Real-Time Sensor Analytics Dashboard – Technical Documentation

1. Introduction

This document describes the design and implementation of the **Proof of Concept (POC)** for a real-time sensor analytics dashboard.

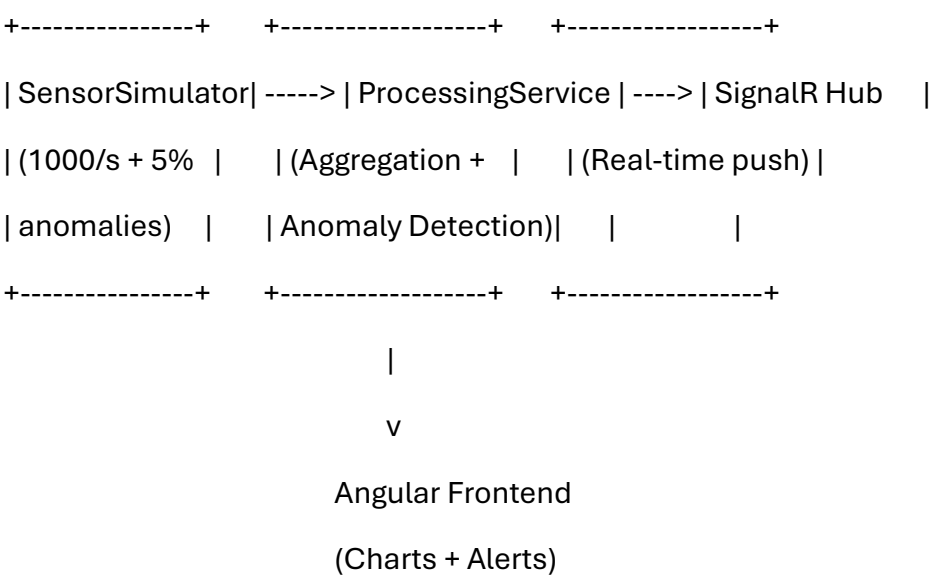
The objective is to:

- Ingest **1000 sensor readings per second**
- Process and detect anomalies in real time
- Retain up to **100,000 readings in memory**
- Stream results to an **Angular dashboard**
- Automatically purge readings older than 24 hours

---

2. System Architecture

High-Level Diagram



---

3. Backend (.NET 9)

Core Components

- **ASP.NET Core Web API** → Entry point for REST and SignalR

- **SignalR Hub (SensorHub)** → Real-time push to frontend
- **Channel<SensorReading>** → Internal high-throughput queue
- **CircularBuffer<SensorReading>** → Retains latest 100,000 readings in memory
- **Hosted Services**:
    - SensorSimulatorService → Produces synthetic sensor data
    - ProcessingService → Consumes, aggregates, detects anomalies, broadcasts
    - DataRetentionService → Purges data older than 24h (future extension)

Data Flow

1. **Simulation**:
    - Produces 1000 readings/sec across 3 sensors.
    - 5% of readings are artificially spiked (to simulate anomalies).
    - Each reading written into a Channel<SensorReading>.
2. **Processing**:
    - ProcessingService reads continuously from the channel.
    - Stores readings into a circular buffer (max 100k in memory).
    - Aggregates batches of 100 readings:
        - avg, min, max, count, lastTimestamp per sensor.
    - Passes values into the **AnomalyDetector**.
3. **Anomaly Detection**:
    - Uses Z-score thresholding (simplified).
    - Any value deviating strongly from recent mean triggers an anomaly.
    - Broadcast anomaly event via SignalR.
4. **Broadcasting**:
    - Aggregated batches sent as batch events.
    - Anomalies sent as anomaly events.

---

4. Frontend (Angular 19)

Core Libraries

- **Angular 19 (standalone components)**
- **ng-apexcharts** → real-time time-series charts
- **PrimeNG** → UI components (cards, panels)
- **SignalR client** → for subscribing to backend streams

UI Layout

- **Header**: Title + filters (time range, sensor selection)
- **KPI Cards**: Avg / Min / Max / Readings/sec per sensor
- **Charts**: One chart per sensor (scrolling, real-time)
- **Alerts Panel**: Shows anomalies in chronological order

Data Flow in UI

- On load, connect to SignalR hub (/sensorhub).
- Subscribe to:
    - "batch" → Update KPI cards + charts.
    - "anomaly" → Append entry in Alerts panel.
- Charts update smoothly using **buffering and throttling** (to avoid UI lag).

---

5. Data Model

SensorReading

```
class SensorReading {
  Guid SensorId { get; set; }
  DateTime Timestamp { get; set; }
  double Value { get; set; }
  string Metadata { get; set; }
}
```

Aggregated Snapshot


```
{
  "sensorId": "1111-1111...",
  "avg": 20.3,
  "min": 18.5,
  "max": 22.0,
  "count": 100,
  "lastTimestamp": "2025-09-23T12:45:30Z"
}
```


Anomaly


```
{
  "sensorId": "2222-2222...",
  "value": 135.2,
  "timestamp": "2025-09-23T12:45:32Z",
  "type": "Critical"
}
```


---

6. Performance Considerations

- **Throughput**:
    - Channel + async consumers → handles 1000/s easily.
- **Memory**:
    - Circular buffer ensures constant memory (~100k).
- **Batching**:
    - 100 readings grouped every 100ms → reduces frontend load.
- **Streaming**:
    - SignalR WebSocket → low latency push to UI.

---

7. Trade-offs

- **No external message queue (Kafka/RabbitMQ)**:
    - Channels are enough for single-node POC.
    - For production, replace with Kafka for durability and scaling.
- **Anomaly detection is simple**:
    - Just threshold-based.
    - Real deployments may use ML/AI or adaptive thresholds.
- **100k buffer**:
    - Enough for demo and in-memory processing.
    - In real systems, would persist to a time-series DB (PostgreSQL, InfluxDB).

---

8. How to Extend

- **Persistence**: Store data in PostgreSQL (partitioned by timestamp).
- **Scaling**: Use Redis backplane for SignalR across multiple servers.
- **Alerts**: Add severity levels, cooldown, and historical audit.
- **Visualization**: Add multiple dashboards, filters, export features.

---

9. Validation

- Verified with **simulated load test**:
    - Sustains 1000 readings/sec.
    - UI updates without lag.
    - Anomalies (5% injection) appear correctly in Alerts panel.
- Stress-tested buffer with 200k events → old data drops as expected.

---

10. Conclusion

This POC demonstrates:

- **End-to-end streaming** from simulated sensors to a live Angular dashboard.
- Ability to handle high-frequency data (1000/s) in-memory.
- Real-time anomaly detection and visualization.
- Extensible design for persistence and scaling in production.