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) 12
4. DYNAMIC MODEL (SEQUENCE DIAGRAMS) 15
5. RATIONALE FOR YOUR DETAILED DESIGN MODEL 16
6. TRACEABILITY FROM REQUIREMENTS TO DESIGN MODEL 16
7. EVIDENCE OF CONFIGURATION MANAGEMENT 13
8. ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS 14
9. ADDITIONAL REFERENCES 14

**LIST OF FIGURES**

Figure 2.1 – CLAP – Example Control Panel 5

Figure 2.2 – CLAP – Example Prediction & Category Probability Panel 6

Figure 3.1 – CLAP – Backend Package & Routes Class Diagram 12

Figure 3.2 – CLAP – Domain / DTO Class Diagram 13

Figure 4.1 – CLAP – Forecast Generation Sequence Diagram 15

**LIST OF TABLES**

Table 2.1 – CLAP – Summary of Dropdown Controls & Action Buttons 7

Table 2.2 – CLAP – Summary of Dropdown Controls & Action Buttons 11

Table 3.1 – CLAP – List of Backend Route Modules & Endpoints 14

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 explore and interpret county-level air quality forecasts. The dashboard is organized into modular display panels that work together to communicate key insights clearly and efficiently.

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.

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 – Example Control Panel**

* 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 – Example Prediction & Category Probability Panel**

* 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.

**Figure 2.3 – Example Historical Trend Panel**

**Table 2.2 – Summary of Displayed Metrics**

|  |  |  |
| --- | --- | --- |
| **Metric:** | **Value:** | **Description / Interpretation:** |
| Algorithm | LightGBM Regressor (Gradient Boosting Decision Tree) | Ensemble method that builds decision trees iteratively to minimize prediction error. |
| Mean Squared Error (MSE) | 93.47 | Average squared prediction error (smaller value means lower error). |
| Root Mean Squared Error (RMSE) | ~10 AQI units | Average prediction error in physical AQI units (smaller value means higher accuracy). |
| Coefficient of Determination (R2) | 0.72 | 72% of the AQI variance is explained by the model – This indicates strong predictive performance. |

* Footer Section:
  + Data Structure: <footer className="bg-slate-900 text-slate-200 py-8 mt-auto">

**STATIC MODEL (CLASS DIAGRAMS)**

The backend route modules of the CLAP system interact with the Domain / Data Transfer Object (DTO) layer. The Backend Package & Routes class diagram (Please see Figure 3.1) shows how each Flask module (e.g. *predict*, *historical*, *model\_metrics, counties*, *categories*, *health*, etc.) acts as a logical service responsible for handling client requests (*app*) and producing JavaScript Object Notation (JSON) responses.

A diagram of a software application

AI-generated content may be incorrect.

**Figure 3.1 – Backend Package & Routes Class Diagram**

The route module response structures (e.g. *predict*, etc.) are represented by corresponding Data Transfer Object (DTO) classes (e.g. *PredictResponse*, *HistoricalResponse*, *ModelMetricsResponse*, *CountiesResponse*, *CategoriesResponse*, *HealthResponse,* etc.). Each DTO encapsulates the data elements returned to the frontend (e.g. JSON), defining a consistent schema for communication between the system’s backend and presentation layers.

A computer screen shot of a data transfer

AI-generated content may be incorrect.

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

The encapsulation of the route modules within the *backend.routes* package highlights the modular design of the system, where the routes focus on request handling, while the DTOs define the response payloads (e.g. every 1-to-1 mapping between each endpoint and respective JSON output).

The table below describes each file that corresponds to a Flask route module, including their behavior and/or purpose.:

**Table 3.1 – List of Backend Route Modules & Endpoints**

|  |  |
| --- | --- |
| **File:** | **Endpoints / Purpose:** |
| aqi\_utils.py | Provides shared utilities used across routes:   * *logger()* and *log\_event()* wrappers for structured logs. * *ds()* to access the data source (*data\_source.py*). * *predictors()* and *get\_predictor()* for model access. * *Build\_features\_from\_recent()* and *iterative\_forecast()* for feature creation and forecasting logic. |
| categories.py | GET */categories* 🡪 Returns the AQI category configuration defined in *Config.AQI\_CATEGORIES*. |
| counties.py | GET */counties* 🡪 Returns a list of all counties derived from the dataset. |
| errors.py | Registers centralized error handlers (e.g. 404, 500) for consistent API responses. |
| health.py | GET */health* 🡪 Returns system status JSON including timestamps and model-loaded flag. |
| historical.py | GET */aqi/historical…* 🡪 Returns recent AQI values for the specified county and state (30-days by default). |
| index.py | GET / 🡪 Serves *frontend/dist*/*index.html* |
| model\_metrics.py | GET */model/metrics…*  🡪 Returns stored model performance metrics (e.g. MSE, RMSE, and R2). |
| predict.py | POST */aqi/predict* 🡪 Accepts JSON *{county, state, model, days}* and returns next-day AQI category predictions. |
| refresh.py | POST */aqi/refresh* 🡪 Runs a combined refresh that regenerates historical data and predictions in a single pipeline. |

**DYNAMIC MODEL (SEQUENCE DIAGRAMS)**

The sequence diagram illustrates the dynamic behavior of the CLAP system by showing how components interact over time to fulfill requests. This diagram captures the chronological flow of messages between the frontend, backend routes, helper utilities, and machine learning components (e.g. *Predictor* instance).

A diagram of a software application

AI-generated content may be incorrect.

**Figure 4.1 – Forecast Generation Sequence Diagram**

When a user interacts with the browser interface (e.g. clicking “Generate Forecast”), the frontend React application sends HTTP requests to the Flask backend, which dispatches them to the appropriate backend route module (e.g. *predict.py* or *historical.py*). These route handlers invoke shared utility functions (e.g. *aqi\_utils.py*) to perform tasks such as retrieving the appropriate model (e.g. *get\_predictor()*), constructing features, and generating predictions using the LightGBM model (e.g. *predict()* or *predict\_proba()*). The results are then formatted into structured JSON responses and returned to the frontend, which renders them as interactive charts and forecast summaries.

(NOTE: Vite’s production build complies the React frontend into a static bundle of optimized HTML, CSS, and JavaScript assets that can be served by the backend (Flask) as a static site).

**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) |  | All Team Members |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |

**Table 7.2 - Difference link**

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