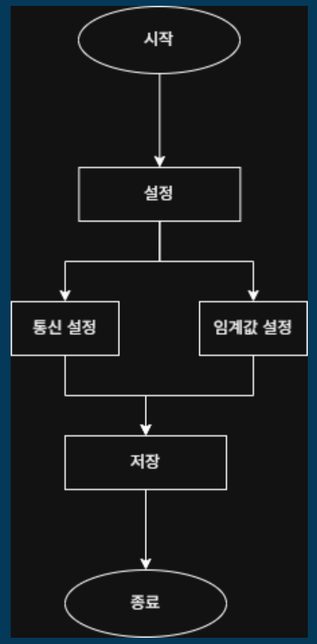
# 2. Flowchart



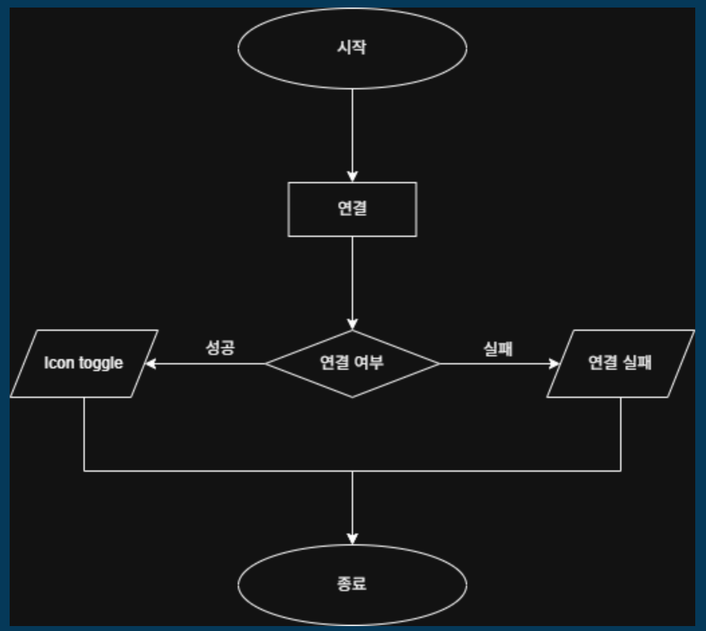

본 시스템은 연결 → 데이터 수신 → 처리 → 저장의 단계로 구성되어 있으며, 각 기능은 독립적인 흐름으로 설계되어 프로그램의 안정성을 높였습니다.

전체

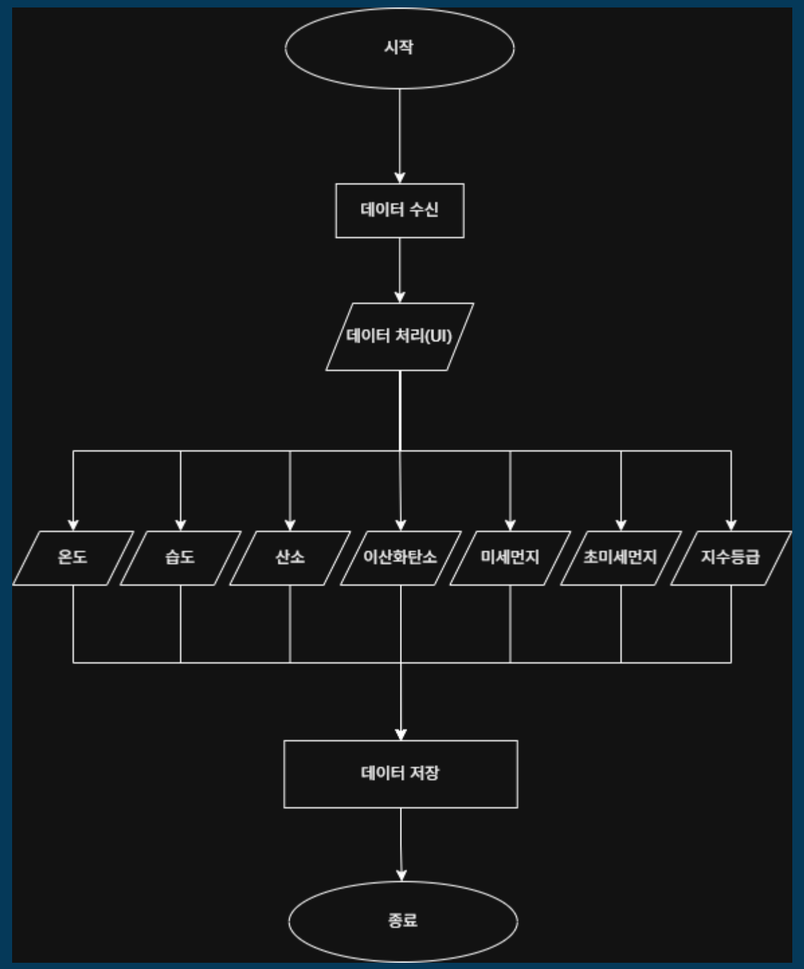
# 2. Flowchart




설정부




연결부




데이터 수신부

# 3. Protocol

## Protocol 규칙

- 기본적으로 요청/응답 방식으로 동작하며
  Event가 발생한 경우에는 요청없이 즉시 응답한다.

- 통신 방식은 TCP 기반이다.

- Packet은 STX + DATA + ETX 로 구성한다.

- STX는 0x02, ETX는 0x03 이다.

- 클라이언트 요청(Request)에 대해
  서버가 응답(Response)하는 방식으로 동작한다.

## COMMAND

| | |
|---|---|
| KICK | 서버에 데이터 수신 시작을 요청 |
| Get_Data | 실시간 센서 데이터 전송을 요청 |
| STOP | 서버의 데이터 송신 중지 및 세션 종료를 요청 |

### 데이터 수신시작

| SENDER | FORMAT |
|---|---|
| HOST | KICK |
| Server | ACK |

### 데이터 송신 중지

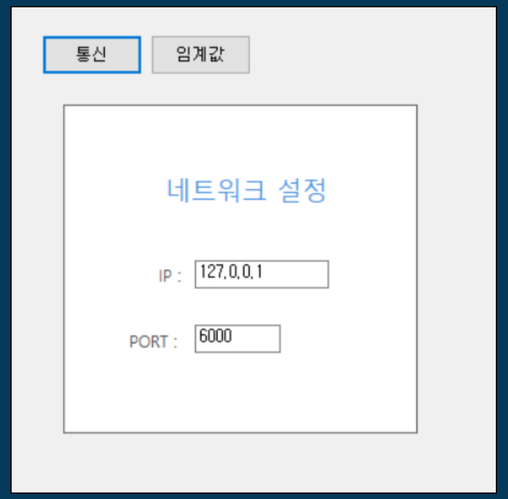
| SENDER | FORMAT |
|---|---|
| HOST | STOP |
| Server | STOP_ACK |

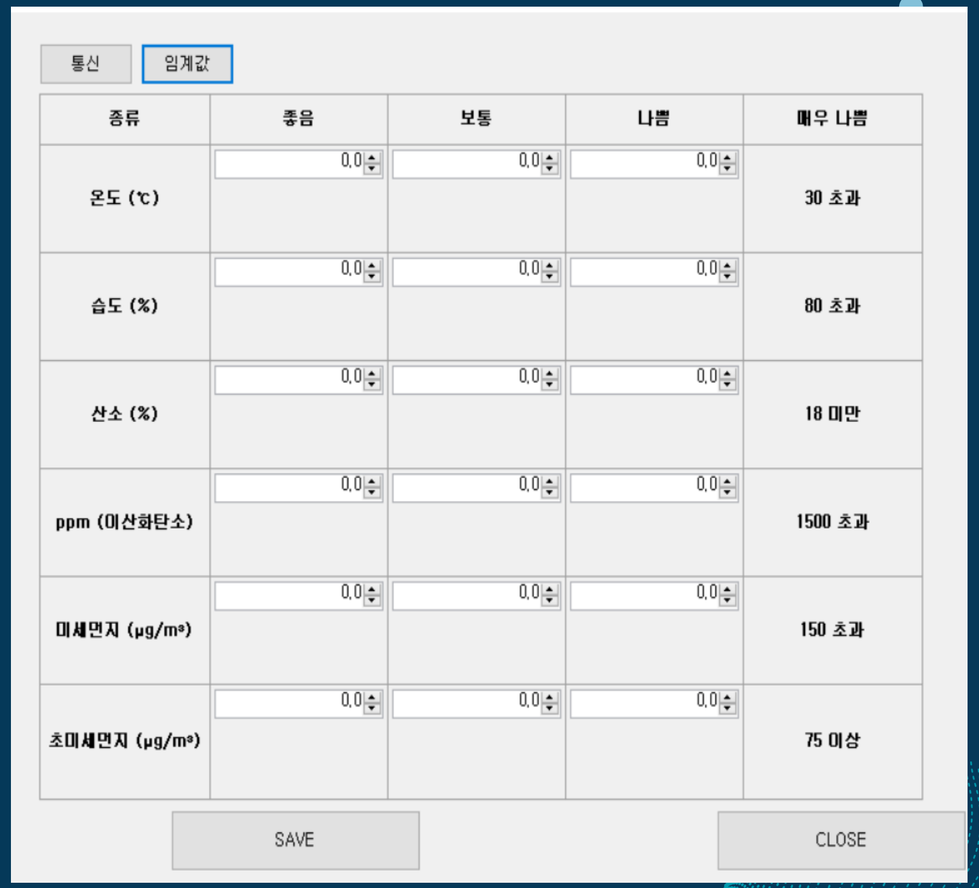
### 센서 데이터 수집

| SENDER | FORMAT |
|---|---|
| HOST | Get_Data |
| Server | DATA,Time,Temp,Hum,O2,CO2,PM10,PM25 |

# 4. UI 및 주요코드

## 4-1. UI

# 4. UI 및 주요코드

**4-2.**
주요코드

```
참조 1개
public void Connect(string host, int port)
{
    if (state == ClientState.Connected || state == ClientState.Connecting)
    {
        return;
    }

    Host = host;
    Port = port;
    state = ClientState.Connecting;

    if (workerThread == null || !workerThread.IsAlive)
    {
        workerThread = new Thread(RunWorker);
        workerThread.IsBackground = true;
        workerThread.Start();
    }
}
```

```
참조 1개
private void RunWorker()
{
    while (state != ClientState.Finished)
    {
        Thread.Sleep(10);

        switch (state)
        {
            case ClientState.Connecting:
                TryConnect();
                break;

            case ClientState.Connected:
                if (socket == null || !socket.Connected)
                    state = ClientState.Disconnecting;
                break;

            case ClientState.Disconnecting:
                TryDisconnect();
                break;

            case ClientState.Disconnected:
                state = ClientState.Finished;
                break;

            case ClientState.Finished:
                return;
        }
    }
}
```

```
참조 1개
private void TryConnect()
{
    try
    {
        if (socket != null)
        {
            try { socket.Close(); }catch { }
        }

        socket = new System.Net.Sockets.Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        var ep = new IPEndPoint(IPAddress.Parse(Host), Port);

        var result = socket.BeginConnect(ep, null, null);
        bool success = result.AsyncWaitHandle.WaitOne(2000, false);

        if (!success || !socket.Connected)
        {
            throw new Exception("연결 시간 초과 또는 서버 응답 없음");
        }

        socket.EndConnect(result);
        state = ClientState.Connected;
        OnStatusChanged?.Invoke($"서버({Host}:{Port}) 연결됨");

        receiveThread = new Thread(ReceiveLoop);
        receiveThread.IsBackground = true;
        receiveThread.Start();
    }
    catch (Exception ex)
    {
        OnStatusChanged?.Invoke($"연결 실패: {ex.Message}");
        state = ClientState.Disconnected;

        state = ClientState.Ready;
    }
}
```

**TCP/IP Socket** 기반 **Client** 연결을 상태**(State)** 기반으로 관리하여 연결/해제 흐름을 안정적으로 제어

**BeginConnect + Timeout(2초)** 적용으로 서버 응답 지연 상황에서도 프로그램 멈춤 없이 연결 실패 처리 가능

별도 **Worker Thread**를 통해 연결 상태를 지속적으로 감시하고 자동으로 상태 전환 수행

# 4. UI 및 주요코드

**4-2.**
주요코드

```
참조 1개
private void ReceiveLoop()
{
    try
    {
        byte[] buffer = new byte[4096];

        while (IsConnected)
        {
            int n = socket.Receive(buffer);
            if (n <= 0) break;

            receiveBuffer.Append(Encoding.UTF8.GetString(buffer, 0, n));

            ProcessReceivedData();
        }
    }
    catch (Exception)
    {
        if (IsConnected) OnStatusChanged?.Invoke("서버 연결 끊김");
    }
    finally
    {
        if (IsConnected) Disconnect();
    }
}
```

```
참조 1개
private void ProcessReceivedData()
{
    while (true)
    {
        string data = receiveBuffer.ToString();
        int stx = data.IndexOf((char)STX);
        int etx = data.IndexOf((char)ETX);

        if (stx >= 0 && etx > stx)
        {
            string msg = data.Substring(stx + 1, etx - stx - 1);
            receiveBuffer.Remove(0, etx + 1);
            OnMessageReceived?.Invoke(msg);
        }
        else
        {
            break;
        }
    }
}
```

별도 수신 **Thread**에서 서버 데이터를 지속적으로 **Receive**하여 **UI** 프리징 없이 실시간 데이터 수신 가능

**STX/ETX** 기반 프로토콜을 적용하여 수신 **Buffer**에 누적된 데이터를 정상 패킷 단위로 분리 처리

데이터가 분절되어 수신되는 상황에서도 안정적으로 메시지를 재조합하여 처리 가능

# 4. UI 및 주요코드

**4-2.**
주요코드

```
참조 1개
private void HandleServerMessage(string msg)
{
    MessageDisplay($"[수신] {msg}");

    if (msg == "ACK")
    {
        client.SendMessage("Get_Data");
        MessageDisplay("[송신] Get_Data");
    }
    else if (msg == "STOP_ACK")
    {
        MessageDisplay("서버 데이터 송신 중지 확인");
    }
    else if (msg.StartsWith("DATA") && isRunning)
    {
        ProcessSensorData(msg);
    }
}
```

서버 응답**(ACK, STOP_ACK, DATA)**을 기반으로 동작하는 메시지 프로토콜 구조 구현

**KICK → ACK → Get_Data → DATA** 수신 흐름으로 데이터 요청/수신 절차를 명확하게 구성

수신 메시지에 따라 제어 흐름을 분기하여 장비 통신 프로그램의 안정성과 유지보수성을 확보

# 4. UI 및 주요코드

**4-2.**
주요코드

```
참조 1개
private void ProcessSensorData(string msg)
{
    var sp = msg.Split(',');

    var data = new SensorData
    {
        Time = DateTime.Now,
        Temp = double.Parse(sp[2]),
        Hum = double.Parse(sp[3]),
        O2 = double.Parse(sp[4]),
        CO2 = double.Parse(sp[5]),
        PM10 = double.Parse(sp[6]),
        PM25 = double.Parse(sp[7])
    };

    buffer.Add(data);

    if (!DataDisplayed)
    {
        var instant = new DataAverage
        {
            Temp = data.Temp,
            Hum = data.Hum,
            O2 = data.O2,
            CO2 = data.CO2,
            PM10 = data.PM10,
            PM25 = data.PM25
        };

        indexgrade.UpdateAverage(instant);
        UpdateLastUpdateTime(DateTime.Now);

        DataDisplayed = true;
        lastDisplayedTime = DateTime.Now;
        lastSaveTime = DateTime.Now;

        return;
    }

    if ((DateTime.Now - lastSaveTime).TotalMinutes >= 1)
    {
        SaveAverage();
        buffer.Clear();
        lastSaveTime = DateTime.Now;
    }
}
```

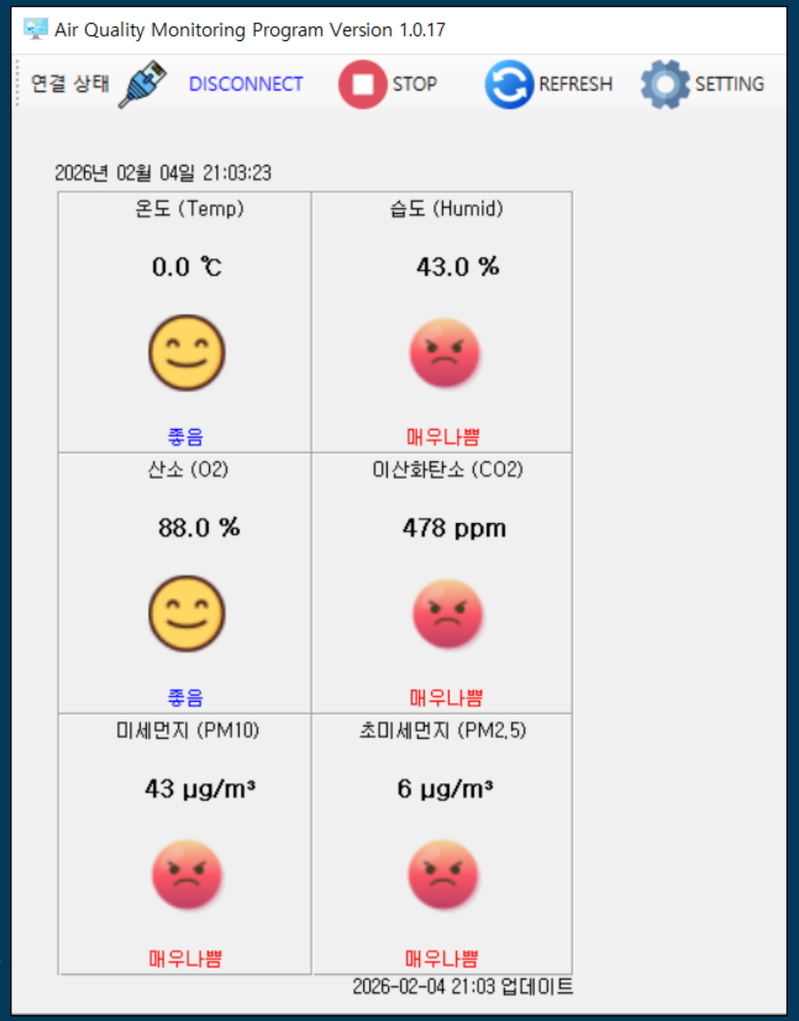
수신된 센서 데이터를 파싱하여 실시간으로 **List Buffer**에 누적 관리

**1**분 단위 평균값 계산을 위해 일정 시간 동안 데이터를 저장하고 평균 처리 로직 수행

최초 수신 시 즉시 **UI** 표시 후, 이후에는 평균값 기반 갱신 방식으로 데이터 모니터링 최적화

# 5. 실행결과

**UI**




서버로부터 수신된 공기질 데이터를 실시간으로 **UI**에 표시하며,

항목별 등급(좋음~매우나쁨)을 시각적으로 확인할 수 있습니다.

**Refresh** 기능을 통해 최신 평균값을 즉시 반영할 수 있도록 구현했습니다.

# 5. 실행결과

## Log

```
2026-02-04 20:52:47.743         Program Open
2026-02-04 21:03:13.978         서버(127.0.0.1:6000) 연결됨
2026-02-04 21:03:15.594         [송신] KICK
2026-02-04 21:03:15.604         [수신] ACK
2026-02-04 21:03:15.605         [송신] Get_Data
2026-02-04 21:03:15.624         [수신] DATA,21:03:15,0,43,88,478,43,6
2026-02-04 21:03:16.636         [수신] DATA,21:03:16,1,4,52,1187,183,85
2026-02-04 21:03:17.648         [수신] DATA,21:03:17,8,98,73,1008,116,2
2026-02-04 21:03:18.664         [수신] DATA,21:03:18,29,99,15,1044,58,79
2026-02-04 21:03:19.671         [수신] DATA,21:03:19,15,57,26,1984,73,14
2026-02-04 21:03:20.686         [수신] DATA,21:03:20,9,43,87,1689,10,54
2026-02-04 21:03:21.701         [수신] DATA,21:03:21,14,97,45,234,150,16
2026-02-04 21:03:22.707         [수신] DATA,21:03:22,35,88,46,189,164,60
2026-02-04 21:03:23.710         [수신] DATA,21:03:23,33,49,68,240,66,77
2026-02-04 21:03:24.717         [수신] DATA,21:03:24,17,47,14,187,169,64
2026-02-04 21:03:25.726         [수신] DATA,21:03:25,32,3,68,756,72,47
2026-02-04 21:03:26.728         [수신] DATA,21:03:26,39,37,93,379,29,61
2026-02-04 21:03:27.751         [수신] DATA,21:03:27,22,61,24,1989,126,40
2026-02-04 21:03:28.761         [수신] DATA,21:03:28,13,29,75,1316,32,87
2026-02-04 21:03:29.771         [수신] DATA,21:03:29,18,11,10,204,81,57
2026-02-04 21:03:30.781         [수신] DATA,21:03:30,15,26,12,1611,62,72
2026-02-04 21:03:31.793         [수신] DATA,21:03:31,19,51,52,1966,22,44
2026-02-04 21:03:32.846         [수신] DATA,21:03:32,26,63,34,148,109,71
2026-02-04 21:03:33.860         [수신] DATA,21:03:33,21,44,84,1495,152,55
2026-02-04 21:03:34.874         [수신] DATA,21:03:34,35,14,86,1627,183,24
2026-02-04 21:03:35.890         [수신] DATA,21:03:35,14,87,97,717,158,88
```

서버 연결 상태 및 데이터 송수신 내역을 로그로 기록하여 프로그램 동작 상태를 추적할 수 있습니다.

오류 발생 시 로그 기반으로 원인 분석이 가능하도록 설계했습니다.

## CSV

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| Time | Temp | Hum | O2 | CO2 | PM10 | PM2.5 |
| 2026-02-04 21:04 | 21.3 | 51.3 | 52.6 | 811 | 105 | 51 |
| 2026-02-04 21:05 | 20 | 44.3 | 50.7 | 985 | 97 | 54 |
| 2026-02-04 21:06 | 19.6 | 47.4 | 49.7 | 1046 | 103 | 52 |

수신된 데이터를 **1**분 단위 평균값으로 계산하여 자동으로 **CSV** 파일에 저장합니다.

날짜별 파일로 관리되도록 구현하여 데이터 기록 및 분석이 용이합니다.

# Thank You