Clean Air COMPASS DMS – System Design Plan

1. Project Overview

The Community of Practice for Air Quality Systems (COMPASS) is developing an open-source, customizable data management system (DMS) for air quality measurement data (Table 1). The COMPASS DMS is being developed to meet global needs for accessible, transparent, and collaborative systems, emphasizing community-informed development and collective learning. By leveraging open-source practices, Clean Air COMPASS seeks to accelerate and scale effective data management worldwide, overcoming inefficiencies and redundancy in proprietary or ad-hoc systems.

Table 1. Summary of the project including website, description, stakeholders, assumptions and key use cases.

Item	Details
Project Name	Community of Practice for Air Quality Systems Data Management System (COMPASS DMS)
Project Website	https://cleanaircompass.org
Website to access system	Coming soon
Project Description	End-to-end platform to ingest, standardize, calibrate, quality-control, store, and publish environmental observations with dashboards and admin tools. The COMPASS DMS will be open-source and available to download, modify, and self-host.
Stakeholders	City/State/National Governments, non-profits and community-based organizations, non-governmental organizations, universities
Intended Users	Air quality program managers, measurements/monitoring personnel, data scientists, QA analysts, public data consumers, IT/Ops
Assumptions	Self-hosted by end-user organizations; Install-anywhere (single VM, on-prem, or cloud VM); no proprietary/cloud-native dependencies; data volume moderate to high (10^6–10^9 observations/year); users prefer low-code/human-readable
Use Cases	Scheduled ingestion from sensors/APIs; schema validation; QC rules; normalization; time-series storage; REST access; dashboards; admin CRUD; notifications; auditing and lineage

1.1 Development in Phases

COMPASS DMS will be developed in phases to deliver value early and grow over time. The initial phase, represented by this plan, will focus on building the Minimum Viable Product (MVP). The MVP will establish the core architecture and implement essential features required to demonstrate system functionality, including data ingestion, standardization, storage, and a basic user interface for access and visualization. As part of the MVP, we will focus on the basics: getting the data pipeline up and running, making sure the data is standardized and stored properly, and giving users a simple interface to

explore their data and manage the system. The MVP will be an open-source COMPASS DMS and source code that is freely available for end-user organizations to download, modify, and self-host. A centralized system that is hosted and maintained by COMPASS may be offered in the future.

Later phases will expand on this foundation by adding more advanced features, such as richer visualization tools, integrations with external systems, and automated notifications. Because the system is open source, we also plan to open the door for community contributions in future phases. This will allow developers and stakeholders outside the core COMPASS team to add new data connectors, visualization modules, or other tools that meet local needs, helping the system evolve faster and stay relevant to a wider set of users.

This phased approach will allow stakeholders to validate the MVP design, provide feedback, and prioritize enhancements before subsequent phases expand functionality, integrate additional modules, and improve scalability, performance, and user experience.

1.2 Stakeholder Workshops

This design plan was developed by analyzing priorities gathered during stakeholder workshops led by core COMPASS organizations: Clean Air Asia, Allin Wayra, YGPE, SciCAN, and ECI. The highest-priority requirements shaped the overall system architecture and selection of the technology stack. Key priorities identified include:

- Data/system sovereignty for end user organizations (no requirements to share data though some workshops identified public data distribution as critical)
- Long term sustainability including a cost-effective architecture and easy installation and maintenance for capacity constrained organizations
- Real-time and historical data ingestion from disparate sources and types (files, API)
- Data harmonization with standardized protocols
- Data QA/QC, anomaly detection, and statistical analysis
- Data storage tiers: raw, validated, aggregated, time-series optimized
- Visualization: dashboards, maps, charts, narratives, multilingual interfaces, data downloads
- Decision/action guidance, forecasting, health messaging, and public alerts
- APIs for integration and bulk download
- Data attribution and sharing agreements
- User-friendly admin interfaces, role-based access control
- Hosting flexibility (on-premises, hybrid, modular)
- System/data backups and redundancy

The architecture outlined in this document for the MVP directly addresses these requirements by ensuring reliable ingestion, robust QC, flexible storage, accessible APIs, multilingual and actionable visualizations, and long-term sustainability.

1.2 Stakeholder Priority Mapping

Table 2 maps each stakeholder group to their top-priority requirements identified during the workshops. This traceability ensures the design plan remains aligned with the needs of all parties. A summary of all requirements identified during the workshops is provided in Appendix A.

Table 2. Summary of stakeholder priority requirements.

Stakeholder Group	Top Priorities
Clean Air Asia	Decision/action guidance based on air quality; Multi-source, real-time data ingestion; Statistical analysis (diurnal/weekly profiles, source ID); Visualization: charts, maps, narratives; Data download & forecasting; Impact assessment of policies/actions
Allin Wayra	Multi-source, real-time data ingestion; Dashboards/maps (real-time + historical); QA/QC & anomaly detection; Data storage (raw, validated, aggregated, time-series optimized); API for bulk download & integration; Machine learning predictions & event detection; Backups/preservation, modular hosting
YGPE	Tiered data storage & access; Dashboards/maps (real-time + historical); Visualization (charts, maps, narratives); Multilingual/localized interface; Health messaging (e.g., AQI-based); Role-based access control, data sharing agreements; Long-term sustainability, local ownership
SciCAN	Data sovereignty & attribution; Multi-source, real-time data ingestion; Data integrity & timeliness; QA/QC & anomaly detection; Dashboards/maps & visualization; Real-time public alerts & health messaging; PII protection in data/maps
ECI	Multi-source, real-time data ingestion; Data storage (raw, validated, aggregated, time-series optimized); Cost-effective solution; Simpler/easier formats & user-friendly interface; Visualization & dashboards; Data sharing agreements & sovereignty; Automated data screening, website/mobile apps

1.2 What is in this Document

This document provides a practical system design outline for an air quality DMS built entirely from open-source, install-anywhere components.

2. Requirements

2.1 Functional Requirements

Table 3 summarizes the non-functional requirements of the system.

Table 3. Summary of priority functional requirements.

General	MVP Requirement	Future Phase Expansion
Raw data ingest from file drops and third-party API	Support for CSV (1 format) and up to three third-party API	Additional formats (CSV, Excel, JSON, etc.) and third-party API
Standardize data formats, naming conventions, units, and time zones	Standardize to Air Quality Data Exchange (AQDx) format for csv; prioritizing criteria pollutants (not VOC, meteorology)	Support for AQDx JSON format; support for additional standardization frameworks
Apply schema validation and QC tests with human-readable specs	Framework for QC tests; support for basic QC tests (e.g., min/max threshold); specific QC/calibration routines for up to three sensors	Additional types of tests and custom routines
Quarantine failures with logging	Quarantine failures	Log failures and trigger alerts
Compute derived metrics (AQI)	Framework for AQI calculations; Up to three custom calculations (U.S. EPA AQI + two additional)	Additional AQI or similar calculations
Persist raw and curated data into a data lake	CSV files	Parquet, compression, retention policies
Persist curated data as structured observations in a database (time-series optimized)	Store processed data; calculate aggregates	Additional aggregates (e.g., rolling averages), retention policies
Expose REST API over database	Core endpoints (query data, user, and system information)	Optimized queries for advanced analyses and visualizations
Provide visualization interface	Basic dashboard with map, time series, QC displays, data downloads	Customizable plots and advanced displays; data QC from the dashboard
Multilingual/localized interface	Framework for supporting different languages; up to 2 specific languages in MVP (English + one additional)*	Custom interfaces for additional languages; custom logos and branding
Provide system admin functions	Web based UI for configuring stations/sensors/parameters/users	Additional CRUD for e.g., RBAC**, QC tests, system configurations
Track data lineage, logs	Log basic qc results	Log manual data qc, user access, notifications sent; Access to logs via API and interface

General	MVP Requirement	Future Phase Expansion
Notifications	Email alerts for administrators	Alerts by user-groups for qc results, concentration thresholds, Public sign-up for notifications
Minimal-configuration setup for multiple operating systems	One OS optimized for MVP	Support for additional OS

*Language support for the MVP will rely on translation tools and may be limited to certain typographic conventions.

**RBAC = Role Based Access Control (specific permissions are assigned to specific users or user groups such as read-only, data validator, system administrator, etc.)

2.2 Non-Functional Requirements

Table 4 summarizes the non-functional requirements of the system to be developed as part of future phase expansion. These requirements may not be developed/evaluated unless a centralized COMPASS DMS is offered.

Table 4. Summary of non-functional system requirements for future phases (not MVP).

Category	Target / SLO
Performance	API p50 ≤ 200 ms; p95 ≤ 800 ms
Throughput	≥ 50k obs/min sustained
Availability	99.5% (single-node) / 99.9% (HA)
Security	OIDC SSO, TLS in transit, RLS at DB
Scalability	Scale vertically; shard by station/time
Maintainability	Single-file Compose; pin images
Observability	Metrics, logs, traces
Portability	Runs on Linux/Windows

3. System Architecture

In addition to the functional and non-functional requirements, several additional criteria were used to inform the design of the system architecture and selection of technology for each component. The criteria include: 1) reliance on open-source technology/software only; 2) flexibility for future versions (e.g., on-premises vs. cloud hosting, compatibility with diverse operating systems); and 3) simplicity, prioritizing a cohesive technology stack built around a small number of programming languages and frameworks.

3.1 High-Level Diagram

The DMS architecture (Figure 1) is designed to reliably manage and deliver environmental data in a structured and reliable way. Incoming data sources may include files such as CSV or Excel files, JSON feeds, and external APIs. These inputs flow into the data pipeline, which is responsible for acquisition, standardization and harmonization of formats and units, quality assurance and quality control (QA/QC), and data aggregation. Once processed, the data is stored in two layers: a file system for raw and reference data, and a relational database for structured and query-ready data. From storage, the system exposes data through an API, which provides controlled and consistent access for both internal components and external users. On top of this, a user-facing front end offers visualization and analysis tools, including interactive dashboards with maps, time series plots, statistical summaries, and options for downloading datasets.

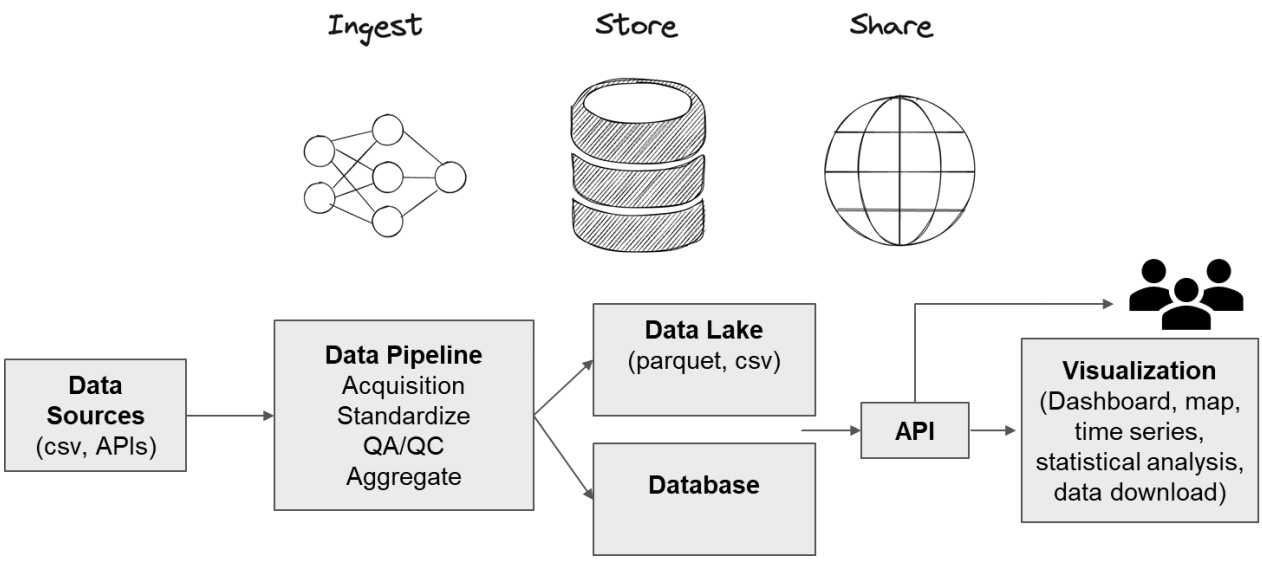

Figure 1. High-level diagram of system architecture.

3.2 Technology Stack

The system uses no cloud-native dependencies, ensuring maximum deployment flexibility (Table 5). It will rely on a Python-based backend for data processing and API services (ETL, QA/QC), a PostgreSQL database with PostGIS extensions for structured and spatial data, and a data lake for raw and processed datasets in CSV format. A FastAPI service will expose the data and analytics through a documented RESTful API. The user-facing application will be built with TypeScript using React and Next.js, providing interactive dashboards, maps, and tools for data exploration and download. Containerization via Docker Compose ensures portability.

This architecture leverages a modern, open-source technology stack. Together, these components provide a robust foundation for scalable, transparent, and user-friendly environmental data management and visualization. Alternative technology is further discussed in Appendix B.

Table 5. Summary of MVP technology by system component.

Component	Technology
Ingestion	Python scripts/services (scheduled jobs)
Validation/Standardization	Python + Pandas + Pydantic (schema enforcement, unit normalization)
Quality Control	Python + custom QC rules (statistical/anomaly detection, QA/QC flags)
Data Lake	File system or object store with CSV
Relational DB	PostgreSQL + PostGIS
API	FastAPI (full REST layer, OpenAPI/Swagger docs)
Frontend (Dashboards, Maps, Admin)	React + Next.js (TypeScript)
Alerts	SMTP via Python
Packaging/Deploy	Docker Compose

3.3 Mapping Stakeholder Priorities to Default Stack Components

Table 6 links the stakeholder-identified priorities to the default open-source stack components chosen for the system. It demonstrates how the architecture addresses stakeholder needs with concrete technologies.

Table 6. Summary of MVP technology by system component.

Stakeholder Priority	Mapped Component(s)	Rationale
Multi-source, real-time data ingestion	Python ingestion services (scheduled/async)	Flexible ingestion of files, APIs, or streaming inputs with logging and lineage tracking
QA/QC & anomaly detection	Pandas + Pydantic + custom Python QC routines	Schema validation, unit checks, and anomaly detection in a single Python codebase
Data storage (raw, validated, aggregated, time-series optimized)	CSV (raw + curated) + PostgreSQL/PostGIS (structured, indexed)	Clean separation of raw vs. structured data, with optimized queries for analytics and spatial data
Visualization: dashboards, maps, charts, narratives	React + Next.js (TypeScript)	Interactive, modern web application with rich mapping and visualization capabilities
Decision/action guidance, health messaging, public alerts	React dashboards + FastAPI custom endpoints	Threshold-based visuals, notifications, and integration-ready endpoints

Stakeholder Priority	Mapped Component(s)	Rationale
Data download & API integration	FastAPI	Full-featured REST API with bulk download endpoints and Swagger docs
Role-based access control / tiered access	Postgres RLS	Row-level security in Postgres with optional identity management
Data sovereignty, attribution, ownership	Metadata tracking and chain of custody in the application	Transparent lineage, reproducible schemas, traceability of data origin
Hosting flexibility (on-premises, hybrid, modular)	Docker Compose	Portable, modular deployments across environments
Backups, preservation, sustainability	pgBackRest + file versioning	Automated backups and versioning for redundancy and long-term sustainability

4. Module Design

The system is organized into modular components, each responsible for a specific set of functions within the data pipeline, storage, API, and frontend layers. Table 7 summarizes each module's purpose, inputs, outputs, and key dependencies, providing a clear view of how data flows from acquisition to user-facing applications. This modular design promotes maintainability, scalability, and flexibility, enabling components to evolve independently as the system grows.

Table 7. Details (inputs, outputs, dependencies) for each MVP module.

Module	Purpose	Inputs	Outputs	Dependencies
Ingestion Flow (Python)	Pulls data from sources on schedules; records provenance; writes raw CSV to data lake; triggers validation/QC	Source URLs/paths, polling intervals, credentials	Raw files (CSV) in data lake; provenance metadata	External APIs, file shares
Validator (Python/Pydantic)	Enforces schemas, field types, required constraints; produces validation report; flags bad records	Dataset (CSV/JSON), schema definition	Validation report (JSON), pass/fail outcome	Schema repo (Git)
QC Runner (Python)	Applies statistical and threshold-based QC checks; annotates qc_flag and qc_detail fields	Validated dataset	QC'd dataset with flags and metadata	Validator output

Module	Purpose	Inputs	Outputs	Dependencies
Transformer (Python/Pandas)	Performs unit normalization, timezone standardization, derived fields; writes CSV; inserts into Postgres	QC'd dataset	Curated files in data lake; inserts to Postgres/PostGIS	Postgres, data lake
API Layer (FastAPI)	Provides REST endpoints (data retrieval, aggregation, metadata, downloads); includes OpenAPI docs	HTTP requests	JSON/CSV responses	Postgres, data lake
Frontend (React + Next.js)	Delivers dashboards, maps, time series, admin console, and download interface	API endpoints	Interactive web app (visualizations, data access)	FastAPI

5. Risks & Mitigations

The MVP system faces potential risks related to data quality, performance, reliability, and security (Table 8). The mitigation strategies listed in the table, ranging from schema validation and partitioning to backups, access controls, and monitoring, ensure that these risks are managed proactively, helping maintain data integrity, availability, and secure operation. In future versions of the system, risk mitigation will be enhanced through both technical and operational improvements. For example, schema validation and QC rules will evolve to cover additional data sources and edge cases, while automated monitoring and alerting will be extended to detect anomalies more proactively. High-availability and failover capabilities for PostgreSQL/PostGIS and the data lake will be implemented to reduce downtime and read replicas or sharding strategies may be introduced to handle increased analytical loads. Security controls, including role-based access, encryption, and audit logging, will be continuously reviewed and updated to align with best practices. These ongoing improvements will ensure that the system remains robust, scalable, and secure as data volumes and user requirements grow.

Table 8. Details (inputs, outputs, dependencies) for each module.

Risk	Impact	Mitigation
Schema drift from partners	Load failures or silent data quality issues	Strict schema validation; quarantine; partner contracts; versioned schemas
QC rule complexity	False positives/negatives	Iterate with data docs; pilot thresholds; per-parameter overrides
Single-node failure	Downtime/data loss	Automated backups; consider HA for Postgres (future phase)
Unbounded growth	Storage/performance degradation	Partitioning + retention; tiering; aggregate rollups

Risk	Impact	Mitigation
Security misconfig	Data exposure	OIDC + RLS; regular audits; secrets management; TLS everywhere (future phase)

6. Next Steps

While this document establishes the high-level architecture and technology stack for the system, several design elements will be developed in greater detail in future iterations of the design plan and system documentation. These additions will ensure that the implementation is guided by precise specifications and that the system is prepared for secure, scalable, and sustainable operation.

Additional design details will include the following:

- Entity-Relationship (ER) Model and Database Schema – A data model for PostgreSQL/PostGIS, including entity relationships, indexing strategies, and geospatial structures.
- API Design – API specification with documented endpoints, input/output formats, authentication methods, and error handling conventions.
- Security Design – Access control strategy, role-based permissions, and data encryption considerations.
- Deployment Architecture Topologies – Alternative deployment diagrams (e.g., on-premises, hybrid, or cloud) with guidance on scaling, redundancy, monitoring, and backups.
- Test plan - Testing will be mult-tiered (e.g., unit tests, integration tests) and include engagement with COMPASS organizations as beta testers.

These details will be fully documented in future iterations of this plan, providing the necessary technical depth to guide development, integration, and operations across different hosting environments.

Appendix A – Workshop Feedback Requirements Summary

Lead Organization	Feature/Function	Normalized Requirement
Clean Air Asia	Data explainers to ensure ease of understanding the numbers	Clear decision/action guidance based on air quality
Clean Air Asia	Ease of consolidating air quality from various sources/instruments	Data ingestion (multi-source, real-time, protocols)
Clean Air Asia	Identification of sources of air pollution	Statistical analysis (diurnal/weekly profiles, source identification)
Clean Air Asia	Clear guidance on decisions/actions based on air quality	Clear decision/action guidance based on air quality
Clean Air Asia	Time series of the data	Data visualization (charts, maps, story maps, narratives)
Clean Air Asia	Short-term actions based on air quality conditions	Clear decision/action guidance based on air quality
Clean Air Asia	Ability to integrate meteorological data to air quality	Statistical analysis (diurnal/weekly profiles, source identification)
Clean Air Asia	Data download	Data download
Clean Air Asia	Long-term actions based on air quality conditions	Impact of actions/policies on air quality
Clean Air Asia	Impact of actions/policies on air quality	Impact of actions/policies on air quality
Clean Air Asia	Air quality forecast	Forecasting (air quality & emissions)
Clean Air Asia	Ability to integrate other emission data to air quality	Statistical analysis (diurnal/weekly profiles, source identification)
Allin Wayra	Ingestion: unified acquisition and transmission protocols; clear typologies by equipment type (reference, LCS, satellite); security and backup of raw data; ingestion from multiple sources in real time with basic air quality index visualization; public availability of data.	Data ingestion (multi-source, real-time, protocols)
Allin Wayra	Storage: separation by typology, redundant backups (local + cloud), and long-term preservation policies with accessible costs.	Data backups / preservation / redundancy

Lead Organization	Feature/Function	Normalized Requirement
Allin Wayra	Sharing/visualization: free download (CSV/API) and intuitive public platform with color-coded map and basic indicators.	Data download
Allin Wayra	Sharing/visualization: free download (CSV/API) and intuitive public platform with color-coded map and basic indicators.	Dashboards/maps (real-time + historical)
Allin Wayra	Hosting/implementation: hybrid local-cloud architecture, open and modular.	Hosting (on-premises, hybrid, modular)
Allin Wayra	Confidence ranking per data point and multi-level QA/QC pre-processing; automatic anomaly filters.	QA/QC & anomaly detection
Allin Wayra	Storage of all data levels (raw, validated, aggregated) in open-source, time-series-optimized databases.	Data storage (raw, validated, aggregated, time-series optimized)
Allin Wayra	Quick statistical analyses (means, diurnal/weekly profiles) and layered visualization integrating meteorological, health, and satellite variables; optimized loading/download times.	Statistical analysis (diurnal/weekly profiles, source identification)
Allin Wayra	Advanced API to automate bulk downloads and forecasting services.	API for bulk download & integration
Allin Wayra	Machine learning applications for event detection and predictions; customizable dashboards for different audiences.	Machine learning predictions & event detection
Allin Wayra	Automatic multilingual translation of the interface and documentation.	Multilingual/localized interface
Allin Wayra	Open publication of standardized scripts/procedures and historical performance metrics.	Open publication of procedures/scripts/metrics
YGPE	Storing historical and real-time data	Data storage (raw, validated, aggregated, time-series optimized)
YGPE	Publicly accessible dashboards (with maps and charts)	Dashboards/maps (real-time + historical)
YGPE	Publicly accessible dashboards (with maps and charts)	Data visualization (charts, maps, story maps, narratives)
YGPE	Language localization (Russian)	Multilingual/localized interface

Lead Organization	Feature/Function	Normalized Requirement
YGPE	Tiered data access (from dashboard) by user account and data type (RT vs historical)	Tiered access by user/data type
YGPE	Data downloads	Data download
YGPE	Standard data metric (such as AQI)	Health messaging (e.g., AQI-based)
YGPE	Easy installation and maintenance for hosting and deployment	Hosting (on-premises, hybrid, modular)
YGPE	On-premises hosting (government end user organizations)	Hosting (on-premises, hybrid, modular)
YGPE	Role based access controls	Role-based access control
YGPE	Data sharing	Data sharing agreements / procedures
YGPE	Long term sustainability	Long-term sustainability
YGPE	Local independence and data ownership by the end user organization	Local independence / end-user data ownership
SciCAN	Data sovereignty	Data sovereignty / ownership
SciCAN	Data attribution	Data attribution
SciCAN	Data accessibility (timeliness)	Data accessibility (timeliness)
SciCAN	Data accessibility (timeliness)	Data ingestion (multi-source, real-time, protocols)
SciCAN	Data integrity (accuracy)	Data integrity (accuracy)
SciCAN	Data integrity (accuracy)	QA/QC & anomaly detection
SciCAN	PII can be protected (e.g., on maps)	PII protection in data/maps
SciCAN	Data visualization (e.g., maps)	Dashboards/maps (real-time + historical)
SciCAN	Application to share story with data (e.g., story map, community experience told with photos, videos, narratives)	Data visualization (charts, maps, story maps, narratives)
SciCAN	Alerts for high pollution that public can subscribe to (texts, emails, calls)	Real-time public alerts/notifications
SciCAN	Real-time information to the public	Real-time public alerts/notifications
SciCAN	Compare data by week/season	Statistical analysis (diurnal/weekly profiles, source identification)
SciCAN	AQI based health messaging	Health messaging (e.g., AQI-based)

Lead Organization	Feature/Function	Normalized Requirement
ECI	Data intake from air quality monitoring stations/instruments/devices	Data ingestion (multi-source, real-time, protocols)
ECI	Data intake for real-time and historical data	Data ingestion (multi-source, real-time, protocols)
ECI	Data sharing agreements for third-party owned data sets	Data sharing agreements / procedures
ECI	Systems and procedures for data storage, processing and quality control	Data storage (raw, validated, aggregated, time-series optimized)
ECI	Cost effective solution	Cost-effective solution
ECI	Systems and procedures for distribution and sharing of data with users	Data sharing agreements / procedures
ECI	simpler, easy to understand and use formats	Simpler/easier formats & use
ECI	User friendly data display and reporting interface	User-friendly interface (reporting)
ECI	Interactive maps	Data visualization (charts, maps, story maps, narratives)
ECI	Website and mobile application(s)	Website and/or mobile applications
ECI	Display real-time and historical data, trends	Dashboards/maps (real-time + historical)
ECI	Data and DMS ownership	Data sovereignty / ownership
ECI	Automated data screening	Automated data screening

Appendix B – Alternative Technology Stack Options by Component

Alternative open-source technologies were evaluated for each layer to ensure flexibility and provide potential options for future iterations or different deployment scenarios. The primary options were chosen to minimize operational overhead, maximize maintainability, and leverage a cohesive stack. This approach allows the development team to use a consistent programming language and tooling across the pipeline, database, API, and frontend, reducing complexity while retaining the ability to swap in alternative solutions if specific needs or scaling requirements arise.

Layer / Category	Primary Option	Alternatives
Ingestion	Python scripts/services	Kafka, Apache NiFi, RabbitMQ
Validation / Standardization	Pydantic + Python	Frictionless, dbt
Quality Control	Custom Python QC functions	Great Expectations, Pandera, Deequ
Data Lake	CSV	Parquet, MinIO, Ceph, S3-compatible stores
Relational DB	PostgreSQL + PostGIS	TimescaleDB, SQLite
API	FastAPI	Flask, Django REST Framework
Frontend	React + Next.js (TypeScript)	Angular, Vue.js, Streamlit, Superset, Metabase
Packaging / Deploy	Docker Compose	Ansible, Kubernetes