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 presents the detailed design for the County Level Air Quality Prediction (CLAP) web application, which predicts next-day county-level Air Quality Index (AQI) categories using historical data provided by the U.S. Environmental Protection Agency (EPA). Building upon the software architecture, this document focuses on the data structure, component behavior, sequence of operations, and traceability from design to requirements. This design defines subsystem interactions through well-defined interfaces to support data ingestion & preprocessing, model inference, dashboard visualization, and user interaction features.

**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 – Browser 5

Figure 2.2 – CLAP – Header 5

Figure 2.3 – CLAP – Select Location & Model 6

Figure 2.4 – CLAP – Next-Day AQI Prediction 8

Figure 2.5 – CLAP – Category Probabilities 9

Figure 2.6 – CLAP – 30-Day Historical AQI Trend 10

Figure 2.7 – CLAP – Model Information 11

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 County Level Air Quality Prediction (CLAP) web application. Its primary purpose is to define the internal structure, component behavior, and data interactions that realize the system’s components. The design expands the software architecture previously established to provide a blueprint for implementation, integration, and testing.

The CLAP system is a predictive analytics web application that forecasts next-day Air Quality Index (AQI) categories at the county level. It ingests historical AQI data from the U.S. Environmental Protection Agency (EPA), processes and stores it in a local persistence layer, and uses a trained LightGBM machine-learning model to generate predictions. The detailed design describes how the frontend, backend, data layer, and dashboard components interact through defined interfaces to deliver next-day AQI forecasts.

The CLAP detailed design implements a modular, layered client-server approach that ensures clear separation of concerns among presentation, application logic, and data management. The design emphasizes modularity, enabling lightweight local deployment while supporting educational objectives in software architecture, machine learning, and system integration.

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.

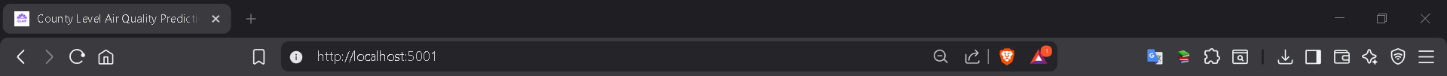
**GUI (GRAPHICAL UESR INTERFACE) DESIGN**

The Graphical User Interface (GUI) is designed for clarity, accessibility, and responsiveness to meet WCAG 2.1 AA standards.

* Clarity & Accessibility: Meet WCAG 2.1 AA (e.g. color contrast, keyboard navigation, focus order, aria-labels).
* Consistency: Use a unified visual language (e.g. Tailwind utility classes, CLAP color palette mapped to AQI categories).
* Feedback & Resilience: Loading states and error banners.
* Observability: In-UI surfacing of model metrics and timestamps.

SCREENS & COMPONENTS (Screens utilize data obtained from 10/28/2025):

* Web Browser Window:



**Figure 2.1 – Browser**

* + Address Bar: URL shows <http://localhost:5001> (Flask backend – protocol (http://), host (localhost), port (5001)).
  + Favicon: The site’s icon – CLAP logo.
  + Tab Title: <title>County Level Air Quality Prediction</title> //html
* Header Section:

A purple background with white text

AI-generated content may be incorrect.

**Figure 2.2 – Header**

* + Header Logo: CLAP header logo “*clap\_logo.png*”(from *frontend/src/assets*):   
    *<className="w-20 h-20 md:w-24 md:h-24 rounded-md shadow-md"…>.*
  + Header Body: Header title element: *<h1 className="text-3xl md:text-5xl font-extrabold flex items-center gap-3">* is wrapped inside page container: *<div className="min-h-screen bg-slate-50 flex flex-col">*, displayed on purple gradient banner *<header className="bg-gradient-to-r from-indigo-500 to-purple-500 text-white py-10">*, wrapped inside content wrapper: *<div className="container-xxl flex flex-col items-center text-center">*.
* Body Section:
  + Body Component #1 – Select Location & Model: This section displays form-like inputs for configuring parameters and buttons for generating AQI forecasts.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2.3 – Select Location & Model**

* + - Data Structure: Card contains custom state components that are presented in the form of **Dropdown Controls** (e.g. Model, County, Forecast Period) and **Action Buttons** (e.g. Generate Forecast, Load Data). Please see Table 2.1 for more information on dropdown control and action button features.
      * Wrapped inside main container: *<main className="container-xxl -mt-6 mb-16 space-y-6">*, wrapper card: *<section className="card shadow-md hover:shadow-xl border border-slate-200 transition-shadow">* & control grid: *<div className="grid md:grid-cols-4 gap-4">*, is titled “Select Location & Model”: *<h2 className="section-title">Select Location & Model</h2>*.

**Table 2.1 – Summary of Dropdown Controls & Action Buttons**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name (Ctrl ID):** | **UI Element:** | **Actions:** | **Data Binding (React):** | **On-Click Behavior:** | **Description:** |
| Model (*model*) | Dropdown (*<Select>*) | LightGBM 🡪  (value = “balanced”) | *model*, via *setModel(v)* | Calls *doRefresh(selected, days, v)* | SelectsML model. Update triggers new forecast refresh. |
| county (*county*) | Dropdown (*<Select>* | Dynamically populated via *countyOptions.map()* | *selected*, via *setSelected(v)* | Calls *doRefresh(v, days, model)* | Lets user choose target county and state for AQI forecasting. Update refreshes results based on location. |
| Forecast Period (*days*) | Dropdown (*<Select>* | Next Day 🡪  (value = 1) | *days*, via *setDays(d)* | Converts value to *Number(v)*, calls *doRefresh(selected, d, model)* | Defines how far ahead the AQI prediction is generated. Only 1 day forecast available on current model. |
| Generate Forecast (N/A) | Button (*<button>)* | N/A | Current state: (*model, selected, days*) | Calls *doRefresh()* | Sends current selections to backend and retrieves next-day forecast. |
| Load Data (N/A) | Button (*<button>)* | N/A | Uses *selected* county | Calls *doLoad(selected)* | Loads or refreshes cached data for chosen county. |

* + Body Component #2 – Next-Day AQI Prediction: This section renders and displays AQI Prediction results (*prediction*) dynamically, showing a conditional AQI prediction depending on the value of *days* (value = 1, by default).

A screenshot of a phone

AI-generated content may be incorrect.

**Figure 2.4 – Next-Day AQI Prediction**

* + - Data Structure: Left card: *<div className="card ... md:col-span-1">* contains and renders reusable state component: <*PredictionCard>*.
      * Wrapped inside conditional wrapper: *{prediction && ( ... )}*, control grid: *<section className="grid md:grid-cols-3 gap-6">,* is titled “Next-Day AQI Prediction”: *<h2 className="section-title">Next-Day AQI Prediction</h2>.*
  + Body Component #3 – Category Probabilities: This section renders and displays probability bars for each AQI category (e.g. Good, Moderate, etc.) dynamically, showing conditional probabilities depending on the value of *days* (value = 1, by default).

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2.5 – Body Component #3: Category Probabilities**

* + - Data Structure: Right card: *<div className="card ... md:col-span-2">* contains and renders reusable state component: *<ProbabilitiesList>* or *<MultiDayChart>* (NOTE: Multi-day option currently not available).
      * Wrapped inside conditional wrapper: *{prediction && ( ... )}*, control grid: *<section className="grid md:grid-cols-3 gap-6">,* is titled “Category Probabilities”: *<h2 className="section-title">Category Probabilities</h2>* .
  + Body Component #4 – 30-Day Historical AQI Trend: This section renders and displays probability bars for each AQI category (e.g. Good, Moderate, etc.) dynamically, showing conditional probabilities depending on the value of *days* (value = 1, by default).

A screenshot of a graph

AI-generated content may be incorrect.

**Figure 2.6 – 30-Day Historical AQI Trend**

* + - Data Structure: Card: *<section className="card shadow-md hover:shadow-xl border border-slate-200 transition-shadow">* contains chart component with smooth blue AQI line, shaded area under line, and green/yellow markers that correspond to category thresholds.
      * Wrapped inside container: *<main className="container-xxl -mt-6 mb-16 space-y-6">*, is titled “30-Day Historical AQI Trend”: *<h2 className="section-title">30-Day Historical AQI Trend</h2>*.
    - Component Summary: Chart component: *<AqiChart>* (e.g. Chart.js) plots AQI changes over time, utilizing array of historical AQI data (e.g. *history*). Color-key legend: *<AqiLegend>* displays a static reference of EPA AQI Category colors based on severity levels.
  + Body Component #5 – Model Information: This section displays the statistical performance of the machine learning algorithm used for AQI forecasting, using reusable display component *<StatsStrip>*.

A white rectangular object with black text

AI-generated content may be incorrect.

**Figure 2.7 – Model Information**

* + - Data Structure: Card: *<section className="card shadow-md hover:shadow-xl border border-slate-200 transition-shadow">* displays defined metrics (e.g. *algorithm, mse, rmse, r2*).
      * Wrapped inside container: *<main className="container-xxl -mt-6 mb-16 space-y-6">*, is titled “Model Information”: *<h2 className="section-title">Model Information</h2>*.

**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 = better). |
| Root Mean Squared Error (RMSE) | ~10 AQI units | Average prediction error in physical AQI units (smaller = more accurate). |
| 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