Technical Backend Report: Intelligent Air Quality Index Monitoring System

1. Executive Summary

This report details the architecture, implementation, and operational capabilities of the backend system designed for the Intelligent Air Quality Index (AQI) Monitoring platform.

We have moved beyond a simple CRUD application to build a **data-driven intelligence platform**. The system does not merely act as a proxy for external APIs; it ingests, persists, analyzes, and proactively disseminates environmental data. By leveraging a modular Spring Boot architecture, we have created a resilient system capable of monitoring the top 20 cities in Pakistan, providing granular pollutant analysis (PM2.5, PM10,O3, etc.), and issuing automated health alerts.

---

2. System Objectives

The backend was engineered to solve three specific problems:

1. **Data Persistence & Independence:** To decouple the frontend from external API rate limits and downtime by maintaining an internal historical database.
2. **Proactive Alerting:** To transform passive data viewing into active user protection via automated email notifications when air quality degrades.
3. **Actionable Intelligence:** To interpret raw scientific data into human-readable health advice and activity recommendations (e.g., "Best time to run").

---

3. Technical Architecture & Workflow

The system follows a **Layered Monolithic Architecture**, designed for ease of deployment and high cohesion.

3.1. Data Ingestion Pipeline (The Heartbeat)

The core of the system is the **Automated Scheduler (`AqiDataPoller`)**.

- **Workflow:** Every 15 minutes, a scheduled task triggers.
- **Operation:** It iterates through a hardcoded list of Pakistan's top 20 major cities (including Lahore, Karachi, Islamabad, and Tier-2 cities like Gujrat and Larkana).
- **External Communication:** It queries the **Open-Meteo Air Quality API**. We migrated to this provider to access granular pollutant data (CO, NO2, SO2, O_3) and to leverage its generous non-commercial rate limits (10,000 calls/day), ensuring scalability without cost.

- **Persistence:** Data is normalized and stored in a PostgreSQL database. This allows for historical trend analysis without re-querying external services.

3.2. The Alerting Engine

- **Trigger:** Immediately after data ingestion, the system evaluates the US AQI score against a predefined threshold (default: 150 - Unhealthy).
- **Logic:** If a city crosses the threshold, the system queries the user database for all subscribers who have designated that city as their "Home Location."
- **Action:** An asynchronous event triggers the `NotificationService`, dispatching email alerts via SMTP (Gmail) without blocking the main execution thread.

3.3. Security Layer

- **Stateless Authentication:** The system uses **JWT (JSON Web Tokens)**.
- **Flow:** Upon login, the server issues a signed token. Subsequent requests must bear this token in the header. The server validates the signature and expiration on every request via a custom filter chain, eliminating the need for server-side session storage.

---

4. Technology Stack

We utilized a modern, industry-standard Java stack optimized for reliability.

4.1. Core Frameworks

- **Java 17 (LTS):** Chosen for its performance improvements and long-term support.
- **Spring Boot 3.5.7:** The backbone of the application, handling dependency injection, web server (Tomcat) configuration, and auto-configuration.
- **Spring Security 6:** Handles authentication (AuthN) and authorization (AuthZ). Configured with a `SecurityFilterChain` to secure endpoints while whitelisting public resources like Swagger UI.

4.2. Database & Persistence

- **PostgreSQL:** A robust relational database used for storing user profiles and time-series AQI data.
- **Spring Data JPA (Hibernate):** Used as the ORM (Object-Relational Mapper). We utilize "Magic Methods" (e.g., `findFirstByCityIgnoreCaseOrderByTimestampDesc`) to abstract complex SQL queries and ensure case-insensitive data retrieval.

4.3. External APIs & Libraries

- **Open-Meteo Air Quality API:** The primary source of truth. It provides current conditions, pollutant breakdown, and 4-day forecasts.

- **JJWT (Java JWT):** Used for generating and verifying security tokens.
- **JavaMailSender:** Standard library for SMTP integration to send HTML-formatted emails.
- **SpringDoc OpenAPI (Swagger UI):** Automatically generates interactive API documentation, allowing the frontend team to test endpoints without writing code.
- **Dotenv (Java):** A critical utility for loading environment variables (`.env`) into the application context, ensuring sensitive keys (DB passwords, JWT secrets) are never hardcoded.

---

5. API Endpoints Specification

The backend exposes RESTful endpoints grouped by functionality. All responses return standard JSON.

5.1. Authentication Module (`/api/auth`)

- **`POST /register`**: Accepts email/password. Creates a user, hashes the password (BCrypt), generates a JWT, and triggers a background "Welcome Email."
- **`POST /login`**: Validates credentials. Returns a JWT if successful, or `401 Unauthorized` if failed.

5.2. User Module (`/api/users`)

- **`GET /me`**: Returns the profile of the currently authenticated user.
- **`PUT /me/city`**: Updates the user's preferred city for monitoring. This links the user to the Alerting Engine.

5.3. AQI Data Module (`/api/aqi`)

- **`GET /{city}`**: Returns the *latest* stored data for a specific city. Includes the AQI, raw pollutant values ($PM_{2.5}$, PM_{10}, etc.), and a dynamically generated **Health Advice** string.
- **`GET /{city}/history`**: Returns a list of historical data points for charting trends.
- **`GET /{city}/forecast`**: Proxies a request to Open-Meteo to return hourly AQI predictions for the next 3-4 days.
- **`GET /{city}/recommendation`**: The "Intelligent" feature. Scans the forecast to find the time window with the lowest AQI in the next 24 hours and returns a message (e.g., *"Best time to go outside: Tomorrow at 7:00 AM"*).

---

6. User Manual & Installation Guide

This section serves as a guide for setting up the backend environment from scratch.

6.1. Prerequisites

Ensure the following are installed on the host machine:

1. **Java Development Kit (JDK) 17** or higher.
2. **Maven** (for dependency management).
3. **PostgreSQL** (Service must be running).
4. **IntelliJ IDEA** (Recommended) or any Java IDE.

6.2. Database Setup

1. Open your terminal or PGAdmin.
2. Create a new database named `aqi_db`.
3. `CREATE DATABASE aqi_db;`
4. The application will automatically create the necessary tables (`users`, `aqi_data_points`) upon the first run.

6.3. Environment Configuration

Crucial Step: The application will *not* start without a properly configured `.env` file.

1. Create a file named `.env` in the **root directory** of the project (same level as `pom.xml`).
2. Populate it with the following credentials:
3. Properties

```
# Database Credentials
DB_PASSWORD=your_postgres_password

# Email Settings (Use an App Password for Gmail, not your login password)
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_EMAIL=your_email@gmail.com
MAIL_PASSWORD=your_google_app_password

# Security
# Generate a long random string (Base64 encoded recommended)
JWT_SECRET=your_very_long_secure_secret_key_minimum_64_chars
```

4.
5.

6.4. Running the System

You can run the system either via the IDE or the command line.

Via IntelliJ IDEA

1. Open the project.
2. Reload the Maven project to download dependencies.

3. Locate `AqiMonitoringSystemApplication.java`.
4. Click the **Run** (Green Play) button.

6.5. Verification

Once the application starts:

1. Watch the console logs. You should see the `[AQI Poller]` activate within 5 seconds and begin fetching data for 20 cities.
2. Open your browser and navigate to: http://localhost:8080/swagger-ui.html
3. This will load the interactive API documentation where you can test endpoints directly.