3. DEVICE COMMUNICATION HANDLING

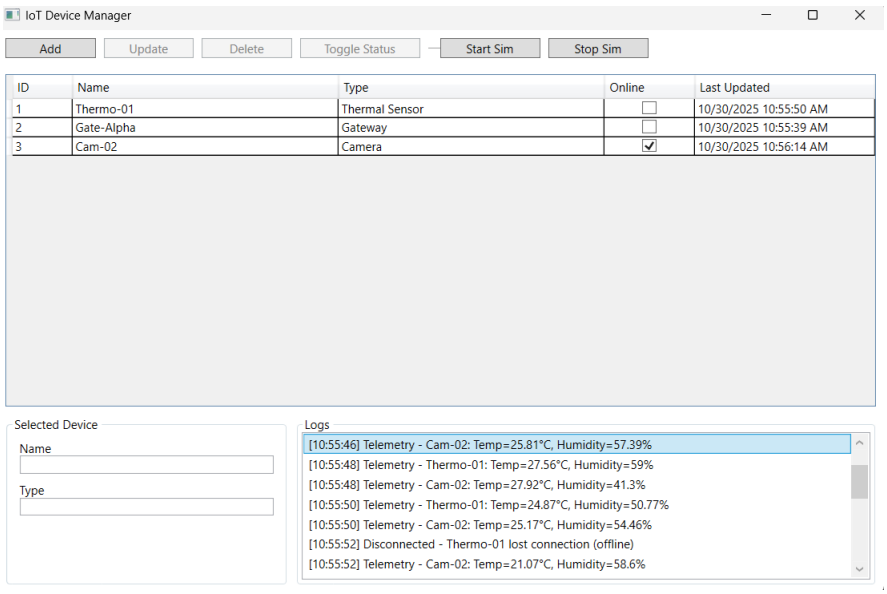
Code Snippet (DeviceSimulator.cs)

```
using System;
using System.Collections.Generic;
using System.Timers;
using IoTDeviceManager.Models;

namespace IoTDeviceManager.Services
{
    public class DeviceTelemetryEventArgs : EventArgs
    {
        public DeviceModel Device { get; }
        public double Temperature { get; }
        public double Humidity { get; }

        public DeviceTelemetryEventArgs(DeviceModel device, double temp, double humidity)
        {
            Device = device;
            Temperature = temp;
            Humidity = humidity;
        }
    }

    public class DeviceSimulator : IDisposable
    {
        private readonly Timer _timer;
        private readonly Random _rand = new();
        private readonly IList<DeviceModel> _devices;

        public event EventHandler<DeviceTelemetryEventArgs>? TelemetryReceived;
        public event EventHandler<DeviceModel>? DeviceDisconnected;
        public event EventHandler<string>? ErrorOccurred;

        public DeviceSimulator(IList<DeviceModel> devices, double intervalMs = 2000)
        {
            _devices = devices;
            _timer = new Timer(intervalMs);
            _timer.Elapsed += Tick;
        }

        public void Start() => _timer.Start();
        public void Stop() => _timer.Stop();

        private void Tick(object? sender, ElapsedEventArgs e)
        {
            foreach (var d in _devices)
            {
                try
                {
                    // 10% chance to simulate network drop
                    if (_rand.NextDouble() < 0.10)
                    {
                        d.IsOnline = false;
                        DeviceDisconnected?.Invoke(this, d);
                        continue;
                    }
```

```
                    // 30% chance to reconnect if offline
                    if (!d.IsOnline && _rand.NextDouble() < 0.30)
                    {
                        d.IsOnline = true;
                        continue;
                    }

                    if (d.IsOnline)
                    {
                        d.LastUpdated = DateTime.Now;
                        double temp = Math.Round(20 + _rand.NextDouble() * 10, 2);
                        double hum = Math.Round(40 + _rand.NextDouble() * 20, 2);

                        TelemetryReceived?.Invoke(this, new DeviceTelemetryEventArgs(d, temp,
hum));
                    }
                }
                catch (Exception ex)
                {
                    ErrorOccurred?.Invoke(this, $"Error communicating with {d.Name}:
{ex.Message}");
                }
            }
        }

        public void Dispose() => _timer.Dispose();
    }
}
```

\

Result:



Explanation:

This simulation uses a **Timer** to emulate periodic communication between the system and connected IoT devices. Every few seconds:

- Each device sends **random temperature and humidity data**.
- A **10% chance** simulates network disconnection.
- A **30% chance** allows an offline device to reconnect.
- Events (TelemetryReceived, DeviceDisconnected, ErrorOccurred) trigger log entries in the UI to mimic live communication feedback.

Real-World Implementation:

In an actual IoT system:

- Devices would communicate over **network protocols** like **MQTT**, **HTTP REST APIs**, or **CoAP**, rather than random generators.
- The server (or gateway) would receive **JSON payloads** with sensor readings and device identifiers.
- **Error handling** would rely on **timeouts**, **retry logic**, or **acknowledgment mechanisms**.
- **Cloud brokers** (e.g., AWS IoT Core, Azure IoT Hub, EMQX) or **edge gateways** would manage connectivity and status updates.
- Each device would use **secure channels** (TLS/SSL) and **asynchronous event-driven** communication for real-time telemetry streaming.

Aspect	Simulation	Real Implementation
Data transport	Timer events & random values	IoT protocols (MQTT, CoAP, HTTP REST, WebSockets)
Device connectivity	Random boolean flag	TCP/IP socket, Wi-Fi, BLE, or serial (COM) connection
Telemetry	Generated in-memory	Actual sensor data (e.g., temperature, pressure, GPS)
Data format	In-app objects	JSON or Protobuf payloads
Error handling	Simple random disconnection	Network timeout, QoS retries, exponential back-off
Logging	Local ObservableCollection in memory	Centralized monitoring (Azure Monitor / AWS CloudWatch)