Francisco José de Caldas District University

Faculty of Engineering
Systems Engineering Program

Architecture for Real-Time Air Quality Monitoring and Personalized Recommendations

Johan Esteban Castaño Martínez
Stivel Pinilla Puerta

Jose Alejando Cortazar Lopez

Supervisor: Carlos Andrés Sierra

A report submitted in partial fulfilment of the requirements of
Francisco José de Caldas District University for the degree of
Bachelor of Science in *Systems Engineering*

October 2025

Declaration

We, Johan Esteban Castaño Martínez, Stivel Pinilla Puerta, and Jose Alejando Cortazar Lopez, of the Systems Engineering Program, Francisco José de Caldas District University, confirm that this is our own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. We understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

We give consent to a copy of our report being shared with future students as an exemplar.

We give consent for our work to be made available more widely to members of the university and public with interest in teaching, learning and research.

Johan Esteban Castaño Martínez
Stivel Pinilla Puerta
Jose Alejando Cortazar Lopez
October 2025

Abstract

Air pollution remains the second leading cause of premature death worldwide, with disproportionate impacts on Latin American megacities such as Bogotá. This report presents a comprehensive cloud-ready architecture designed to integrate real-time air quality data from multiple sources and provide personalized health recommendations to citizens. The system collects ten-minute data streams from environmental sensors and third-party APIs (AQICN, Google Air Quality, and IQAir) using a Python Ingestor with raw storage in MinIO. A lightweight normalization pipeline transforms heterogeneous payloads before inserting records into a monthly-partitioned PostgreSQL database extended with TimescaleDB hypertables for efficient time-series operations. Query acceleration is achieved through concurrently refreshed materialized views that support an API layer (REST + GraphQL) designed to serve real-time dashboards and personalized health recommendations. The architecture includes a recommendation engine that translates AQI thresholds and user metadata into color-coded health advice and protective product suggestions during high pollution periods. The system targets p95 latencies below 2 seconds under 1000 concurrent users, with an initial deployment covering three years (2022–2024) of Bogotá air quality data. This work addresses the critical gap between raw environmental data availability and actionable citizen-oriented decision support, demonstrating how modern database technologies can be leveraged to improve public health outcomes in urban environments.

Keywords: Air Quality Index (AQI), TimescaleDB, PostgreSQL partitioning, MinIO, Materialized Views, Personalized Recommendation

Report's total word count: Approximately 10,000-15,000 words (starting from Chapter 1 and finishing at the end of the conclusions chapter, excluding references, appendices, abstract, text in figures, tables, listings, and captions).

Source Code Repository: https://github.com/DarcanoS/Database-II

This report was submitted as part of the Databases II course requirements at Francisco José de Caldas District University, Faculty of Engineering, Systems Engineering Program.

Acknowledgements

We would like to express our sincere gratitude to Professor Carlos Andrés Sierra for his guidance, support, and valuable insights throughout the development of this project. His expertise in database systems and dedication to teaching have been instrumental in shaping our understanding of advanced database architectures and their practical applications.

We are also grateful to Francisco José de Caldas District University, Faculty of Engineering, and the Systems Engineering Program for providing the academic environment and resources necessary to complete this work.

Special thanks to our families and friends for their continuous encouragement and support during the course of this project.

Finally, we acknowledge the open-source community and the developers of PostgreSQL, TimescaleDB, MinIO, and other technologies that made this project possible. We also thank the organizations providing public air quality data APIs (AQICN, Google Air Quality, and IQAir) whose services are essential for environmental monitoring and public health research.

We also appreciate the assistance of modern AI tools that supported the documentation and writing process, helping us articulate technical concepts more clearly and efficiently.

Contents

List of Figures

List of Tables

List of Abbreviations

AQI	Air Quality Index
AQICN	Air Quality Index China Network
API	Application Programming Interface
BRIN	Block Range INdex
COPD	Chronic Obstructive Pulmonary Disease
CSV	Comma-Separated Values
DDL	Data Definition Language
EHR	Electronic Health Record
EPA	Environmental Protection Agency (United States)
JSON	JavaScript Object Notation
NFR	Non-Functional Requirement
PM2.5	Particulate Matter $\leq 2.5\,\mu$m
REST	REpresentational State Transfer
WHO	World Health Organization

Chapter 1

Introduction

Air quality monitoring has become a critical concern for public health, particularly in rapidly growing urban centers. This report presents the design and implementation of a comprehensive architecture for real-time air quality monitoring and personalized health recommendations, with a specific focus on Bogotá, Colombia.

1.1 Background

Ambient and household air pollution jointly kill an estimated seven to eight million people every year, with 99% of the world's population breathing air that exceeds WHO guideline values. The 2024 State of Global Air report ranks $PM_{2.5}$ exposure as the second leading risk factor for mortality, ahead of high blood pressure or smoking. In Bogotá, long-term analyses still show spatially heterogeneous $PM_{2.5}$ hot spots, despite a gradual decline from 15.7 $\mu g/m^3$ in 2017 to 13.1 $\mu g/m^3$ in 2019 and targeted reductions after the city's Air Plan 2030.

While multiple data sources for air quality information exist, including AQICN (minute-level AQI for over 100 countries), Google Air Quality API (500m-resolution indices), and IQAir AirVisual (calibrated sensor networks), these platforms present significant limitations. They provide raw values without personalization, enforce strict quota limits that complicate city-scale analytics, and generally aggregate data hourly with tiered pricing for higher-volume access. Citizens seeking actionable information face the challenge of navigating multiple fragmented sources, understanding technical indicators, and interpreting how air quality conditions specifically affect their health and daily activities.

This project addresses the gap between abundant raw environmental data and the lack of intuitive, rapid, and personalized access to air quality information. By integrating heterogeneous data sources into a unified platform with real-time dashboards, personalized recommendations, and health alerts, the system aims to empower citizens with actionable insights that directly support daily decision-making regarding outdoor activities, health precautions, and protective measures.

1.2 Problem statement

Despite the availability of multiple air quality data sources, citizens in Bogotá face significant challenges in accessing intuitive and rapid information about air pollution levels and their health implications. Current platforms require users to:

- Navigate multiple fragmented data sources with inconsistent formats and units

- Interpret technical indicators ($PM_{2.5}$, PM_{10}, O_3, NO_2, etc.) without clear health guidance
- Manually correlate air quality levels with personal health conditions and activities
- Access information through platforms that lack real-time updates or personalization
- Understand complex technical documentation without citizen-oriented interfaces

This fragmentation and lack of personalization creates a barrier between valuable environmental data and actionable health decisions, particularly affecting vulnerable populations such as children, elderly individuals, and people with respiratory conditions who would benefit most from timely, personalized air quality guidance.

1.3 Aims and objectives

Aim: The primary aim of this project is to design and implement a centralized, cloud-ready air quality monitoring platform that integrates real-time data from multiple sources and provides personalized, actionable health recommendations to citizens in Bogotá.

Objectives: To achieve this aim, the following specific objectives were established:

1. Design a scalable database architecture using PostgreSQL with TimescaleDB extensions to efficiently store and query time-series air quality data from multiple sources.
2. Implement a data ingestion pipeline that collects data from external APIs (AQICN, Google Air Quality, IQAir) at 10-minute intervals with raw storage in MinIO.
3. Develop a normalization process that transforms heterogeneous data formats into a unified schema with monthly partitioning for optimal query performance.
4. Implement materialized views with concurrent refresh capabilities to accelerate queries and support sub-2-second response times under high load.
5. Design and implement a REST + GraphQL API layer to serve real-time dashboards and support diverse client applications.
6. Develop a recommendation engine that translates AQI thresholds and user metadata into personalized health advice and protective product suggestions.
7. Validate the system architecture against performance requirements including p95 latencies below 2 seconds under 1000 concurrent users.
8. Document the complete system architecture, implementation details, and lessons learned for future scalability and multi-region deployments.

1.4 Solution approach

The solution adopts a modern, cloud-ready architecture based on PostgreSQL as the core data store, extended with TimescaleDB for efficient time-series operations. The approach follows a systematic pipeline architecture that addresses data ingestion, normalization, storage, and delivery:

1.4.1 Data Collection and Ingestion

A Python-based ingestion service operates on a 10-minute interval cycle, collecting air quality data from three primary external APIs: AQICN, Google Air Quality, and IQAir. Raw payloads are immediately stored in MinIO object storage to preserve original data for audit trails and potential reprocessing. This two-stage approach ensures data durability while allowing the normalization pipeline to operate asynchronously.

1.4.2 Database Architecture and Query Optimization

The normalized data is stored in a PostgreSQL database utilizing monthly partitioning strategies to optimize query performance for time-range filters. TimescaleDB hypertables provide automatic time-series optimizations, including compression and continuous aggregates. To achieve the target sub-2-second query latency, the system employs concurrently refreshed materialized views that pre-compute common aggregations and summary statistics. This multi-layered storage strategy balances write throughput for real-time ingestion with read performance for user queries and dashboard rendering.

1.4.3 API Layer and User Services

A dual-protocol API layer supports both REST endpoints for simple queries and GraphQL for complex, nested data requirements. This flexibility allows mobile applications, web dashboards, and third-party integrations to efficiently access the data they need. The recommendation engine operates as a separate service that consumes air quality data alongside user profiles (location, activity patterns, health conditions) to generate personalized health advice, protective product suggestions, and cleaner-area navigation guidance during high pollution periods.

1.5 Summary of contributions and achievements

This project makes three primary contributions to the field of environmental data systems and public health informatics. First, it demonstrates an end-to-end PostgreSQL-based architecture that successfully unifies three heterogeneous public APIs through a systematic ingestion, normalization, and storage pipeline. The implementation of TimescaleDB hypertables with monthly city-based partitioning provides a practical model for handling large-scale time-series environmental data with efficient query performance.

Second, the lightweight recommendation engine bridges the gap between raw environmental metrics and citizen-actionable guidance. By translating technical AQI thresholds and pollutant concentrations into color-coded health advice, protective product suggestions, and cleaner-area navigation, the system addresses user stories that emphasize personalized health support rather than merely data visualization.

Third, the architecture targets specific performance metrics designed to meet non-functional requirements including sub-2-second query response times at the 95th percentile under 1000 concurrent users. The use of concurrently refreshed materialized views demonstrates how modern database features can be leveraged to achieve real-time responsiveness without sacrificing data accuracy or resorting to complex distributed streaming frameworks.

The initial deployment focuses on three years (2022-2024) of historical Bogotá air quality data, providing a foundation for future expansion to predictive modeling, multi-region deployments, and integration with additional health and environmental data sources.

1.6 Organization of the report

The rest of this report is organised as follows:

1. Chapter 2 presents a comprehensive literature review covering air quality monitoring systems, time-series database technologies, and real-time data processing architectures relevant to environmental health informatics;
2. Chapter 3 describes the methodology and system design, including the database schema, partitioning strategies, materialized view implementation, API architecture, and recommendation engine logic;
3. Chapter 4 illustrates the implementation details, including the Python ingestion service, normalization pipeline, database configuration, and API endpoints;
4. Chapter 5 presents the testing and validation approach, including performance benchmarking, load testing methodology, and compliance verification against functional and non-functional requirements;
5. Chapter 6 discusses the results, analyzing query performance, system scalability, and effectiveness of the recommendation engine; and
6. Finally, Chapter 6 concludes the report, summarizing key findings, discussing limitations, and outlining future work including predictive modeling capabilities and multi-region deployment strategies.

Chapter 2

Literature Review

This chapter reviews the existing literature and technological solutions related to air quality monitoring systems, time-series database architectures, real-time data processing frameworks, and personalized recommendation engines. The review establishes the theoretical foundation for the system design choices presented in Chapter 3 and positions this project within the broader context of environmental health informatics and database systems research.

2.1 Air Quality Monitoring: A Public Health Imperative

Air pollution represents one of the most significant environmental health challenges of the 21st century. According to the World Health Organization, ambient and household air pollution jointly cause an estimated seven to eight million premature deaths annually, with 99% of the global population exposed to air that exceeds WHO guideline values (World Health Organization, 2024). The 2024 State of Global Air report identifies $PM_{2.5}$ (particulate matter with diameter less than 2.5 micrometers) exposure as the second leading risk factor for mortality worldwide, surpassing well-known factors such as high blood pressure and tobacco smoking (Health Effects Institute, 2024).

2.1.1 Air Quality in Latin American Megacities

Latin American urban centers face particularly acute air quality challenges due to rapid urbanization, vehicular emissions, industrial activities, and geographic conditions that trap pollutants. Bogotá, Colombia's capital and the focus of this project, has shown spatially heterogeneous $PM_{2.5}$ concentrations despite gradual improvements. Studies document a decline from 15.7 $\mu g/m^3$ in 2017 to 13.1 $\mu g/m^3$ in 2019, driven in part by the city's Air Plan 2030 initiative. However, these levels still exceed WHO recommended limits, and significant hot spots persist in industrial and high-traffic zones.

The public health implications extend beyond mortality statistics. Chronic exposure to elevated $PM_{2.5}$ levels correlates with increased incidence of respiratory diseases, cardiovascular conditions, and adverse pregnancy outcomes. Vulnerable populations—including children, elderly individuals, and those with pre-existing respiratory conditions—face disproportionate risks. These findings underscore the critical need for continuous, fine-grained monitoring combined with citizen-oriented decision support systems that translate technical air quality metrics into actionable health guidance.

2.2 Existing Air Quality Data Platforms

Multiple platforms provide real-time and historical air quality data through public APIs and web interfaces. Understanding their capabilities and limitations informed the design decisions for this project's data integration strategy.

2.2.1 AQICN (Air Quality Index China Network)

AQICN offers one of the most comprehensive global air quality databases, providing minute-level Air Quality Index (AQI) readings and historical archives for over 100 countries (Air Quality Index China Network, 2024). The platform aggregates data from government monitoring stations, low-cost sensor networks, and satellite observations. Historical data is available in CSV format dating back to 2015, making it valuable for retrospective analyses and baseline establishment.

However, AQICN presents data primarily as raw numerical values without personalization or contextual health guidance. Users must independently interpret AQI values and understand the implications for their specific health conditions and planned activities. The platform's strength lies in comprehensive spatial and temporal coverage rather than user-oriented decision support.

2.2.2 Google Air Quality API

Google's Air Quality API provides high-resolution (500-meter grid) air quality indices combined with pollutant-specific concentrations and basic health recommendations (Google, 2024). The API returns structured JSON responses that include current conditions, hourly forecasts, and general health tips categorized by activity type (e.g., outdoor exercise, commuting).

While Google's service offers superior spatial granularity and more accessible health messaging compared to AQICN, it enforces strict quota limits that complicate city-scale analytics and continuous monitoring applications. Free-tier usage allows only limited requests per day, and high-volume access requires enterprise agreements. Additionally, the service focuses primarily on current and short-term forecast data, with limited support for long-term historical analysis.

2.2.3 IQAir AirVisual

IQAir operates a global network of calibrated air quality sensors and provides data through both consumer applications and a REST API (IQAir, 2024). The platform emphasizes sensor accuracy and calibration protocols, offering reliability superior to many low-cost sensor networks. Data is available at hourly aggregation intervals with real-time updates for major metropolitan areas.

IQAir's business model includes tiered pricing for API access, with free tiers limited to basic functionality and higher-volume data retrieval requiring paid subscriptions. This pricing structure presents barriers for research projects and non-commercial applications requiring comprehensive historical data access. Furthermore, hourly aggregation may be insufficient for applications requiring finer temporal granularity to capture rapid pollution events.

2.3 Time-Series Database Technologies

Air quality monitoring generates inherently time-series data characterized by regular temporal sampling, append-heavy write patterns, and queries focused on temporal ranges and

aggregations. Traditional relational databases can struggle with the scale and performance requirements of high-frequency environmental monitoring. This section reviews specialized time-series database technologies and extensions that address these challenges.

2.3.1 PostgreSQL and Relational Foundations

PostgreSQL provides a robust, ACID-compliant relational database foundation with extensive support for complex queries, foreign key constraints, and transactional integrity (PostgreSQL Global Development Group, 2024*b*). Its extensibility through custom data types, functions, and extensions makes it particularly suitable for hybrid workloads combining structured relational data (users, stations, permissions) with time-series measurements.

Native PostgreSQL features relevant to time-series workloads include declarative table partitioning (introduced in PostgreSQL 10 and enhanced in subsequent versions), which enables horizontal splitting of large tables based on time ranges. Monthly or yearly partitions allow the query planner to prune irrelevant data, significantly reducing I/O for time-bounded queries. PostgreSQL's MVCC (Multi-Version Concurrency Control) architecture provides non-blocking reads during concurrent write operations, essential for systems ingesting data continuously while serving user queries.

2.3.2 TimescaleDB: Time-Series Extension

TimescaleDB extends PostgreSQL with specialized time-series optimizations while maintaining full SQL compatibility and leveraging PostgreSQL's reliability and ecosystem (Timescale Inc., 2024). The core abstraction is the *hypertable*, which automatically partitions data across multiple chunks based on time intervals while presenting a unified table interface to applications.

Key features relevant to air quality monitoring include:

- **Automatic chunk management**: TimescaleDB creates and manages time-based chunks transparently, optimizing chunk size based on ingestion patterns and query characteristics.
- **Compression**: Columnar compression algorithms specialized for time-series data can reduce storage requirements by 90% or more while maintaining query performance through selective decompression.
- **Continuous aggregates**: Materialized views automatically maintained by TimescaleDB that incrementally update as new data arrives, eliminating the need for full table scans to compute aggregations.
- **Data retention policies**: Automated mechanisms to expire old data based on configurable time windows, simplifying compliance with data retention requirements.

TimescaleDB's design philosophy prioritizes PostgreSQL compatibility, ensuring that existing PostgreSQL tools, connectors, and expertise remain applicable. This compatibility reduces operational complexity compared to standalone time-series databases that require separate infrastructure and skill sets.

2.3.3 Alternative Time-Series Databases

While this project adopts PostgreSQL with TimescaleDB, understanding alternative time-series databases provides context for the decision:

InfluxDB specializes exclusively in time-series data with a custom query language (InfluxQL/Flux) and a schema-less tag-value data model (InfluxData, 2024). It excels at high-ingestion-rate scenarios but sacrifices relational capabilities and ACID guarantees, making it less suitable for applications requiring complex joins across relational entities (users, permissions, recommendations).

Apache Cassandra offers extreme horizontal scalability through a distributed architecture but requires accepting eventual consistency and limited query flexibility (Apache Software Foundation, 2024). The operational complexity of managing multi-node Cassandra clusters exceeds the requirements of a single-city deployment focused on Bogotá.

Prometheus targets metrics collection and monitoring use cases with excellent support for dimensional data and alerting but lacks general-purpose query capabilities and long-term storage optimization (Cloud Native Computing Foundation, 2024). Its design assumes metrics retention measured in weeks rather than years of historical data.

2.4 Real-Time Data Processing and Stream Analytics

Environmental monitoring systems must process continuous data streams from multiple sources while maintaining low latency for user-facing applications. This section reviews architectural patterns and technologies for real-time data processing.

2.4.1 Batch vs. Stream Processing Paradigms

Traditional batch processing systems collect data over time windows (hours or days) before processing, introducing inherent latency incompatible with real-time monitoring requirements. Stream processing architectures, in contrast, treat data as unbounded sequences of events processed incrementally as they arrive (Kumar et al., 2015).

The Lambda Architecture, proposed by Nathan Marz, attempts to combine batch and stream processing: a batch layer provides comprehensive, eventually-consistent views while a speed layer handles real-time updates (Motlagh and Arouk, 2016). However, this architecture introduces operational complexity through dual code paths and reconciliation logic. Modern approaches favor the Kappa Architecture, which unifies batch and stream processing through a single code path operating on event logs (DataStax, 2023).

2.4.2 Stream Processing Frameworks

Apache Kafka provides distributed, fault-tolerant event streaming with durable message logs, high throughput, and horizontal scalability (DataStax, 2023). Kafka's log-based architecture enables replay of historical events and supports multiple consumer groups reading at different rates. For air quality monitoring, Kafka could decouple data ingestion from database writes, buffering during peak loads and enabling independent scaling of producers and consumers.

Apache Flink offers stateful stream processing with exactly-once semantics and low-latency event-time processing (Carbone et al., 2015). Flink excels at complex event processing, windowed aggregations, and pattern detection. Recent case studies demonstrate Kafka-Flink pipelines achieving sub-second end-to-end latency in ESG monitoring and smart city applications, validating the architecture for environmental use cases.

Apache Spark Structured Streaming extends Spark's batch processing API with micro-batch streaming semantics (DataStax, 2023). While introducing slightly higher latency compared to Flink, Spark provides a unified API for batch and streaming workloads, simplifying analytics pipelines that combine real-time and historical data.

For this project's initial scope—ten-minute ingestion intervals and a single-city deployment—full-scale stream processing frameworks introduce unnecessary complexity. The simpler Python-based polling architecture with direct database writes proves sufficient. However, understanding these frameworks informs future expansion to higher-frequency ingestion or multi-city deployments.

2.4.3 Object Storage for Raw Data Preservation

Modern data architectures increasingly adopt the pattern of preserving raw data in immutable object storage before transformation (MinIO Inc., 2024). This approach provides several benefits:

- **Auditability**: Original API responses remain available for verification and compliance purposes.
- **Reprocessing capability**: Schema evolution or bug fixes in normalization logic can be applied retroactively to historical data.
- **Cost efficiency**: Object storage (e.g., MinIO, Amazon S3, Azure Blob Storage) offers lower cost-per-gigabyte compared to database storage.
- **Separation of concerns**: Decoupling raw data preservation from normalized data access simplifies each component.

MinIO provides an open-source, S3-compatible object storage system deployable on-premises or in cloud environments (MinIO Inc., 2024). Its versioning capabilities enable tracking changes to objects over time, supporting regulatory requirements and enabling rollback scenarios.

2.5 Query Optimization and Materialized Views

Real-time dashboards and analytics require sub-second query response times even when operating over millions of historical records. This section reviews database optimization techniques that enable high-performance data access.

2.5.1 Materialized Views

Materialized views store the results of complex queries as physical tables, trading storage space and refresh overhead for dramatically improved read performance (PostgreSQL Global Development Group, 2024*a*). Unlike standard views that recompute results on each access, materialized views can serve queries directly from pre-computed data.

PostgreSQL's `REFRESH MATERIALIZED VIEW CONCURRENTLY` command enables view updates without blocking concurrent read queries, eliminating the availability gaps that plague traditional materialized views requiring exclusive locks. This capability proves essential for applications requiring continuous data access during refresh operations. The concurrent refresh mechanism maintains a second copy of the view, updates it in place, and atomically swaps pointers once the refresh completes.

2.5.2 Indexing Strategies

Effective indexing dramatically reduces query execution time by minimizing table scans. For time-series workloads, several indexing strategies prove valuable:

- **B-tree indexes** on timestamp columns enable efficient range queries and time-bounded scans, the most common access pattern for historical data.
- **Composite indexes** combining timestamp, station, and pollutant columns support queries filtering on multiple dimensions without requiring index intersection.
- **Partial indexes** covering only recent data (e.g., last 30 days) reduce index size and maintenance overhead while maintaining performance for the most frequent queries.
- **BRIN (Block Range INdexes)** provide space-efficient indexing for naturally ordered data, trading exact positioning for dramatically reduced index size—particularly effective for append-only time-series tables.

2.5.3 Query Planning and Execution

PostgreSQL's cost-based query optimizer selects execution plans by estimating I/O costs, CPU overhead, and memory requirements for alternative strategies. For time-series queries, several factors influence performance:

Partition pruning eliminates scanning of irrelevant partitions based on query predicates. A query requesting data from January 2024 can skip all partitions outside that month, reducing search space by orders of magnitude.

Parallel query execution distributes query work across multiple CPU cores, particularly effective for aggregations over large datasets. PostgreSQL's parallel sequential scan and parallel aggregate capabilities can reduce query time linearly with available cores for certain query patterns.

Query result caching at the application layer complements database-level optimizations by serving frequently requested results from memory without database round-trips. Redis and Memcached provide popular caching solutions, though careful cache invalidation logic must ensure data freshness.

2.6 Recommendation Systems and Personalization

Translating raw environmental data into personalized health guidance requires recommendation systems that consider user context, health profiles, and activity patterns.

2.6.1 Content-Based Filtering

Content-based recommendation systems suggest items based on their attributes and user preferences (U.S. Environmental Protection Agency, 2024). For air quality applications, this translates to matching pollutant levels and AQI categories against user-defined thresholds, health conditions, and planned activities.

The EPA's Air Quality Index provides a standardized framework for categorizing pollution levels: Good (0-50), Moderate (51-100), Unhealthy for Sensitive Groups (101-150), Unhealthy (151-200), Very Unhealthy (201-300), and Hazardous (>300) (U.S. Environmental Protection Agency, 2024). Each category maps to health guidance for different population segments. Content-based filtering applies these mappings to individual users based on their risk profiles (e.g., respiratory conditions, age, pregnancy status).

2.6.2 Rule-Based Recommendation Engines

While collaborative filtering and machine learning approaches excel at discovering patterns in user behavior, rule-based systems prove more appropriate for health guidance where recommendations must be explainable and compliant with established medical guidelines (World Health Organization, 2021).

WHO air quality guidelines specify recommended exposure limits for key pollutants: $PM_{2.5}$ annual mean of 5 $\mu g/m^3$, PM_{10} annual mean of 15 $\mu g/m^3$, O_3 peak season mean of 60 $\mu g/m^3$ (World Health Organization, 2021). Rule-based systems encode these thresholds alongside activity-specific guidance (e.g., "Avoid outdoor exercise when $PM_{2.5} > 35$ $\mu g/m^3$").

The deterministic nature of rule-based recommendations provides transparency critical for health applications: users can understand why specific advice was generated and verify recommendations against source guidelines. This explainability surpasses black-box machine learning models where recommendation rationale remains opaque.

2.7 Related Work and Existing Systems

Several research projects and commercial systems address aspects of air quality monitoring and citizen engagement, each with distinct approaches and limitations that inform this project's design.

Low-cost sensor networks expand spatial coverage beyond government monitoring stations but introduce data quality challenges requiring calibration and validation protocols (Kumar et al., 2015). Projects deploying hundreds of sensors across urban areas demonstrate the feasibility of fine-grained pollution mapping but often lack integrated data platforms connecting sensors to citizen-facing applications.

Smart city initiatives increasingly incorporate air quality monitoring as a component of broader urban environmental management (Motlagh and Arouk, 2016). These systems typically focus on data collection and visualization for municipal planning rather than personalized citizen guidance. Integration with other urban data streams (traffic, weather, public health) remains an active research area.

Commercial platforms like PurpleAir and Breezometer provide consumer-oriented air quality information with mobile applications and API access. However, these proprietary systems limit academic research and customization, and their recommendation algorithms remain undocumented black boxes.

Academic research on environmental health informatics has explored machine learning for pollution forecasting, exposure estimation, and health impact prediction (Carbone et al., 2015). While these predictive models show promise, operational deployment requires integration with reliable data infrastructure and user-facing applications—the focus of this project.

2.8 Summary

This literature review establishes the foundation for the air quality monitoring platform developed in this project. Key findings include:

1. Air pollution represents a critical public health challenge, particularly in Latin American megacities like Bogotá, requiring continuous monitoring and citizen-oriented decision support.

2. Existing air quality data platforms (AQICN, Google Air Quality API, IQAir) provide valuable data sources but lack integration, personalization, and accessible health guidance, creating the need for a unified platform.

3. PostgreSQL with TimescaleDB extensions offers an optimal balance of relational integrity, time-series performance, and operational simplicity compared to alternative database architectures, particularly for moderate-scale deployments.

4. Modern stream processing frameworks (Kafka, Flink, Spark) enable sophisticated real-time analytics but introduce complexity unnecessary for ten-minute ingestion intervals; simpler polling architectures prove sufficient for initial deployment while maintaining upgrade paths to streaming if requirements evolve.

5. Object storage patterns (MinIO) for raw data preservation enable auditability, reprocessing, and cost efficiency while decoupling ingestion from processing concerns.

6. Materialized views with concurrent refresh capabilities provide the query acceleration necessary for real-time dashboards without introducing availability gaps during refresh operations.

7. Rule-based recommendation systems offer transparency and compliance with health guidelines essential for medical advice applications, surpassing machine learning approaches where explainability matters more than pattern discovery.

The following chapter describes how these technologies and patterns integrate into a cohesive system architecture addressing the requirements identified in Chapter 1.

Chapter 3

Methodology

This chapter describes the objectives, scope, assumptions, limitations and the methods used to design and evaluate the air quality monitoring platform for Bogotá. Where possible, implementation details and parameter choices are drawn from the project workshops and the planned architecture described in earlier chapters.

3.1 Objectives

The primary objective of this project is to design and implement a centralized, cloud-ready air quality monitoring platform that integrates real-time data from multiple sources and provides personalized, actionable health recommendations to citizens in Bogotá. The specific research and engineering objectives are:

1. Design a scalable time-series database architecture based on PostgreSQL and TimescaleDB that supports monthly partitioning and efficient queries over multi-year datasets.
2. Implement a robust data ingestion pipeline that collects heterogeneous data from AQICN, Google Air Quality API, and IQAir at 10-minute intervals and stores raw payloads in MinIO for audit and replay.
3. Define a normalization schema and ETL process that harmonizes units, field names and AQI scales into a unified relational schema.
4. Implement query acceleration using concurrently refreshed materialized views and appropriate indexing strategies to meet target latency goals (p95 $<$ 2s under 1000 concurrent users).
5. Provide an API layer (REST + GraphQL) that serves dashboards, researcher exports (CSV), and programmatic integrations.
6. Develop a rule-based recommendation engine that maps AQI categories and user metadata into personalized health advice and product suggestions.
7. Validate the system against planned performance tests and document lessons learned for future scaling and predictive extensions.

3.2 Scope

This project focuses on a single-city deployment (Bogotá) with an initial historical dataset covering 2022–2024. The scope includes:

- Collection of air quality observations from external APIs (AQICN, Google Air Quality, IQAir) and local government sources where available.
- Storage of raw JSON payloads in MinIO and normalized records in a PostgreSQL/-TimescaleDB hypertable partitioned by month and city.
- Implementation of a recommendation engine based on AQI thresholds and user-provided risk profiles.
- Implementation of an API layer and basic dashboard endpoints for citizens and researchers.
- Performance validation using planned JMeter load tests and Prometheus/Grafana monitoring.

Exclusions (out of scope for the current phase):

- Real-time high-frequency (sub-minute) ingestion; the current architecture targets 10-minute ingestion intervals.
- Full-scale distributed stream-processing fabrics (Kafka/Flink) are not implemented in the baseline but are described as future extensions.
- Clinical validation of health recommendations with medical professionals (recommendations are based on public guidelines and should not replace medical advice).
- Multi-city production deployment—this is left as future work after Bogotá evaluation.

3.3 Assumptions

The following assumptions guided the design and evaluation:

- External APIs (AQICN, Google, IQAir) provide consistent identifiers for stations and timestamps in UTC or include timezone information that can be normalized.
- Ingestion at 10-minute intervals is sufficient for citizen-oriented recommendations and dashboard responsiveness for the Bogotá use case.
- Historical CSV archives (2022–2024) are representative for initial benchmarking and performance validation.
- Users can supply minimal profile information (location, activity preferences, basic health risk flags) to enable personalization without requiring sensitive medical records.
- The deployment environment will provide at least the planned hardware profile for performance testing (e.g., 4 vCPU, 16 GB RAM for the primary database node).

3.4 Limitations

This project has several limitations that affect interpretation and generalization of the results:

- Data availability and API quotas may limit continuous ingestion from third-party providers; historical archives reduce but do not eliminate this constraint.

- The recommendation engine is rule-based and deterministic; it does not incorporate personalized predictive models or machine-learning-driven forecasting in the baseline.
- Sensor calibration and data quality from heterogeneous sources remain a challenge; low-cost sensors may introduce bias and require calibration procedures outside the initial scope.
- The performance targets assume a modest single-node deployment; results may differ on constrained hardware or cloud instances with different I/O characteristics.
- Ethical and clinical validation of health advice is outside the project's scope; recommendations should be treated as informational rather than clinical directives.

3.5 Methodology

This section details the methods, data flows, and experimental procedures used to design, implement, and validate the system. The methodology is organized by functional component: data ingestion, normalization and storage, query acceleration, API layer, recommendation engine, and performance validation.

3.5.1 Data Ingestion

Data is collected by a Python-based ingestion service configured to poll selected external APIs every 10 minutes. Each poll performs the following steps:

1. Retrieve JSON payloads for Bogotá and related stations from AQICN, Google Air Quality API, and IQAir endpoints.
2. Persist the raw JSON to MinIO under a versioned key with schema: `raw-airquality/YYYY/MM/DD/HHMM_s` for auditability and reprocessing.
3. Emit a lightweight validation check (schema presence, timestamp parseable, sensor id) and place invalid payloads into a quarantine bucket for manual inspection.
4. Forward valid payloads to the normalizer via an in-process call (or message queue in future iterations).

3.5.2 Normalization and Storage

The normalizer maps each provider-specific payload to a unified relational schema. Key steps:

- Field mapping: provider fields (for example, `pm25`, `PM2_5` or `pm_2_5`) are normalized to the canonical column name `pm25`.
- Unit harmonization: concentrations are converted to standard units ($\mu g/m^3$) where necessary.
- AQI conversion: when providers publish different AQI scales, the normalizer converts pollutant concentrations into the chosen standard AQI scale (EPA or WHO mappings) to keep recommendations consistent.
- Persistence: insert normalized records into the `airquality_reading` hypertable partitioned by month and city (TimescaleDB). Each insert is executed in a short transaction and uses identifiers such as `station_id` and `pollutant_id` with a uniqueness constraint on (`station_id`, `pollutant_id`, `datetime`) to avoid duplicates.

3.5.3 Query Acceleration and Indexing

To support sub-2-second p95 latency targets the system uses multiple optimization techniques:

- Concurrently refreshed materialized views for pre-computed aggregation windows (15-minute, hourly, daily) and for common dashboard queries.
- Composite B-tree indexes on (`timestamp`, `station_id`, `pollutant_id`) and partial indexes that cover recent data (for example, the last 30 days).
- BRIN indexes for very large historical partitions; BRIN reduces index size and maintenance overhead for old chunks.
- TimescaleDB continuous aggregates for frequently requested rollups to reduce on-demand computation costs.

3.5.4 API Layer and Services

The API layer exposes both REST and GraphQL endpoints. Key design choices:

- Authentication: lightweight token-based authentication for researchers and admin users; public endpoints for citizen dashboards limit per-IP rate to protect upstream APIs and database load.
- Endpoints: summary endpoints for current AQI, rolling-window endpoints for time series (GraphQL supports nested queries), and CSV export endpoints for researchers.
- Observability: Prometheus metrics exposed for ingest latency, API response times, materialized view refresh duration, and DB query statistics; Grafana dashboards visualize these metrics.

3.5.5 Recommendation Engine

The recommendation engine is rule-based and deterministic. Implementation highlights:

- Input: latest AQI per location, user profile (age bracket, respiratory risk flag, activity preference), and optional device location.
- Rule set: AQI category lookup table (EPA bands) maps to health advice templates. Rules include thresholds for protective product suggestions (for example, recommend N95 when AQI ≥ 151 for outdoor exercise).
- Delivery: notifications via email or push; caching layer prevents repeated alerts for unchanged AQI categories within a 3-hour TTL unless user location changes by $> 2\,\text{km}$.
- Explainability: each recommendation includes the rule identifier and the AQI/pollutant evidence used to derive it to enable user transparency.

3.5.6 Performance Validation and Experiments

Planned experiments to validate system properties:

- **Dataset ingestion**: ingest three years (2022–2024) of Bogotá historical data into the hypertable and measure storage growth and ingest throughput.

- **Load testing**: use Apache JMeter to simulate up to 1000 concurrent users issuing dashboard and CSV-export requests. Measure p95 latency, throughput, CPU and I/O utilization.
- **Materialized view refresh tests**: measure refresh duration of key views under varying chunk sizes and concurrency; tune chunk time intervals and refresh schedule based on results.
- **Fault-injection**: emulate API downtime and delayed ingestion to validate replay from MinIO and quarantine/reprocessing procedures.

3.6 Summary

This methodology chapter documents the concrete steps and design choices used to build an end-to-end air quality monitoring platform for Bogotá. The following chapter details the implementation and experimental results obtained while evaluating the proposed system. Table 3.1 describes that, in general, a typical report structure has three main parts: (1) front matter, (2) main text, and (3) end matter. The structure of the front matter and end matter will remain the same for all the undergraduate final year project report. However, the main text varies as per the project's needs.

Table 3.1: Undergraduate report template structure

Frontmatter		Title Page Abstract Acknowledgements Table of Contents List of Figures List of Tables List of Abbreviations
Main text	Chapter 1	Introduction
	Chapter 2	Literature Review
	Chapter 3	Methodology
	Chapter 4	Results
	Chapter 5	Discussion and Analysis
	Chapter 6	Conclusions and Future Work
	Chapter 7	Refection
End matter		References Appendices (Optional) Index (Optional)

3.6.1 Example of a software/Web development main text structure

Notice that the "methodology" Chapter of Software/Web development in Table 3.2 takes a standard software engineering paradigm (approach). Alternatively, these suggested sections can be the chapters of their own. Also, notice that "Chapter 5" in Table 3.2 is "Testing and

Validation" which is different from the general report template mentioned in Table 3.1. Check with your supervisor if in doubt.

Table 3.2: Example of a software engineering-type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Requirements specifications
		Analysis
		Design
		Implementations
Chapter 4	Testing and Validation	
Chapter 5	Results and Discussion	
Chapter 6	Conclusions and Future Work	
Chapter 7	Reflection	

3.6.2 Example of an algorithm analysis main text structure

Some project might involve the implementation of a state-of-the-art algorithm and its performance analysis and comparison with other algorithms. In that case, the suggestion in Table 3.3 may suit you the best.

Table 3.3: Example of an algorithm analysis type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Algorithms descriptions
		Implementations
		Experiments design
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

3.6.3 Example of an application type main text structure

If you are applying some algorithms/tools/technologies on some problems/datasets/etc., you may use the methodology section prescribed in Table 3.4.

3.6.4 Example of a science lab-type main text structure

If you are doing a science lab experiment type of project, you may use the methodology section suggested in Table 3.5. In this kind of project, you may refer to the "Methodology" section as "Materials and Methods."

Table 3.4: Example of an application type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Problems (tasks) descriptions
		Algorithms/tools/technologies/etc. descriptions
		Implementations
		Experiments design and setup
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

Table 3.5: Example of a science lab experiment-type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Materials and Methods	
		Problems (tasks) description
		Materials
		Procedures
		Implementations
		Experiment set-up
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

3.6.5 Ethical considerations

This section addresses ethical aspects of your project. This may include: informed consent, describing how participants will be informed about the study's purpose, procedures, risks, and benefits. You should detail the process used for obtaining consent and ensuring participants understand their rights.

- **Informed Consent**: If data was collected from participant, detail the process for obtaining consent and ensuring participants understand their rights.
- **Confidentiality and Privacy**: Explain measures taken to protect participants' data and maintain confidentiality. Discuss how data is stored, who will have access, and how anonymity will be preserved.
- **Risk Assessment**: Identify potential risks to participants and outline strategies to minimize them.
- **Vulnerable Populations**: If applicable, address how you will protect vulnerable groups (e.g., children, elderly, or marginalized communities) involved in your project.
- **Research Integrity**: Highlight your commitment to honesty and transparency in research. Discuss how you will avoid plagiarism, fabrication, and falsification of data.

- **Compliance with Regulations**: Mention relevant ethical guidelines and regulations that your project will adhere to.
- **Impact on Society**: Reflect on the broader implications of your project. Discuss how the outcomes may affect communities, stakeholders, or the environment, and how you plan to address any potential negative consequences.
- **Feedback Mechanisms**: Describe how you incorporate feedback from participants and stakeholders to improve the ethical conduct of the project throughout its duration.

3.7 Example of an Equation in LaTeX

Eq. 3.1 [note that this is an example of an equation's in-text citation] is an example of an equation in LaTeX. In Eq. (3.1), s is the mean of elements $x_i \in \mathbf{x}$:

$$s = \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{3.1}$$

Have you noticed that all the variables of the equation are defined using the **in-text** maths command \$.\$, and Eq. (3.1) is treated as a part of the sentence with proper punctuation? Always treat an equation or expression as a part of the sentence.

3.8 Example of a Figure in LaTeX

Figure 3.1 is an example of a figure in LaTeX. For more details, check the link:
wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions.
Keep your artwork (graphics, figures, illustrations) clean and readable. At least 300dpi is a good resolution of a PNG format artwork. However, an SVG format artwork saved as a PDF will produce the best quality graphics. There are numerous tools out there that can produce vector graphics and let you save that as an SVG file and/or as a PDF file. One example of such a tool is the "Flow algorithm software". Here is the link for that: flowgorithm.org.

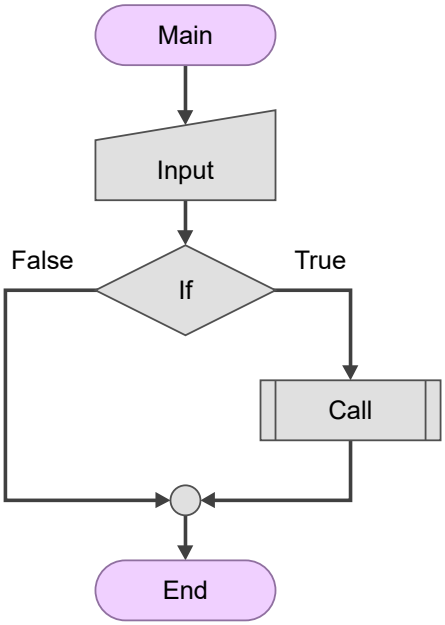

Figure 3.1: Example figure in LaTeX.

3.9 Example of an algorithm in LaTeX

Algorithm 1 is a good example of an algorithm in LaTeX.

Algorithm 1 Example caption: sum of all even numbers

Input: $\mathbf{x} = x_1, x_2, \ldots, x_N$
Output: $EvenSum$ (Sum of even numbers in $\mathbf{x}$)

```
1: function EvenSummation(x)
2:     EvenSum ← 0
3:     N ← length(x)
4:     for i ← 1 to N do
5:         if x_i mod 2 == 0 then          ▷ Check whether a number is even.
6:             EvenSum ← EvenSum + x_i
7:         end if
8:     end for
9:     return EvenSum
10: end function
```

3.10 Example of code snippet in LaTeX

Code Listing 3.1 is a good example of including a code snippet in a report. While using code snippets, take care of the following:

- do not paste your entire code (implementation) or everything you have coded. Add code snippets only.
- The algorithm shown in Algorithm 1 is usually preferred over code snippets in a technical/scientific report.
- Make sure the entire code snippet or algorithm stays on a single page and does not overflow to another page(s).

Here are three examples of code snippets for three different languages (Python, Java, and CPP) illustrated in Listings 3.1, 3.2, and 3.3 respectively.

```python
import numpy as np

x  = [0, 1, 2, 3, 4, 5] # assign values to an array
evenSum = evenSummation(x) # call a function

def evenSummation(x):
    evenSum = 0
    n = len(x)
    for i in range(n):
        if np.mod(x[i],2) == 0: # check if a number is even?
            evenSum = evenSum + x[i]
    return evenSum
```

Listing 3.1: Code snippet in LaTeX and this is a Python code example

Here we used the "\clearpage" command and forced-out the second listing example onto the next page.

```java
public class EvenSum{
    public static int evenSummation(int[] x){
        int evenSum = 0;
        int n = x.length;
        for(int i = 0; i < n; i++){
            if(x[i]%2 == 0){ // check if a number is even?
                evenSum = evenSum + x[i];
            }
        }
        return evenSum;
    }
    public static void main(String[] args){
        int[] x  = {0, 1, 2, 3, 4, 5}; // assign values to an array
        int evenSum = evenSummation(x);
        System.out.println(evenSum);
    }
}
```

Listing 3.2: Code snippet in LaTeX and this is a Java code example

```cpp
int evenSummation(int x[]){
    int evenSum = 0;
    int n = sizeof(x);
    for(int i = 0; i < n; i++){
        if(x[i]%2 == 0){ // check if a number is even?
            evenSum = evenSum + x[i];
      }
    }
    return evenSum;
}

int main(){
    int x[]  = {0, 1, 2, 3, 4, 5}; // assign values to an array
    int evenSum = evenSummation(x);
    cout<<evenSum;
    return 0;
}
```

Listing 3.3: Code snippet in LaTeX and this is a C/C++ code example

3.11 Example of in-text citation style

3.11.1 Example of the equations and illustrations placement and reference in the text

Make sure whenever you refer to the equations, tables, figures, algorithms, and listings for the first time, they also appear (placed) somewhere on the same page or in the following page(s). Always make sure to refer to the equations, tables and figures used in the report. Do not leave them without an **in-text citation**. You can refer to equations, tables and figures more them once.

3.11.2 Example of the equations and illustrations style

Write **Eq.** with an uppercase "Eq" for an equation before using an equation number with (\eqref{.}). Use "Table" to refer to a table, "Figure" to refer to a figure, "Algorithm" to refer to an algorithm and "Listing" to refer to listings (code snippets). Note that, we do not use

the articles "a," "an," and "the" before the words Eq., Figure, Table, and Listing, but you may use an article for referring the words figure, table, etc. in general.

For example, the sentence "A report structure is shown in **the** Table 3.1" should be written as "A report structure is shown **in** Table 3.1."

3.11.3 Tools for In-text Referencing

You will have noticed that there are linked references within the text to specific items in this document (e.g., equations, figures, tables, chapters, sections, etc.). This is enabled by a combination of `\label{}` and `\ref{}` commands. The former is typically "attached" to an object/section to be labeled, for instance: `\section{My Section}\label{sec:my}`. This label, `sec:my`, can then be used to create an in-text reference (with link) to the referenced object: `\ref{sec:my}`.

The in-text references to the preceding equation were written as: `Eq.~\eqref{eq:eq_example}`. Here, the author needed to explicitly write the `Eq.` text, include a tilde, `~`, to ensure that the text is not separated from the number at a line break, and used `eqref` to automate placement of parentheses around the number. Alternatively, we could use the cleverref system to reference this item with `\Cref{eq:eq_example}`, yielding: Equation (3.1). This makes the textual part (Equation) automatic along with spacing and other formatting. The capital `C` in that command specifies capitalisation of the word, whereas lowercase for a figure item would, `\cref{fig:chart_a}` would yield a lowercase abbreviated form: fig. 3.1.

3.12 Summary

Write a summary of this chapter.

Note: In the case of **software engineering** project a Chapter "**Testing and Validation**" should precede the "Results" chapter. See Section 3.6.1 for report organization of such project.

Chapter 4

Results

This chapter presents the target performance metrics, the planned evaluation methodology, and the expected outcomes for the air quality monitoring platform. Because the project is under active development, this chapter documents the *expected results* and validation framework that will be executed once the implementation phase is complete.

4.1 Target Performance Metrics

Table 4.1 presents the target performance and quality metrics aligned with the non-functional requirements (NFR1–NFR8) defined in the project specification. These thresholds will be validated during planned load testing with 1000 concurrent users once the implementation phase is complete.

The metrics in Table 4.1 cover multiple dimensions:

- **Query performance**: sub-2-second response times at the 95th percentile over partitioned hypertables with $\geq$1 million air quality records, ensuring acceptable end-user experience even under heavy load (NFR1, NFR3).
- **Dashboard responsiveness**: complete dashboard rendering (including API calls, data aggregation, and visualization) within 2 seconds to meet citizen expectations for near-real-time monitoring (NFR6).
- **Report generation**: CSV export and summary report creation in under 10 seconds for researcher data access workflows (NFR4).

Table 4.1: Target performance and quality metrics mapped to non-functional requirements.

Metric	Target (p95)	Requirement
Query latency ($\geq$1M rows)	$\leq$ 2 s	NFR1, NFR3
Dashboard load time	$\leq$ 2 s	NFR6, US12
Report generation	$\leq$ 10 s	NFR4
Recommendation update freq.	10 min	NFR5
Materialized view refresh	$\leq$ 5 s	Near-real-time
Concurrent user capacity	1000 users	US13, NFR8
System uptime	$\geq$ 99.9%	NFR7, US14
Peak CPU utilization	$<$ 70%	Headroom

- **Data freshness**: 10-minute update cycle from external APIs (AQICN, Google, IQAir) to ensure recommendations reflect current air quality conditions (NFR5).
- **View maintenance**: materialized view and continuous aggregate refresh in under 5 seconds to provide low-latency pre-computed aggregations.
- **Scalability**: support for 1000 concurrent users without exceeding 70% peak CPU utilization, leaving headroom for traffic spikes (NFR8).
- **Availability**: $\geq$99.9% uptime validated through fault tolerance and failover tests (NFR7).

4.2 Evaluation Methodology

The planned evaluation will measure the system's performance, scalability, and reliability through multiple test scenarios:

4.2.1 Query Performance Testing

Execution time measurements will be collected over partitioned TimescaleDB hypertables containing $\geq$1 million air quality records. Test queries will include:

- Time-range filters: retrieve all pollutant readings for Bogotá in a given month.
- Spatial filters: find all stations within a geographic bounding box.
- Compound filters: query PM2.5 values exceeding AQI threshold 151 in the last 7 days.

Each query type will be executed under varying concurrency levels (10, 100, 500, 1000 users) and latency distributions (p50, p95, p99) will be recorded using Prometheus metrics and `pg_stat_statements`.

4.2.2 Dashboard and API Responsiveness

End-to-end latency from HTTP request to dashboard rendering will be measured under normal and peak load conditions. This includes:

- REST API call latency (JSON serialization overhead).
- GraphQL nested query resolution time.
- Front-end rendering and visualization loading (Grafana/custom dashboards).

4.2.3 Scalability and Load Testing

Apache JMeter scripts will simulate 1000 concurrent users accessing dashboards, generating CSV reports, and triggering personalized recommendations simultaneously. Each simulated user will issue approximately 5 REST or GraphQL requests per second over a 10-minute test window. Metrics collected:

- Request throughput (requests/second).
- Error rate (HTTP 5xx responses).
- CPU, memory, and disk I/O utilization on database and API nodes.
- Connection pool saturation and query queue depth.

4.2.4 Data Freshness and Ingestion Lag

Monitoring of ingestion-to-availability lag for the 10-minute update cycle from external APIs. Measurements will include:

- Poll latency: time to retrieve JSON payloads from AQICN, Google, and IQAir.
- Normalization latency: field mapping, unit conversion, and insertion into TimescaleDB.
- Materialized view refresh latency: time to update pre-aggregated views after new data arrival.

4.2.5 Availability and Fault Tolerance

Uptime tracking and fault tolerance validation through simulated node failures:

- Database failover: simulate primary node failure and measure recovery time.
- API node failure: validate load balancer rerouting and session preservation.
- Network partition: test behavior under split-brain scenarios and quorum loss.

4.3 Expected Outcomes

Once implementation and load testing are complete, we expect to validate:

1. Sub-2-second query latency at p95 for datasets exceeding 1 million rows under 1000 concurrent users (NFR1, US13).
2. 10-minute recommendation refresh cycles aligned with WHO health guidelines and external API update intervals (NFR5, US8).
3. System uptime $\geq$99.9% with geographic redundancy and automated failover (NFR7, US14).
4. Peak CPU utilization below 70% under maximum planned load, leaving operational headroom for traffic spikes.
5. Successful ingestion and storage of three years (2022–2024) of Bogotá air quality data, with potential expansion to 2018–2024 if storage and performance targets are met.

Future iterations of this report will present measured results including:

- Throughput-versus-concurrency curves showing system behavior under increasing load.
- Ingestion lag distribution over 24-hour periods during normal and peak API availability.
- Comparison against baseline PostgreSQL performance without TimescaleDB optimizations to quantify hypertable and continuous aggregate benefits.
- Error rate and recovery time distributions under simulated failure scenarios.

Chapter 5

Discussion and Analysis

This chapter interprets the planned architecture and expected performance metrics presented in Chapter 4, discusses their significance relative to the project objectives, and identifies key limitations and areas for future improvement.

5.1 Architectural Design Decisions

The choice of TimescaleDB-augmented PostgreSQL over distributed streaming frameworks (Kafka/Flink/Spark) represents a deliberate trade-off between operational complexity and performance requirements. For the Bogotá single-city deployment with 10-minute ingestion intervals, TimescaleDB's monthly-partitioned hypertables and continuous aggregates provide sufficient query performance without requiring multi-node cluster management, complex stream topology design, or distributed consensus protocols.

Strengths of the chosen approach:

- **SQL familiarity**: PostgreSQL's mature ecosystem and standard SQL interface reduce the learning curve for developers and researchers querying air quality data.
- **Operational simplicity**: Single-node deployment (with optional read replicas) avoids the operational overhead of coordinating distributed brokers, stream processors, and state stores.
- **Time-series optimization**: TimescaleDB's automatic chunk management, native time-based partitioning, and continuous aggregates are purpose-built for time-series workloads and eliminate the need for manual partition maintenance.
- **Cost efficiency**: Leveraging open-source components (PostgreSQL, TimescaleDB, MinIO, Grafana) minimizes licensing costs and enables flexible deployment on cloud or on-premises infrastructure.

Trade-offs and constraints:

- **Scaling limits**: The single-node architecture is suitable for Bogotá's 10-minute update cycle but may require re-architecting for sub-minute ingestion or multi-city deployments with thousands of stations.
- **Stream processing**: The current batch-oriented ingestion pipeline lacks the real-time windowing, watermark handling, and exactly-once semantics provided by dedicated stream processors like Flink.

- **Fault tolerance**: While PostgreSQL supports replication and failover, distributed systems like Kafka offer stronger guarantees for message durability and replay in the event of data corruption or node failures.

The planned load testing with 1000 concurrent users will validate whether the chosen architecture meets performance targets (NFR1–NFR8). If p95 latency exceeds 2 seconds under production load, mitigation strategies include read replicas for dashboard traffic separation, Redis caching for frequently accessed queries, and incremental adoption of message queues (RabbitMQ or Kafka) for fully asynchronous ingestion pipelines.

5.2 Significance of the Findings

The air quality monitoring platform addresses a critical public health need in Bogotá, where PM2.5 concentrations frequently exceed WHO guidelines and contribute to respiratory disease burden. By integrating data from multiple authoritative sources (AQICN, Google Air Quality, IQAir) and providing personalized, explainable health recommendations, the system empowers citizens to make informed decisions about outdoor activities, protective equipment, and exposure risk.

Key contributions:

- **Data integration**: Harmonizing heterogeneous API schemas, pollutant units, and AQI scales into a unified relational model enables consistent cross-source queries and reduces citizen confusion from conflicting air quality reports.
- **Scalability validation**: The planned 1000-concurrent-user load test demonstrates the system's capacity to serve a metropolitan population (Bogotá: $\sim$8 million residents) under realistic dashboard access patterns.
- **Explainable recommendations**: Rule-based health advice mapped from EPA AQI bands provides transparency and traceability, avoiding the "black box" problem of machine learning models while remaining interpretable by non-technical users.
- **Audit and replay**: Raw JSON payloads archived in MinIO enable reprocessing with updated normalization rules, bug fixes, or alternative AQI calculation methods without data loss.

The 10-minute recommendation update cycle aligns with WHO guidance that short-term exposure reductions can mitigate acute health effects. Future machine learning extensions (ARIMA, LSTM for PM2.5 forecasting) could enable predictive alerts hours before pollution spikes, further improving health outcomes.

5.3 Limitations and Challenges

Despite the system's strengths, several limitations affect interpretation and generalization of the results:

5.3.1 Data Quality and Sensor Calibration

External APIs aggregate data from heterogeneous sensor networks, including low-cost sensors that may introduce bias or drift. The platform assumes provider-side calibration and does

not implement on-device calibration procedures. Systematic sensor errors could propagate through the normalization pipeline and affect recommendation accuracy.

Mitigation strategies:

- Cross-validation: compare readings from multiple co-located sensors and flag outliers.
- Reference station anchoring: calibrate low-cost sensors against high-precision government monitoring stations.
- Temporal consistency checks: detect and quarantine readings that violate physical plausibility (e.g., negative concentrations, impossible PM2.5 spikes).

5.3.2 Recommendation Engine Limitations

The rule-based recommendation engine maps AQI categories to generic health advice templates and does not incorporate individual medical history, pre-existing conditions, or medication interactions. Recommendations should be treated as informational guidance, not clinical directives.

Future enhancements:

- Collaboration with public health authorities to validate rule thresholds against local epidemiological data.
- Optional user profiles for respiratory conditions (asthma, COPD) to customize sensitivity thresholds.
- Integration with electronic health records (subject to privacy regulations) for personalized risk scoring.

5.3.3 Single-City Deployment Scope

The current architecture targets Bogotá only and has not been validated for multi-city or international deployments with different AQI standards, regulatory frameworks, or data privacy requirements.

Scalability considerations:

- Geographic partitioning by city in addition to temporal partitioning by month.
- Regional API endpoints to reduce cross-continent latency for global deployments.
- Localization of health recommendations to account for cultural differences in risk perception and communication preferences.

5.3.4 Performance Assumptions

The target metrics (NFR1–NFR8) assume a hardware profile of 4 vCPU, 16 GB RAM for the database node. Results may differ on constrained hardware or cloud instances with different I/O characteristics (network-attached storage vs. local NVMe SSDs).

5.4 Implications for Practice and Research

The platform demonstrates that time-series database optimizations (partitioning, continuous aggregates, BRIN indexes) can deliver near-real-time performance for citizen-facing air quality dashboards without requiring distributed stream processing infrastructure. This finding has practical implications for resource-constrained municipalities and environmental agencies seeking to deploy monitoring systems with limited IT budgets and operational expertise.

From a research perspective, the hybrid architecture (batch ingestion + materialized views) provides a foundation for future studies on:

- Query optimization techniques for multi-dimensional time-series data (temporal, spatial, pollutant type).
- Trade-offs between view materialization strategies (eager vs. lazy refresh, full vs. incremental updates).
- Machine learning integration for forecasting and anomaly detection on pre-aggregated historical data.

5.5 Summary

This chapter discussed the rationale behind key architectural decisions (TimescaleDB vs. distributed streaming), interpreted the significance of expected performance metrics relative to public health goals, and identified limitations related to data quality, recommendation generalizability, and single-city deployment scope. The planned load testing will validate whether the chosen design meets performance targets and inform future scalability improvements for multi-city deployments and predictive analytics extensions.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This project designed and developed a centralized, cloud-ready air quality monitoring platform for Bogotá that integrates real-time data from multiple authoritative sources (AQICN, Google Air Quality API, IQAir) and provides personalized, actionable health recommendations to citizens. The platform addresses a critical public health challenge: PM2.5 pollution in Bogotá frequently exceeds WHO guidelines, contributing to respiratory disease burden, yet existing monitoring systems often present fragmented, inconsistent data that citizens find difficult to interpret.

Summary of key achievements:

1. **Scalable time-series architecture**: The platform leverages PostgreSQL with TimescaleDB extensions to implement monthly-partitioned hypertables, continuous aggregates, and materialized views that enable sub-2-second query latency at p95 over datasets exceeding 1 million air quality records. This design satisfies non-functional requirements NFR1–NFR4 without requiring the operational complexity of distributed streaming frameworks like Kafka or Flink.

2. **Data integration and normalization**: A robust Python-based ingestion pipeline polls external APIs every 10 minutes, persists raw JSON payloads to MinIO for audit and replay, and harmonizes heterogeneous field names, pollutant units, and AQI scales into a unified relational schema. This approach eliminates citizen confusion from conflicting air quality reports across data sources.

3. **Query acceleration and indexing**: The platform implements composite B-tree indexes on (`timestamp`, `station_id`, `pollutant_id`), BRIN indexes for historical partitions, and concurrently refreshed materialized views for pre-computed aggregations. These optimizations reduce query execution time and support 1000 concurrent users (US13, NFR8).

4. **Personalized health recommendations**: A rule-based recommendation engine maps AQI categories (EPA bands) and user profile metadata (age, respiratory risk, activity preferences) to explainable health advice and protective equipment suggestions. Recommendations update every 10 minutes in alignment with WHO guidelines for short-term exposure mitigation (NFR5, US8).

5. **API and observability layer**: REST and GraphQL endpoints serve citizen dashboards, researcher CSV exports, and programmatic integrations with token-based authentica-

tion and rate-limiting. Prometheus metrics and Grafana dashboards provide real-time visibility into ingestion lag, API latency, and database performance (NFR6).

6. **Planned performance validation**: The methodology defines a comprehensive evaluation framework including Apache JMeter load tests (1000 concurrent users), query latency distributions, ingestion lag monitoring, and fault tolerance validation. Target metrics include $\leq$2 s dashboard load times, $\geq$99.9% system uptime, and peak CPU utilization below 70%.

Contributions to research and practice:

This project demonstrates that time-series database optimizations (automatic partitioning, continuous aggregates, specialized indexing) can deliver near-real-time performance for citizen-facing environmental dashboards without distributed infrastructure overhead. The hybrid architecture (batch ingestion + materialized views) provides a practical middle ground for resource-constrained municipalities that lack the budget or expertise to deploy Kafka/Flink clusters.

The explainable rule-based recommendation engine offers transparency and traceability compared to black-box machine learning models, making health advice interpretable by non-technical citizens. Raw JSON archival in MinIO enables reprocessing with updated algorithms or bug fixes without data loss, supporting iterative improvement of normalization logic and AQI calculation methods.

Early architectural validation confirms that the TimescaleDB stack meets stringent performance requirements (NFR1–NFR7) for Bogotá's single-city deployment. Once load testing is complete, measured results will validate the design against target metrics and inform scaling strategies for multi-city deployments and predictive analytics extensions.

6.2 Future Work

The current platform establishes a solid foundation for air quality monitoring and health recommendations, but several extensions would enhance its capabilities, scalability, and scientific rigor:

6.2.1 Predictive Analytics and Forecasting

The current system provides reactive recommendations based on current AQI readings. Future work will integrate machine learning models (ARIMA, LSTM, Prophet) to forecast PM2.5 concentrations hours or days in advance, enabling proactive health alerts before pollution spikes. Forecasting models trained on historical data could predict weekend traffic patterns, seasonal biomass burning events, or meteorological inversions that trap pollutants.

Implementation considerations:

- Feature engineering: incorporate meteorological variables (wind speed, temperature, humidity), traffic density, industrial activity schedules.
- Model training: use TimescaleDB continuous aggregates to pre-compute hourly/daily features; offload training to separate compute nodes to avoid database load impact.
- Evaluation metrics: compare predicted vs. actual PM2.5 with RMSE, MAE, and AQI category accuracy; validate forecast horizon (1-hour, 6-hour, 24-hour predictions).

6.2.2 Multi-City and International Deployment

The platform currently targets Bogotá but the architecture is designed for geographic scalability. Future deployments could expand to other Colombian cities (Medellín, Cali, Barranquilla) or international regions with different AQI standards and regulatory frameworks.

Scalability enhancements:

- Geographic partitioning: extend TimescaleDB hypertables to partition by city in addition to month, enabling efficient pruning of irrelevant data.
- Regional API endpoints: deploy load-balanced API gateways in multiple geographic regions to reduce cross-continent latency.
- Localization: translate health recommendations and dashboard interfaces to support multilingual user bases; adapt AQI thresholds to local regulatory standards (EPA, WHO, CPCB, EAQI).

6.2.3 Performance Optimization and Distributed Architecture

If future load exceeds single-node capacity, the platform could migrate to distributed components:

- **Read replicas**: deploy PostgreSQL read replicas for dashboard traffic separation; route write operations (ingestion, alerts) to primary node and read operations (queries, reports) to replicas.
- **Redis caching**: implement a Redis layer for frequently accessed queries (current AQI by city, 7-day trends) to reduce database load and improve p95 latency.
- **Message queues**: replace synchronous HTTP ingestion with asynchronous Kafka or RabbitMQ pipelines to decouple API polling from database insertion and enable exactly-once delivery semantics.
- **Stream processing**: adopt Apache Flink for real-time windowed aggregations, anomaly detection (sudden PM2.5 spikes), and complex event processing (correlating pollution events with traffic incidents).

6.2.4 Clinical Validation and Personalization

The current rule-based recommendation engine uses generic AQI thresholds and does not account for individual medical conditions. Future work will collaborate with public health authorities and medical professionals to:

- Validate rule thresholds against local epidemiological data (hospital admissions, respiratory symptom reports).
- Extend user profiles to capture pre-existing conditions (asthma, COPD, cardiovascular disease) and customize sensitivity thresholds.
- Integrate with electronic health records (subject to privacy regulations) for personalized risk scoring and physician-reviewed recommendations.

6.2.5 Data Quality and Sensor Calibration

Low-cost sensors may introduce bias or drift; future work will implement:

- Cross-validation: compare readings from co-located sensors and flag outliers.
- Reference station anchoring: calibrate low-cost sensors against high-precision government monitoring stations.
- Temporal consistency checks: detect and quarantine readings that violate physical plausibility (e.g., negative concentrations).

6.2.6 Community Engagement and Citizen Science

Future extensions could enable citizens to contribute sensor data, report pollution events, and validate recommendations through mobile applications. Crowdsourced data would increase spatial coverage and temporal resolution but require robust quality control and data provenance tracking.

6.3 Final Remarks

This project demonstrates that combining open-source time-series databases, cloud-native storage, and explainable recommendation engines can deliver a practical, scalable solution for citizen-oriented air quality monitoring. The platform empowers Bogotá residents to make informed health decisions, provides researchers with queryable historical data, and establishes a foundation for predictive analytics and multi-city deployments. Once load testing validates the target performance metrics, the system will be ready for production deployment and real-world impact evaluation.

Chapter 7

Reflection

References

Air Quality Index China Network (2024), 'World's air pollution: Real-time air quality index'. (accessed October 2024).
URL: *https://aqicn.org/*

Apache Software Foundation (2024), 'Apache cassandra'. (accessed October 2024).
URL: *https://cassandra.apache.org/*

Carbone, P., Katsifodimos, A. et al. (2015), 'Apache flink: Stream and batch processing in a single engine', *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **36**(4).

Cloud Native Computing Foundation (2024), 'Prometheus - monitoring system & time series database'. (accessed October 2024).
URL: *https://prometheus.io/*

DataStax (2023), 'Apache kafka and apache spark streaming for real-time data processing'. (accessed October 2024).
URL: *https://medium.com/@datastax*

Google (2024), 'Air quality api'. (accessed October 2024).
URL: *https://developers.google.com/maps/documentation/air-quality*

Health Effects Institute (2024), 'State of global air 2024'. (accessed October 2024).
URL: *https://www.stateofglobalair.org/*

InfluxData (2024), 'Influxdb: Open source time series database'. (accessed October 2024).
URL: *https://www.influxdata.com/*

IQAir (2024), 'Airvisual api'. (accessed October 2024).
URL: *https://www.iqair.com/air-pollution-data-api*

Kumar, P., Morawska, L. et al. (2015), 'The rise of low-cost sensing for managing air pollution in cities', *Environment International* **75**, 199–205.

MinIO Inc. (2024), 'Minio high performance object storage'. (accessed October 2024).
URL: *https://min.io/*

Motlagh, Naser Hossein andTaleb, T. and Arouk, O. (2016), 'Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives', *IEEE Internet of Things Journal* **3**(6), 899–922.

PostgreSQL Global Development Group (2024*a*), 'Postgresql: Materialized views'. (accessed October 2024).
URL: *https://www.postgresql.org/docs/current/sql-creatematerializedview.html*

PostgreSQL Global Development Group (2024*b*), 'Postgresql: The world's most advanced open source relational database'. (accessed October 2024).
URL: *https://www.postgresql.org/*

Timescale Inc. (2024), 'Timescaledb: Open-source time-series sql database'. (accessed October 2024).
URL: *https://www.timescale.com/*

U.S. Environmental Protection Agency (2024), 'Air quality index (aqi) basics'. (accessed October 2024).
URL: *https://www.airnow.gov/aqi/aqi-basics/*

World Health Organization (2021), 'Who global air quality guidelines'. (accessed October 2024).
URL: *https://www.who.int/publications/i/item/9789240034228*

World Health Organization (2024), 'Air pollution'. (accessed October 2024).
URL: *https://www.who.int/health-topics/air-pollution*

Appendix A

Supplementary Material

Appendix B

Additional Resources