SE 4485: Software Engineering Projects

Fall 2025

Detailed Design 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 Detailed Design for the County Level Air Quality Prediction (CLAP) web application. The CLAP system forecasts next-day county-level Air Quality Index (AQI) categories using historical data provided by the U.S. Environmental Protection Agency (EPA) and a trained machine-learning (ML) model. Building upon the established architecture, this design describes subsystem responsibilities, component interactions, and data flows that support data ingestion, preprocessing, model inference, and user visualization. The design defines interfaces between presentation, application, and data layers, ensuring modularity, maintainability, and traceability to all functional and non-functional requirements.

**TABLE OF CONTENTS**

1. INTRODUCTION 4
2. GUI (GRAPHICAL USER INTERFACE) DESIGN 5
3. STATIC MODEL (CLASS DIAGRAMS) 10
4. DYNAMIC MODEL (SEQUENCE DIAGRAMS) 13
5. RATIONALE FOR YOUR DETAILED DESIGN MODEL 14
6. TRACEABILITY FROM REQUIREMENTS TO DESIGN MODEL 16
7. EVIDENCE OF CONFIGURATION MANAGEMENT 13
8. ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS 15
9. ADDITIONAL REFERENCES 16

**LIST OF FIGURES**

Figure 2.1 – CLAP – Control Panel Example 6

Figure 2.2 – CLAP – Prediction & Category Probability Panel Example 7

Figure 2.3 – CLAP – Historical Trend Panel Example 8

Figure 2.4 – CLAP – Model Information Panel Example 9

Figure 3.1 – CLAP – Backend Route Subsystems Class Diagram 11

Figure 3.2 – CLAP – Domain / DTO Model Class Diagram 12

Figure 4.1 – CLAP – Forecast Generation Sequence Diagram 13

**LIST OF TABLES**

Table 3.1 – CLAP – Backend Route Subsystems & Responsibilities 10

Table 7.1 – GitHub – Single File Revision 13

Table 7.2 – GitHub – Difference Link 13

**INTRODUCTION**

This document presents the detailed design model for the CLAP system. Its purpose is to elaborate on the internal organization and behavior of each subsystem, providing sufficient design detail to guide implementation and verification.

The CLAP system consists of a predictive analytics workflow that ingests daily EPA AQI data, processes it into feature datasets, and applies a gradient boosting decision tree ML model (e.g. LightGBM Regressor) to generate next-day forecasts.

The system architecture follows a layered client-server style, composed of a presentation subsystem for user interaction and visualization, an application subsystem that coordinates data processing & prediction, and a data management subsystem responsible for persistence of AQI data & model artifacts.

This document describes the graphical interface, component structure, control logic, and design rationale that together satisfy system requirements while maintaining portability and accessibility compliance (e.g. WCAG 2.1 AA).

The remainder of this document is organized as follows: graphical user interface (GUI) design, static and dynamic models, rationale for the detailed design model, traceability from requirements to design, configuration management, relevant standards and constraints, and references.

**GUI (GRAPHICAL UESR INTERFACE) DESIGN**

The Graphical User Interface (GUI) provides an interactive environment for users to view, interpret, and explore county-level AQI forecasts. It is designed as a modular and responsive dashboard composed of independent yet integrated display panels that collectively support intuitive data interpretation during user interactions.

KEY DESIGN OBJECTIVES:

* Clarity & Accessibility: Meet WCAG 2.1 AA guidelines for color contrast, focus order, and keyboard navigation.
* Consistency: Apply a unified visual layout and standardized color palette aligned with EPA AQI categories.
* Responsiveness: Adjust dynamically to various screen sizes and devices.
* Feedback: Provide visible status updates for loading, errors, and model metrics.

GUI DESIGN INTENT:

This design emphasizes user experience at the architectural level while abstracting away from specific implementation details or technologies. The interface is organized using a modular subsystem layout, allowing new visualization panels or user-integration elements to be integrated without altering the core framework. Each panel conforms to a consistent API contract and visualization standard, ensuring long-term scalability and design consistency.

The GUI follows a subsystem-oriented design, where each visual component operates as an autonomous unit with a defined purpose and interaction boundary. Collectively, these subsystems create a cohesive and traceable user workflow that supports the project’s quality attributes:

* Accessibility: Promotes visual and navigational inclusivity in compliance with accessibility standards.
* Maintainability: Supports reusable and replaceable interface components for efficient updates.
* Extensibility: Enables future expansion by allowing additional visualization panels or user controls without structural rework.
* Traceability: Aligns each visual function directly with corresponding system and user requirements.

The GUI communicates with backend services through RESTful API endpoints. Each user interaction (e.g. clicking “Refresh Forecast”) triggers asynchronous data retrieval and visualization updates, ensuring real-time responsiveness without requiring a full page reload.

MAJOR INTERFACE PANELS:

* Header Panel: The header panel displays the application’s title and branding elements of choice.
* Control Panel: The control panel provides user interaction mechanisms that allow selection of input parameters and initiation of forecast generation. Its primary purpose is to facilitate configuration and execution of forecast requests by collecting necessary user inputs.
  + Responsibilities:
    - Present available parameters (e.g. model type, county, etc.) in an accessible, structured form.
    - Initiate backend communication for data retrieval, feature generation, and prediction upon user confirmation.
    - Ensure accessibility compliance (e.g. WCAG 2.1 AA) for all interactive controls and maintain clear feedback for user actions (e.g. loading, etc.).

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2.1 – Control Panel Example**

* Prediction Panel: The prediction panel communicates the system’s next-day AQI forecast for the selected county. Its primary purpose is to present results from the predictive ML model in a concise, interpretable format.
  + Responsibilities:
    - Display the predicted AQI category and corresponding index value generated by the model.
    - Provide contextual cues (e.g. “Forecast for <datetime>”) to maintain situational awareness.
    - Convey confidence levels or qualitative AQI category indicators when available (e.g. “Good, Moderate, etc”).
* Category Probability Panel: The category probability panel visualizes the ML model’s confidence distribution across possible AQI categories. This panel supports transparency by revealing the model’s probabilistic reasoning.
  + Responsibilities:
    - Present a probability breakdown for each EPA AQI category (e.g. Good, Moderate, etc.) using clear, color-coded bars or indicators to depict category likelihoods.
    - Apply consistent color mappings with the main forecast display.
    - Support keyboard and screen-reader accessibility for all graphical elements.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2.2 – Prediction & Category Probability Panel Examples**

* Historical Trend Panel: The historical trend panel contextualizes the forecast by displaying recent AQI trends. It provides insight into historical AQI fluctuations that impact current model predictions.
  + Responsibilities:
    - Present a time-series visualization of at least the most recent 30 days of AQI data for the selected county.
    - Enable visual comparison between historical patterns and forecasted values.
    - Use accessible chart colors and labeling aligned with WCAG 2.1 AA guidelines.

A graph on a white background

AI-generated content may be incorrect.

**Figure 2.3 – Historical Trend Panel Example**

* Model Information Panel: The model information panel summarizes metadata about the predictive model currently in use. It supports user understanding of system performance and model validity.
  + Responsibilities:
    - Display key performance metrics such as mean squared error (MSE), root mean squared error (RMSE), and R2 values.
    - Serve as informational reference for evaluating forecast reliably.

A screenshot of a phone

AI-generated content may be incorrect.

**Figure 2.4 – Model Information Panel Example**

CRITICAL RENDERING PATH (OPTIONAL):

The GUI design is capable of minimizing the critical rendering path (CRP) by structuring content and assets for efficient loading and display. For example:

* Static resources (e.g. HTML, CSS, and JavaScript) can be pre-built and served directly from the Flask static directory, allowing the browser to construct the Document Object Model (DOM) and render essential interface elements before asynchronous data fetches complete.
* Charts and tables can be designed to load progressively as data becomes available through RESTful responses, ensuring perpetual performance, where the dashboard appears interactive within seconds even while background data retrieval continues.

This approach aligns with the system’s performance and usability requirements by prioritizing visible content and reducing time-to-interactive.

**STATIC MODEL (CLASS DIAGRAMS)**

The CLAP system’s backend services are structured as modular route subsystems that collectively manage data ingestion, forecasting, and system monitoring. Each subsystem defines a clear responsibility boundary and interacts with the Domain / Data Transfer Object (DTO) layer to maintain consistency between application logic and API responses. This design promotes modularity, reuse, and traceability across backend components.

STATIC MODEL DESIGN INTENT:

The static model defines the internal organization of backend services at the design level, emphasizing interface behavior and subsystem collaboration. The encapsulation of route handlers within the backend enables a clean separation of concerns:

* Routes manage request handling, validation, and response orchestration.
* DTOs define standardized response schemas exchanged between the backend and frontend.
* Shared services (e.g. data access, logging, and model management) provide reusable capabilities without coupling to specific endpoints.

**Table 3.1 – Backend Route Subsystems and Responsibilities**

|  |  |  |
| --- | --- | --- |
| **Subsystem:** | **Endpoint(s):** | **Main Responsibility:** |
| Health Monitoring Service | GET /health | Reports API availability and model readiness status. |
| Data Refresh Service | POST /aqi/refresh | Regenerates stored datasets and updates forecasts in a single workflow. |
| County Registry Service | GET /counties | Returns the list of supported counties and associated metadata. |
| Prediction Service | GET /aqi/predict | Handles forecast requests by performing model inference and returning next-day AQI predictions. |
| Category Registration Service | GET /categories | Serves AQI category definitions consistent with EPA standards. |
| Historical Data Service | GET /aqi/historical | Provides recent AQI values for a specified county and state. |
| Model Metrics Service | GET /model/metrics | Returns model performance indicators (e.g. MSE, RMSE, R2). |

CLASS DIAGRAMS:

Figure 3.1 illustrates the logical relationships among backend route subsystems. Each subsystem acts as a service endpoint responsible for receiving client requests, invoking the appropriate processing logic and returning structured JSON responses.

Dashed dependency lines represent static “uses” relationship between route subsystems and shared services.

A diagram of a system

AI-generated content may be incorrect.

**Figure 3.1 – Backend Route Subsystems Class Diagram**

Figure 3.2 depicts the DTO layer, which defines the data schemas exchanged between backend services and the frontend dashboard. DTOs enforce uniform response formats, encapsulate key attributes (e.g. AQI category, probability scores, timestamps, etc.), and ensure traceability between server outputs and client visualizations.

Each route is designed to return a stable, route-specific JSON schema (e.g. DTO) under normal operation, so that the client can reliably bind data to UI components.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 3.2 – Domain / DTO Model Class Diagram**

**DYNAMIC MODEL (SEQUENCE DIAGRAMS)**

The dynamic model describes how the CLAP system behaves at runtime when a user requests an updated forecast. It shows the order of messages exchanged among the browser UI, backend, route subsystem, shared utilities layer, and the persisted dataset / ML model artifacts.

Figure 4.1 illustrates the runtime interaction for UC2 – Provide County-Level Forecast:

* The sequence begins when the user requests an updated forecast from the browser dashboard.
* The UI issues an HTTP POST /aqi/predict call to the application, which dispatches the request to the Prediction Route Subsystem.
* The route delegates feature construction and data access the Utility / Data Service later, which retrieves the most recent county-level AQI records, generates lag features, and invokes the ML Predictor (e.g. LightGBM model artifact) to obtain the next-day AQI category and associated probabilities.
* The route then formats the result into a route-specific JSON DTO and returns it to the client.
* The browser then updates the dashboard panels without a full page reload, ensuring responsive visualization aligned with SRS FR-3 and FR-4.

**Figure 4.1 – Forecast Generation Sequence Diagram**

**RATIONALE FOR YOUR DETAILED DESIGN MODEL**

The Detailed Design Model refines the architectural description into an implementable form by specifying the internal structure and behavior of system components. It provides the blueprint developers use to construct, integrate, and test the system while maintaining alignment with requirements and architectural goals.

* It expands the architecture (e.g. Client-Server + Layered) into concrete modules and classes.
* It defines each route (e.g. *predict.py*, etc.) and utility (e.g. *aqi\_utils.py*) in terms of responsibility, interface, and interaction.
* It specifies sequence diagrams for runtime behavior, and class diagrams for static structure.
* It ensures that every function, route, and data exchange (e.g. DTO) can be traced back to its’ requirement (e.g. FR, NFR).

**TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL**

The Detailed Design Model provides a concrete realization of each functional and non-functional requirement defined and mapped through the Architecture. This section establishes direct traceability between requirements and the design-level elements implemented in the CLAP system (e.g. modules, classes, functions, and data interactions). A traceability matrix is shown below:

**EVIDENCE THE DESIGN MODEL 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 | 10/20/25 | v0.1 () | v0.2 () | Jay Chung (cwc130330) | Initial draft | All Team Members |
| 2 | 10/31/25 | v0.2 () | v0.3 () | Jay Chung (cwc130330) | Revision Edit – All sections | All Team Members |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |

**Table 7.2 - Difference link**

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

**ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS**

Engineering Standards:

* IEEE Std 1016-1998-(Revision-2009): Software Design [[pdf](https://course.techconf.org/se4485/IEEE/IEEE-Std-1016-1998-(Revision-2009)-Software-Design.pdf)]

Multiple Constraints:

* Project may utilize one data set so 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**

[1] Larman, C., 2012. Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development. Pearson Education

[2] Hyman, B., 1998. Fundamentals of Engineering Design. New Jersey: Prentice Hall

[3] Simon, H.A., 2014. A Student's Introduction to Engineering Design: Pergamon Unified Engineering Series (Vol. 21). Elsevier