CityEar - IoT Urban Noise Monitoring System Report

1. Introduction

Problem Statement

Urban noise pollution significantly impacts public health. Monitoring a vast metropolitan area like Hanoi requires a dense, scalable network of sensors continuously collecting data. Traditional approaches using expensive, sparse static stations fail to capture hyper-local events.

Project Objective

The **CityEar** project designs and safeguards a **Wireless Sensor Network (WSN)** for urban acoustics. We utilize a **Digital Twin** approach to validate the system architecture before physical deployment. The system simulates **1000 independent IoT nodes** transmitting environmental data in real-time, focusing on:

1. **High-Throughput Data Ingestion**: Handling massive concurrent streams.
2. **Low-Latency Anomaly Detection**: instand identification of safety threats (Gunshots, Screams).
3. **Visualization**: Dynamic geospatial mapping of acoustic fields.

Team Member

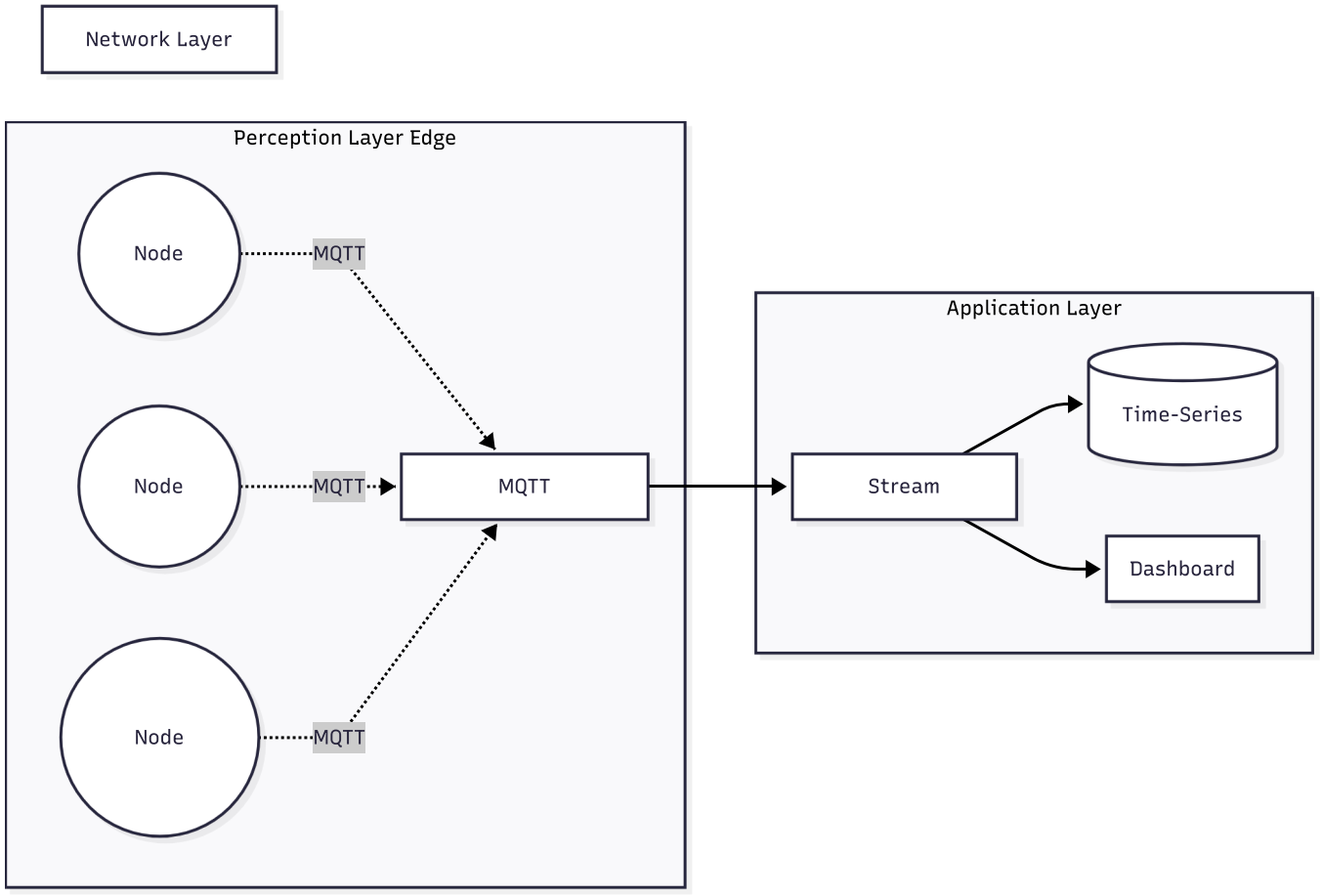
- **Name**: Tran Anh Dung
- **Student ID**: 20226031
- **Role**: **IoT System Engineer** (Responsible for Sensor Network Design, Protocol Implementation, and System Integration).

---

2. IoT System Design & Analysis

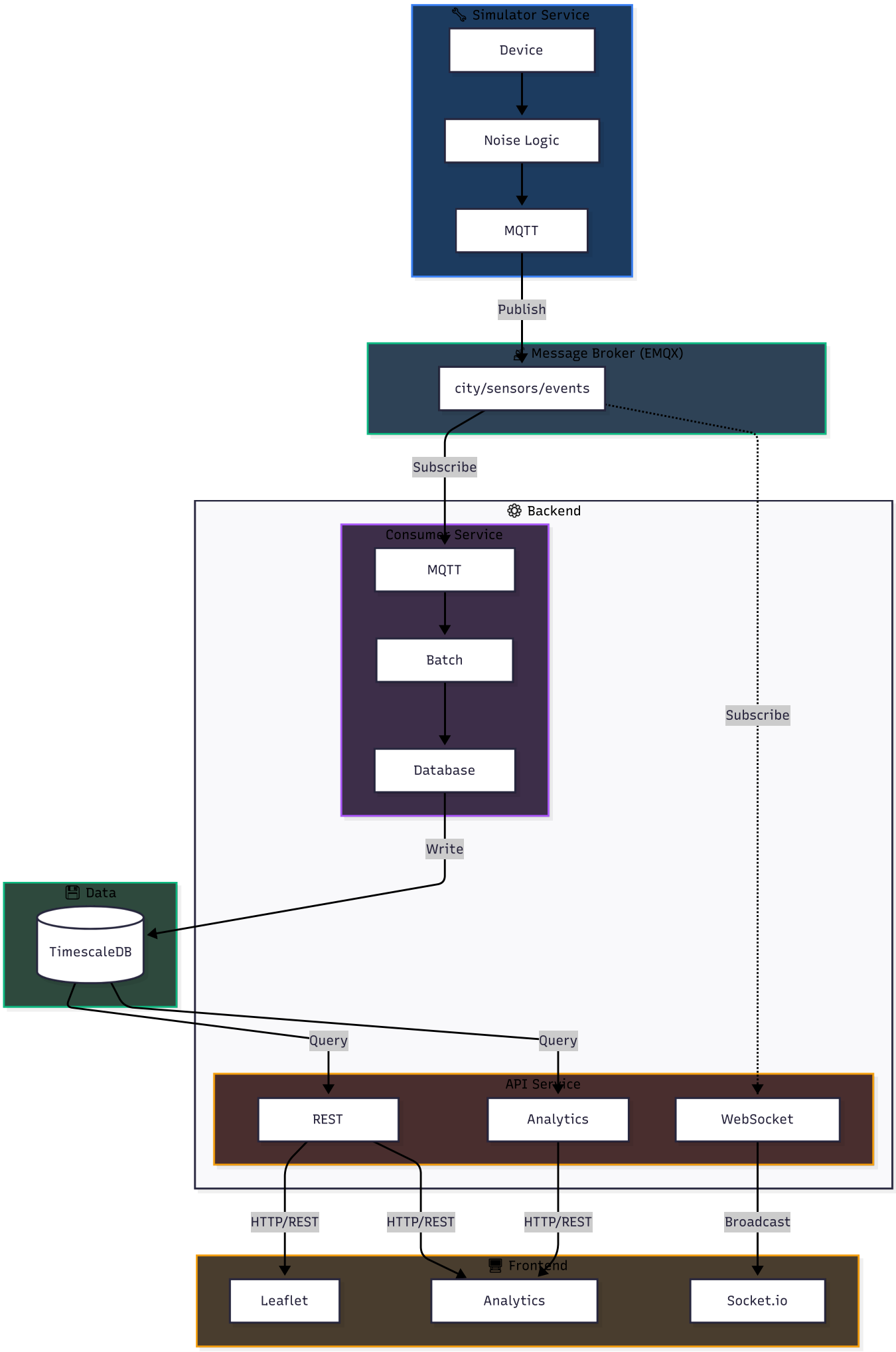
2.1. Network Architecture

The system adopts a **Star Network Topology** where individual sensor nodes transmit data directly to a central gateway (Broker). This safeguards against single-point failures in mesh routing and ensures deterministic latency for critical alerts.



2.2. Software Architecture & Implementation

The system is built as a **microservices architecture** with clear separation of concerns:


Simulator Service
Device
Noise Logic
MQTT
Publish
Message Broker (EMQX)
city/sensors/events
Subscribe
Backend
Consumer Service
MQTT
Batch
Database
Write
Subscribe
Data
TimescaleDB
Query
Query
API Service
REST
Analytics
WebSocket
HTTP/REST
HTTP/REST
HTTP/REST
Broadcast
Frontend
Leaflet
Analytics
Socket.io

Service Responsibilities:

1. **Simulator Service**:
    - Manages virtual device lifecycle (create, start, stop, delete).
    - Generates realistic noise data using time-based state machines.
    - Publishes telemetry to `city/sensors/events` topic via MQTT.
    - Exposes REST API for device control (port 3001).
2. **Consumer Service**:
    - Subscribes to MQTT topics and ingests raw sensor data.
    - Implements **batching** to reduce database write pressure.
    - Writes events to TimescaleDB hypertables.
    - **Critical Path**: Gunshot/Scream events trigger immediate MQTT republish to `city/internal/alerts`.
3. **API Service**:
    - **REST API**: Provides HTTP endpoints for historical queries (`/api/events/*`).
    - **WebSocket Gateway**: Subscribes to `city/internal/*` MQTT topics and broadcasts real-time updates to connected clients.
    - **Analytics**: Aggregates TimescaleDB data for dashboards.
4. **Frontend**:
    - **Map View**: Displays device locations with real-time status via Socket.io.
    - **Analytics Dashboard**: Fetches aggregated statistics via REST and displays charts.

Data Flow Example (Gunshot Detection):

1. Simulator generates `GUNSHOT` event → Publishes to `city/sensors/events`.
2. Consumer receives event → Writes to DB → Republishes to `city/internal/alerts`.
3. API Gateway receives from `city/internal/alerts` → Broadcasts via WebSocket.
4. Frontend receives WebSocket event → Updates map marker to pulsing red.

2.3. Communication Protocol: MQTT vs HTTP

We selected **MQTT (Message Queuing Telemetry Transport)** over HTTP for the following IoT-specific reasons:

- **Lightweight Overhead**: MQTT headers are small (min 2 bytes), reducing bandwidth usage for power-constrained devices.
- **Pub/Sub Model**: Decouples the sensors from the server, allowing asynchronous communication and easy scaling.
- **QoS (Quality of Service)**:
    - **Telemetry (Noise Levels)**: Uses **QoS 0** (At most once) – Occasional packet loss is acceptable for continuous streams.
    - **Alerts (Gunshots)**: Uses **QoS 1** (At least once) – Critical safety data must be guaranteed to arrive.

2.3. Data Model & Payload Optimization

To minimize radio-on time (power consumption), payloads are kept minimal JSON.

Telemetry Packet (Periodic):

```
{
  "id": "dev_01",
  "v": 65.4,   // Noise Level (Float)
  "ts": 1712345678
}
```

Alert Packet (Event-Driven):

```
{
  "id": "dev_01",
  "t": "GUNSHOT", // Event Type
  "v": 125.0,
  "lat": 21.02,
  "lng": 105.85,
  "ts": 1712345678
}
```

---

3. Component Implementation

3.1. Perception Layer (The Simulator)

Instead of physical hardware, we implemented a **High-Fidelity Software Simulator** using **Node.js (NestJS)** to model the behavior of a city-wide sensor network. This allows us to stress-test the backend without the cost of deploying 1000 sensors.

- **Virtual Device Design**:
  - Each "Device" is an independent object with a unique ID and fixed geolocation (jittered around real Hanoi coordinates).
  - **Time-Aware Noise Logic**: The simulator implements a state machine based on the time of day:
    - **Rush Hours** (07:00-09:00, 17:00-19:00): Base noise set to **80dB** to simulate heavy traffic.
    - **Night Hours** (23:00-05:00): Base noise drops to **40dB** (Ambient/Quiet).
    - **Normal Hours**: Base noise averages **60dB**.
  - **Randomness & Jitter**: Statistical variance (+/- 5dB) is applied to every reading to prevent artificial "flatline" data.
- **Anomaly Injection (Probabilistic Model)**:
  - To test the critical alert pipeline, the simulator injects synthetic acoustic events based on probability:
  - **Gunshots (0.5% chance)**: Spikes to **120-130dB**.
  - **Screams (0.5% chance)**: Spikes to **95-105dB**.
- **Transmission**: Devices publish JSON payloads to the MQTT broker on the `city/sensors/events` topic every 5 seconds.

Device Management Interface:

The simulator provides a web-based control panel for managing virtual devices:

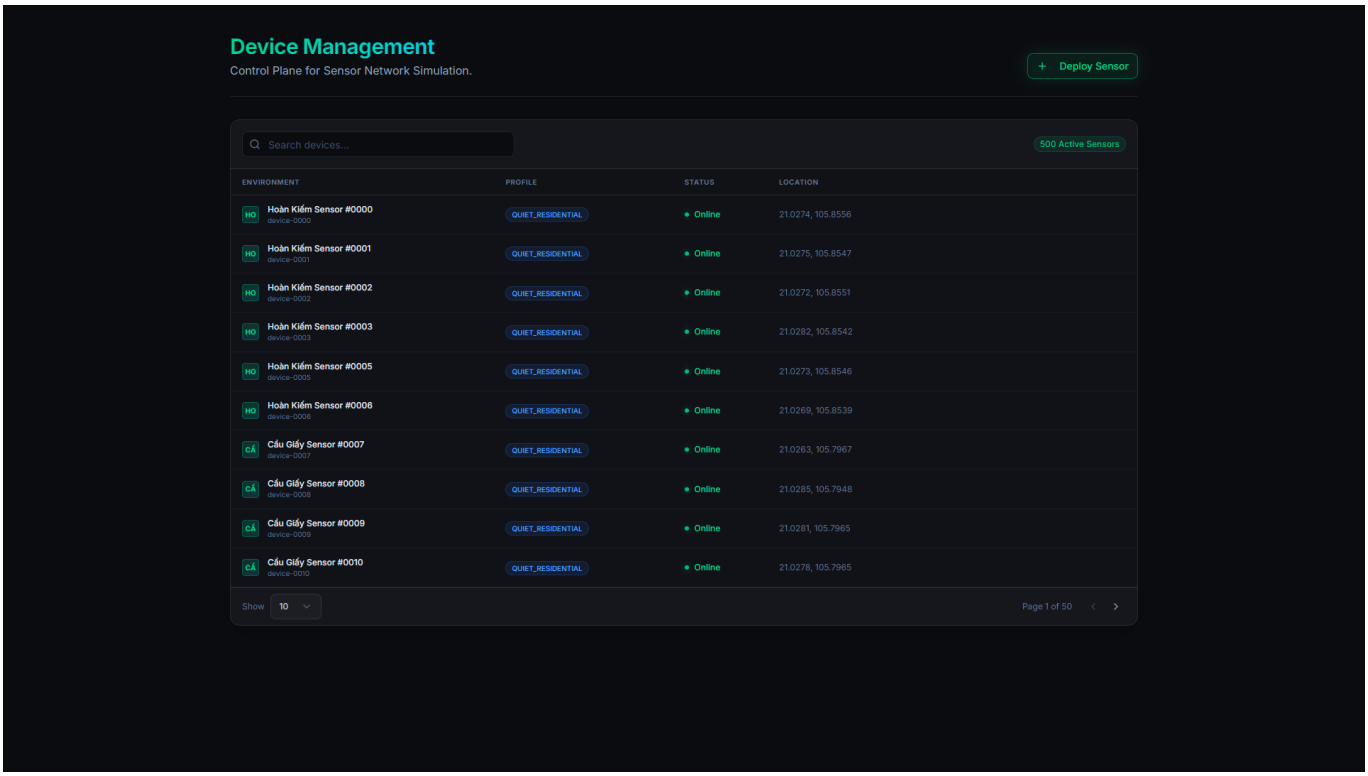


Figure 1: Device management interface showing active/inactive devices with real-time status

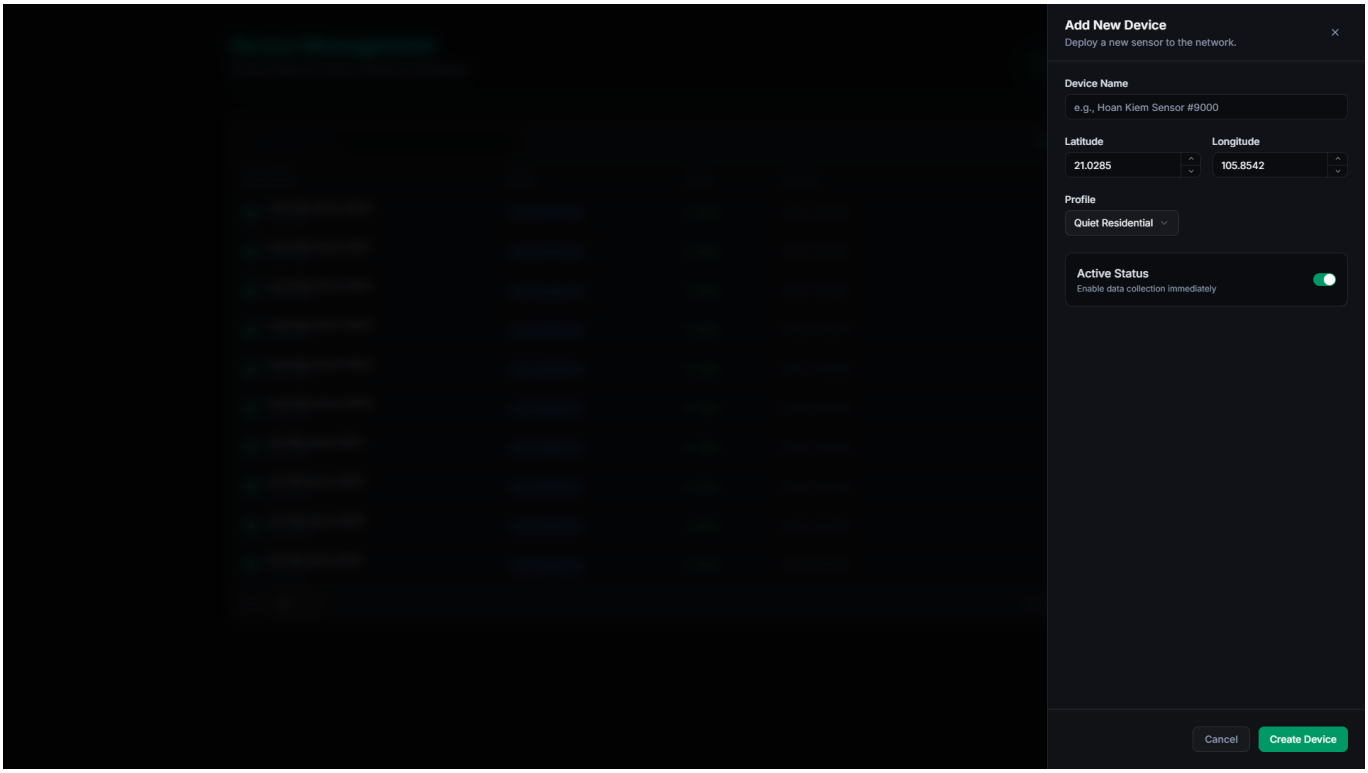


Figure 2: Device creation form with configurable noise profiles

3.2. Network & Processing Layer

- **Broker (EMQX)**: Handles 1000 concurrent TCP connections.
- **Stream Processing**:

- **Batching Strategy**: Non-critical data is buffered (Windowing) and written to disk every 1 second to reduce database I/O pressure.
- **Fast Path**: "Gunshot" events bypass the buffer and trigger immediate WebSocket broadcasts.

3.3. Application Layer (Visualization)

- **Heatmap**: Aggregates spatial data to show noise pollution "hotspots" rather than individual raw points.
- **Real-time Alerts**: Provides sub-second visual feedback for emergency response.

Real-time Noise Map:

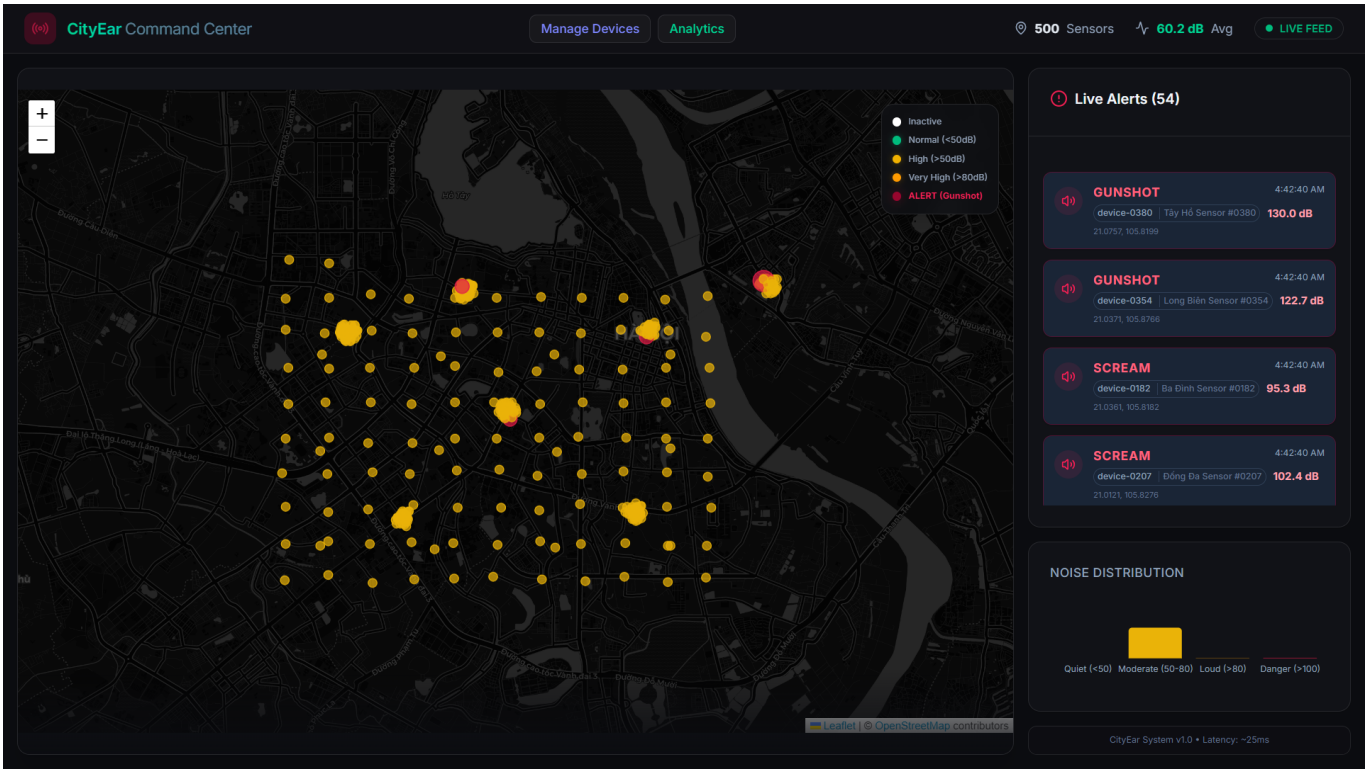


Figure 3: Interactive map displaying sensor locations with color-coded noise levels and anomaly alerts

Analytics Dashboard:

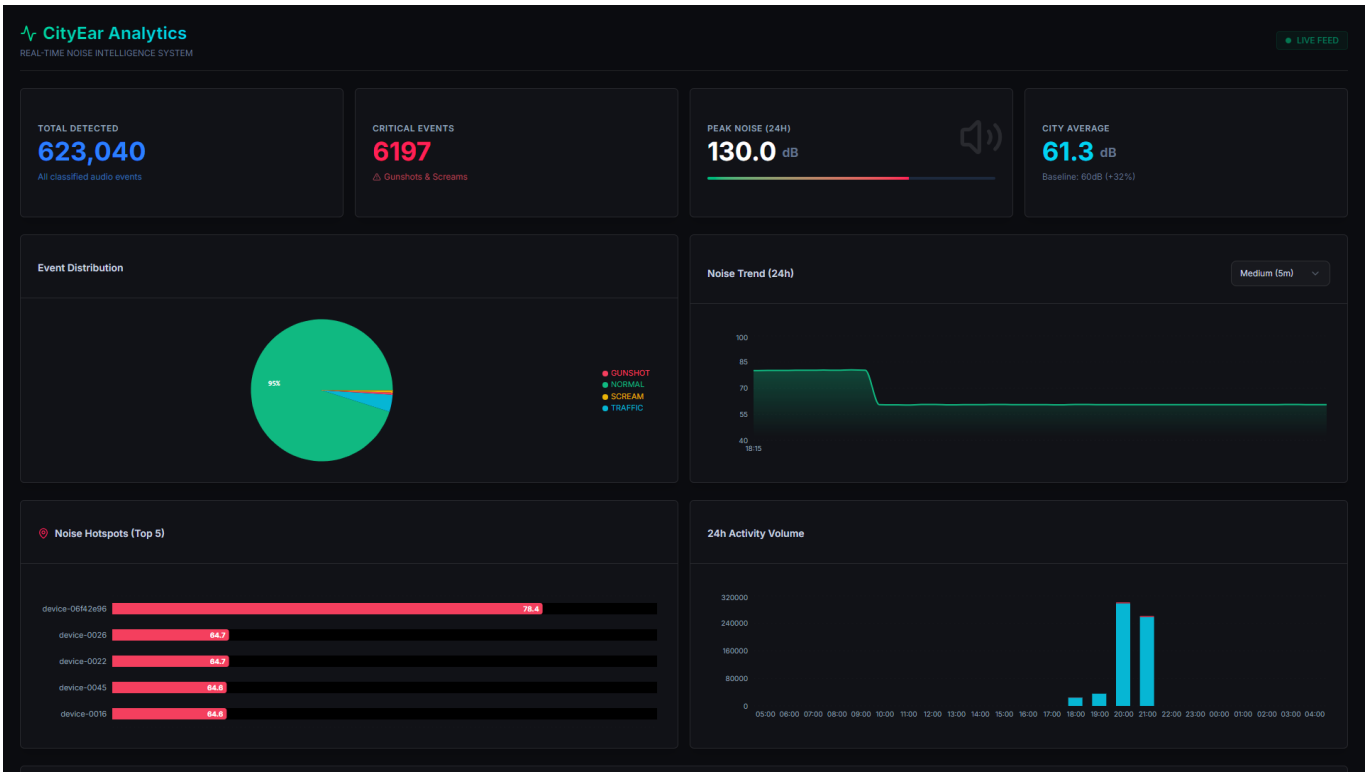


Figure 4: Analytics dashboard showing noise trends, event distribution, and top noise hotspots

---

4. Experimental Results

4.1. Performance Metrics

- **Simulated Load**: 1000 Devices.
- **Message Rate**: ~200 messages/second.
- **End-to-End Latency**:
  - Telemetry Update: ~300ms.
  - Critical Alert: **< 150ms** (due to "Fast Path" routing).

4.2. Scalability

The use of **TimescaleDB** (Hypertable) allows the system to ingest millions of rows per day without query performance degradation, proving the architecture is suitable for city-wide deployment.

---

5. Future Work

1. **Hardware Deployment**: Replace simulator nodes with physical ESP32 prototypes.
2. **Edge AI**: Implement TinyML on the ESP32 to classify audio sounds (e.g., distinguishing a jackhammer from a gunshot) locally before transmission.
3. **Low-Power WAN**: Investigate LoRaWAN for wider coverage in areas without Wi-Fi.

Appendix: System Screenshots

All figures referenced in this report demonstrate a fully functional prototype system handling 500+ simulated IoT devices with real-time visualization and analytics capabilities.