SE 4485: Software Engineering Projects

Fall 2025

Architecture Documentation

|  |  |
| --- | --- |
| Group Number | Group 1 |
| Project Title | County Level Air Quality Prediction Application |
| Sponsoring Company | Raytheon (Team A) |
| Sponsor(s) | Ryan Havens <Ryan.Havens@rtx.com>,  Marc Perna <marc.perna@rtx.com>,  Trey Williams <trey.williams@rtx.com>,  Trevor Lang <trevor.a.lang@rtx.com> |
| Students | 1. Jay Chung <cwc130330@utdallas.edu>  2. Amelia Quinn <qcb220000>  3. Kevin Melo <ksm220005>  4. AJ Kimbrough <ank210005>  5. David Santos <des210001>  6. Andrew Enright <ame210008> |

**ABSTRACT**

This document defines the software architecture for the County Level Air Quality Prediction (CLAP) web application. The CLAP system provides an interactive, browser-based dashboard that ingests daily Air Quality Index (AQI) data from the U.S. Environmental Protection Agency (EPA), persists historical datasets, generates lag-based analytical features, and trains a predictive analytic machine-learning (ML) model (e.g. LightGBM). The loaded model produces next-day, county-level AQI forecasts served through a Representational State Transfer (REST) Application Programming Interface (API). The architecture supports data processing, modular model management, and responsive visualization for end users.

**TABLE OF CONTENTS**

1. INTRODUCTION 3
2. ARCHITECTURAL STYLE(S) USED 4
3. ARCHITECTURAL MODEL 6
4. TECHNOLOGY, SOFTWARE, AND HARDWARE USED 8
5. RATIONALE FOR YOUR ARCHITECTURAL STYLE AND MODEL 9
6. TRACEABILITY FROM REQUIREMENTS TO ARCHITECTURE 10
7. EVIDENCE OF CONFIGURATION MANAGEMENT 12
8. ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS 12
9. ADDITIONAL REFERENCES 12

**LIST OF FIGURES**

Figure 2.1 – CLAP – Client-Server 4

Figure 2.2 – CLAP – Layered Architecture 4

Figure 2.3 – CLAP – Pipe-And-Filter Architecture 5

Figure 3.1 – CLAP – Package-Level Decomposition 6

**LIST OF TABLES**

Table 2.1 – CLAP – Key System Features 5

Table 4.1 – CLAP – Technologies and Software Used 8

Table 6.1 – CLAP – FR-Architecture Compact Traceability Matrix 10

Table 6.2 – CLAP – NFR-Architecture Compact Traceability Matrix 11

Table 7.1 – GitHub – Single File Revision 12

Table 7.2 – GitHub – Difference Link 12

**INTRODUCTION**

This document presents the Software Architecture Description (SAD) for the County Level Air Quality Prediction (CLAP) web application. Its primary purpose is to define the structural organization, components, and interaction mechanisms that enable the system to meet its functional and non-functional requirements. The architecture ensures that the design and implementation align with the objectives established by the project sponsors.

The CLAP system is a predictive analytics web application designed to forecast next-day Air Quality Index (AQI) categories at the county level. The system ingests historical AQI data provided by the U.S. Environmental Protection Agency (EPA), processes and stores it in a structured database, and applies ML algorithms (e.g. LightGBM model) to generate next-day AQI predictions. This architecture demonstrates an integrated workflow of data collection, model inference, and visualization for educational purposes.

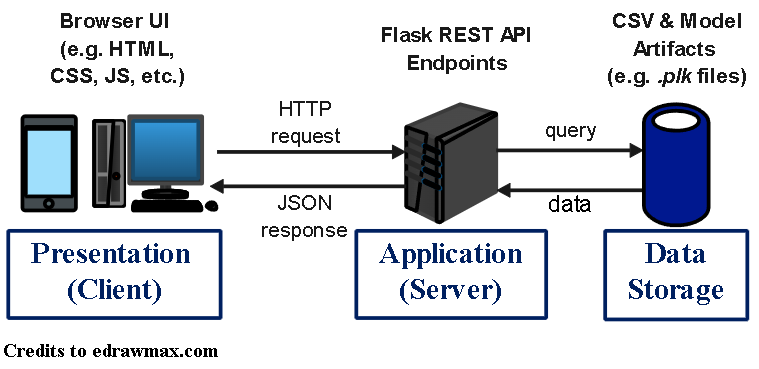
The CLAP architecture implements a modular, layered client-server design that separates responsibilities among data ingestion, model management, and presentation layers. A RESTful backend provides prediction services consumed by a responsive browser-based dashboard that visualizes recent AQI trends and next-day forecasts. The architecture emphasizes scalability, maintainability, and testability while ensuring compatibility with lightweight local deployment environments.

The remainder of this document is organized as follows: description of architectural styles used, detailed architectural model and diagrams, technologies and tools applied, rationale for the selected styles and models, traceability from requirements to architecture, configuration management evidence, relevant standards and constraints, and references.

**ARCHITECTURAL STYLE(S) USED**

The CLAP web application adopts a **client-server** architecturewith a **layered design** that separates the system into **Presentation (client)**, **Application (server)**, and **Data** layers. This separation of concerns enables end-to-end functionality while improving **modularity, maintainability, and testability**, and allows each layer to evolve independently without impacting the others.

SYSTEM LEVEL ARCHITECTURAL DESIGN:



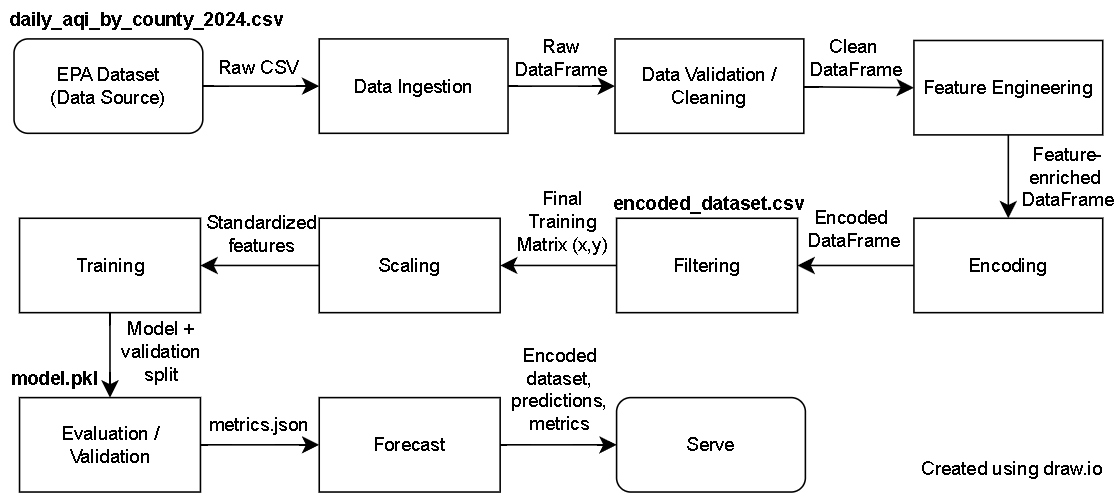
**Figure 2.1 – Client-Server Architecture**

SOFTWARE LEVEL ARCHITECTURAL DESIGN:

A diagram of a software process

AI-generated content may be incorrect.

**Figure 2.2 – Layered Architecture**



**Figure 2.3 – Pipe-And-Filter Architecture**

CORE SYSYEM DESIGN FEATURES:

This architecture supports the core design features of the CLAP system as follows:

**Table 2.1 – Support for Core System Features**

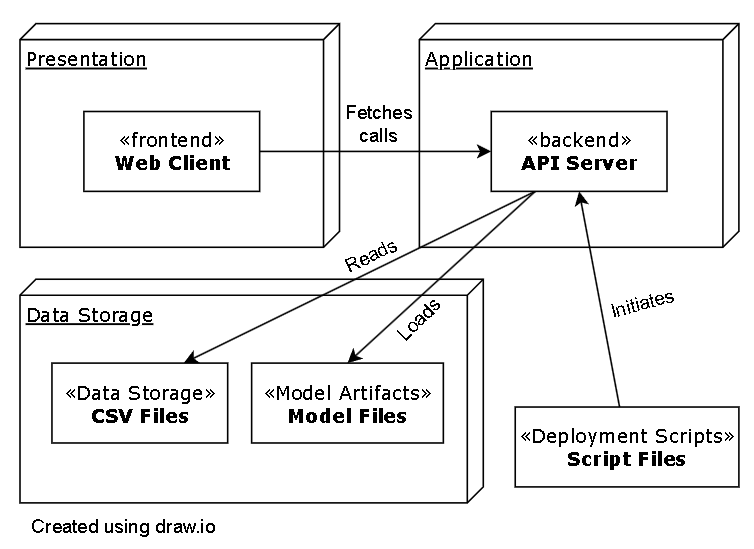
|  |  |
| --- | --- |
| **SYSTEM FEATURES:** | **ARCHITECTURAL SUPPORT DESCRIPTION:** |
| **Manual Data Ingestion:** | The **Application Layer** handles on-demand ingestion, connecting to EPA endpoints and validating schema before persistence. |
| **Data Persistence:** | The **Data Layer** manages cleaned AQI data, lag features, and model outputs using CSV and model artifacts. |
| **Model Training & Prediction:** | The **Application Layer** orchestrates model training & inference, ensuring modular separation between data, logic, and presentation. |
| **Interactive Dashboard:** | The **Presentation Layer** visualizes AQI trends and predictions, consuming RESTful JSON data served by the backend. |
| **Scalability & Maintainability:** | Each subsystem can be modified or upgraded independently (e.g. model re-training, dashboard upgrade, etc.) without disrupting other layers. |

**ARCHITECTURAL MODEL**

The architectural model is represented using **package-style decomposition,** where each component represents a functional unit responsible for a distinct concern in the overall architecture.

PACKAGE-LEVEL DECOMPOSITION:

This package-level model shows how major subsystems interact without exposing class-level details. Each <<subsystem>> fulfills distinct responsibilities within the overall Client-Server architecture.



**Figure 3.1 – Package-Level Decomposition**

SUBSYSTEM OVERVIEW:

1. <<FRONTEND>> (WEB CLIENT):

* Files:
  + HTML, CSS, and JavaScript Files (e.g. *index.html*, *styles.css*, and *app.js*).
* Responsibilities:
  + Provides the user interface for county selection and forecast generation.
  + Calls RESTful API endpoints to retrieve the historical and predictive AQI data.
  + Renders AQI trends and forecast results using *Chart.js* for visualization.
  + Displays user feedback (e.g. loading states, model metrics, and errors).
* API Integration:
  + API\_BASE\_URL = <http://localhost:5001/api>
  + Implements fetch calls to “/counties”, “/aqi/historical”, “/aqi/predict”, and “/model/metrics”.

1. <<BACKEND>> (API Server – Flask):

* Location:
  + Python files in the “/backend” directory (e.g. launched via *app.py* in *run* script).
* Responsibilities:
  + Exposes RESTful API endpoints to serve the list of available counties, provide historical AQI data for selected counties, and run model inference and return predicted AQI categories.
  + Loads persisted LightGBM models (e.g. *.pkl* files).
  + Returns results as JavaScript Object Notation (JSON) responses consumed by the FRONTEND.
* Deployment:
  + Executed using the provided script (e.g. *run.bat* for Windows OS or *run.sh* for Mac OS), which initializes the Python virtual environment and starts the Flask server.

1. <<DATA STORAGE>>:

* Storage Mechanism:
  + Local CSVs (e.g. encoded\_dataset.csv, etc.)
* Responsibilities:
  + Stores cleaned historical AQI data and generated lag features for model input.
  + Maintain data integrity, versioning, and accessibility for reproducibility.

1. <<MODEL ARTIFACTS>>:

* Files:
  + Trained model files (e.g. *balanced\_lightgbm\_model.pkl,* etc.).
* Responsibilities:
  + Stores trained ML model artifacts used for inference.
  + Supports reloading of models by the API for real-time predictions.

1. <<DEPLOYMENT SCRIPTS>>:

* Files:
  + Batch Script Files (e.g. *run.bat* for Windows OS or *run.sh* for Mac OS).
* Responsibilities:
  + Automates local deployment and environment setup.
  + Creates and activates a Python visual environment.
  + Installs project dependencies, ensures *.env* configuration, and launches the Flask server.
  + Enables consistent and reproducible setup for development and testing.

**TECHNOLOGY, SOFTWARE, AND HARDWARE USED**

This section describes the technologies, software, and hardware used to implement the CLAP system, lists supporting software and hardware requirements, and explains how communication occurs between the application server and the database server.

TECHNOLOGIES AND SOFTWARE USED:

**Table 4.1 – Technologies and Software Used**

|  |  |  |
| --- | --- | --- |
| **LAYER / COMPONENT:** | **TECHNOLOGY / TOOLS:** | **PURPOSE:** |
| **Frontend:** | HTML5, CSS, Vanilla JavaScript, Chart.js | Implements interactive dashboard and chart visualization. |
| **Backend:** | Python 3.13+, Flask Framework | Hosts RESTful APIs, manages model inference & data orchestration. |
| **Machine Learning:** | LightGBM, Pandas, Scikit-learn API | Supports AQI model training & prediction. |
| **Persistence:** | CSV files, model artifacts (*.pkl* files) | Stores historical AQI data, lag features, and model outputs. |
| **Runtime Environment** | Virtualenv, *run.bat* / *run.sh* | Automates setup and ensures environment isolation. |
| **Version Control** | Github | Enables configuration management and collaboration. |

SOFTWARE AND HARDWARE REQUIREMENTS:

* Operating System: Windows, mac OS, or Linux
* Python Environment: Python 3.13+ with pip dependency manager
* Minimum Hardware:
  + CPU: Dual-core processor or higher
  + RAM: 8 GB or higher
  + Storage: ≥ 100 GB available disk space
  + Network: Internet access for data retrieval

COMMUNICATION BETWEEN APPLICATION SERVER AND DATABASE SERVER:

The **Flask backend** acts as the **application server**, communicating with the **data storage layer** through **local file I/O** (e.g. Pandas for CSV-based persistence, loading of *.pkl* files). During execution, the backend retrieves clean AQI datasets and lag features. The trained LightGBM model is loaded to fetch inference on the selected county data. The backend sends results as JSON responses to the frontend via REST APIs. This communication flow ensures efficient data handling, clear separation of responsibilities, and modular scalability across layers.

**RATIONALE FOR YOUR ARCHITECTURAL STYLE AND MODEL**

The chosen architecture combines **layered architecture** with the **Client-Server** **model**,providing clear modular boundaries between the user interface, application logic, and data storage. This separation of concerns simplifies debugging, testing, and maintenance while enabling independent updates to the dashboard, API logic, or data components without impacting other layers. The model achieves a balance between **simplicity, modularity, and maintainability,** supporting both the project’s technical objectives within academic constraints.

ALIGNMENT WITH REQUIREMENTS:

The architecture directly supports functional requirements defined in the SRS, including:

* FR-1 – Manual data ingestion
* FR-2 – Storage and preprocessing
* FR-3 – Model-based forecasting
* FR-4 – Interactive dashboard visualization.

SEPARATION OF CONCERNS:

By separating the frontend, backend, data management, and ML subsystems, the system maintains a clean modular structure. This facilitates localized maintenance, reducing coupling, and allows parallel development across team members or future contributors.

SCALABILITY AND PORTABILITY:

The use of lightweight frameworks such as Flask and portable storage options (e.g. CSV) enables deployment on student hardware while remaining extensible to cloud environments. This ensures future scalability for handling larder datasets or multi-county inference without major re-structuring of this architecture.

MAINTAINABILITY AND TRACEABILITY:

Subsystem boundaries are explicitly defined, supporting traceability from requirements 🡪 architecture 🡪 test artifacts. This structure allows individual subsystems to evolve independently, minimizing regression risk.

FEASIBILITY:

The client-server model provides an end-to-end demonstration of core data science and software-engineering principles – from ingestion through prediction to visualization – within the time and resource constraints of an academic project.

**TRACEABILITY FROM REQUIREMENTS TO ARCHITECTURE**

This section maps each requirement to architectural elements of the CLAP system and points to specific implementation evidence. The goal is to make it clear where and how each requirement is realized in the design.

FUNCTIONAL REQUIREMENTS TO ARCHITECTURE MAPPING:

**Table 6.1 – FR-Architecture Compact Traceability Matrix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FR ID:** | **FR DESCRIPTION:** | **ARCHITECTURAL ELEMENT(S):** | **INTERFACES / ARTIFACTS:** | **HOW THE FR IS CAPTURED:** |
| FR-1.1 | Utilize daily EPA AQI data, ingested manually. | Deployment Scripts, Backend (API Server), Data Store. | Ingestion endpoint, I/O, Run scripts. | Maintainer triggers manual ingest. API validates and writes cleaned data to CSV. |
| FR-2.1 | Store historical AQI data in CSV. | Backend, Data Store. | CSV, I/O. | Data Store persists historical datasets. API performs I/O via pandas. |
| FR-2.2 | Generate lag features for prediction. | Backend, Data Store. | Feature tables, CSV. | API computes lag features and persists outputs alongside base data for reproducible training. |
| FR-3.1 | Train and run a predictive model using historical AQI + county location. | Backend, Data Store, Model Artifacts. | Training pipeline, LightGBM, *.pkl* | API loads and saves model artifacts, uses Data Store inputs to train and inter models. |
| FR-3.2 | Output next-day AQI category for selected county. | Backend, Model Artifacts. | /aqi/predict JSON | API performs inference with loaded model and returns predicted category. |
| FR-3.3 | Provide probability score for each category. | Backend, Model Artifacts. | /aqi/predict JSON | API includes class probabilities in JSON response. |
| FR-4.1 | Provide a GUI for visualizing AQI data & predictions. | Frontend | HTML/CSS/JS, Chart.js | Frontend renders history and prediction views, calling API endpoints. |
| FR-4.2 | Display chart of last 30 days + next-day category with probabilities. | Backend, Frontend | /aqi/historical, /aqi/predict JSON, Chart.js | Frontend queries API, renders 30-day chart and probability display. |
| FR-4.3 | Provide Refresh control buttons that updates from local dataset and regenerates forecast. | Backend, Frontend, Data Store, Model Artifacts. | Refresh control button 🡪 API calls 🡪 Data access | Clicking Refresh control buttons trigger API read of local data, re-runs inference, updates UI. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NFR ID:** | **NFR DESCRIPTION:** | **ARCHITECTURAL ELEMENT(S):** | **MECHANISMS:** | **HOW THE NFR IS CAPTURED:** |
| NFR-1.1 | Ingestion + validation + features ≤ 60 s for one county. | Deployment Scripts, Backend, Data Store. | Efficient I/O, vectorized feature engineering. | Minimal I/O hops. Scripting ensures optimized local env. |
| NFR-1.2 | Dashboard renders ≤ 5s after Refresh. | Backend, Frontend. | Small static bundle, compact JSON caching. | Lightweight assets and efficient API response achieves latency target. |
| NFR-2.1 | Use EPA category labels (e.g. Good…Hazardous). | Frontend | Category mapping, UI labels. | UI uses standard EPA labels in legends. |
| NFR-2.2 | Use distinct EPA/AirNow colors per category. | Frontend | Color palette. | *Chart.js* theme applies target colors (e.g. green, yellow, orange, red, purple, maroon). |
| NFR-3.1 | Conform to WCAG 2.1 AA (SC 1.4.1, 1.4.3, 1.4.11). | Frontend | Contrast checks, non-color cues, Accessible Rich Internet Applications (ARIA). | UI provides sufficient contrast, non-color indicators, and accessible semantics. |
| NFR-4.1 | Daily ingest succeeds on ≥ 90% of runs. | Deployment Scripts, Backend, Data Store. | Retries & backoff. Atomic writes & health checks. | Retry policies and atomic persistence improve success rate. |
| NFR-5.1 | Dashboard availability ≥ 99% over any 30-day period. | Deployment Scripts, Backend, Frontend. | Stateless API. Simple deploy. | Static frontend and simple Flask service minimize downtime. Scripts standardize recovery. |
| NFR-6.1 | Implement structured logging for key operations. | Backend, Deployment Scripts. | Structure logs (JSON / text) | Logs emitted for ingest, validation, feature engineering, and prediction. |
| NFR-6.2 | Logs include timestamps and error details. | Backend. | Logger configuration. | Logger attaches ISO timestamps and error metadata to entries. |

NON-FUNCTIONAL REQUIREMENTS TO ARCHITECTURE MAPPING:

**Table 6.2 – NFR-Architecture Compact Traceability Matrix**

**EVIDENCE THE DOCUMENT HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT**

The team has selected GitHub as the configuration tool for this project. The tables below provide evidence of configuration management by recording version history, authorship, and reviews of document changes. The *ID* column identifies each entry. The *date of change* column indicates when a modification was made to an existing file, and the v*ersion (before & after)* columns include the associated Git commit hash for distinction. The *author* column refers to the author of the new version. The *difference link* column provides a URL to the GitHub comparison view between two consecutive commits. The format of the difference link is as follows:

“https://github.com/cchung7/rtx\_team1/compare/<ver-before-hash>..<ver-after-hash>”.

**Table 7.1 – Single file revision**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID:** | **Date of Change:** | **Version Before:** | **Version After:** | **Author:** | **Review -Change Summary:** | **Reviewers:** |
| 1 | 10/12/25 | v0.1 (26b1e4b) | v0.2 (5c314e8) | Jay Chung (cwc130330) | Peer Review – Consistency pass for all Sections | All Team Members |
| 2 | 10/14/25 | v0.2 (5c314e8) | v0.3 () | Jay Chung (cwc130330) | Traceability & Consistency pass | All Team Members |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |

**Table 7.2 – Difference link**

|  |  |
| --- | --- |
| **ID:** | **Difference Link:** |
| 1 | https://github.com/cchung7/rtx\_team1/compare/26b1e4b..5c314e8 |
| 2 | https://github.com/cchung7/rtx\_team1/compare/5c314e8.. |
| 3 |  |
| 4 |  |

**ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS**

Engineering Standards:

* IEEE Std 1058-1998: Software Project Management Plans [[pdf](https://course.techconf.org/se4485/IEEE/IEEE-Std-1058-1998-Software-Project-Management-Plans.pdf)]
* PMBOK® Guide: Project Management Body of Knowledge [[pdf](https://course.techconf.org/se4485/IEEE/PMBOKR.pdf)]
* IEEE Std 12207: Software Life Cycle Processes [[pdf](https://course.techconf.org/se4485/IEEE/IEEE%2012207%20(2017)%20-%20Software%20Life%20Cycle%20Processes.pdf)]
* IEEE Std 15939: Measurement Process [[pdf](https://course.techconf.org/se4485/IEEE/IEEE%2015939%20(2017)%20-%20Measurement%20Process.pdf)]
* ISO/IEC/IEEE Std 29148-2018: Systems and Software Engineering

§ Life Cycle Processes

§ Requirements Engineering [[pdf](https://course.techconf.org/se4485/IEEE/ISO-IEC-IEEE-29148-2018.pdf)]

Multiple Constraints:

* Project may utilize one data set as long as multiple fields are used to train the predictive analytics model.
* Frontend must conform to WCAG 2.1 Level AA accessibility requirements for visual content (e.g. SC 1.4.1 “Use of Color”, SC 1.4.3 “Contract (Minimum)”, SC 1.4.11 “Non-text Contrast”).

**ADDITIONAL REFERENCES**

* Lattanze, A.J., 2008. Architecting Software Intensive Systems: A Practitioner’s Guide.

CRC Press

* Bass, L., Clements, P. and Kazman, R., 2003. Software Architecture in Practice.

Addison-Wesley