CW3 Explanation Document

Word Count: 993

Student ID: s2337850

Contents

1 Introduction and Architecture

1.1 Project Overview

This project addresses the lack of visual feedback in CW2, where routing behaviour, explored nodes and drone allocation were difficult to interpret. It extends CW2 by adding a real-time, interactive visualisation layer that reveals these internal processes on an intuitive map. Users can submit delivery orders, inspect constraints and monitor drone activity through a clear control panel, improving transparency and supporting smoother collaboration between operators, planners and automated decision-making systems.

1.2 System Architecture

The system adopts a client–server design combining a React–TypeScript frontend with the existing CW2 Spring Boot backend.As shown in Figure 1, CW2 retains responsibility for all core logic, including drone availability, constraint handling and A* pathfinding. These results are exposed through REST endpoints for batch calculations and a dedicated WebSocket channel for incremental progress updates. The frontend is modularised into map rendering, control panels, performance monitoring and animation components. Incoming REST responses and streamed WebSocket events are integrated into a unified state layer, enabling smooth synchronisation between UI updates and backend computation. This layered architecture enforces clear module boundaries, supports extensibility and enables high-frequency real-time visualisation.

1.3 Technology Stack Justification

React and TypeScript provide strong typing, reusable UI components and efficient rendering suited to dynamic map interactions. Leaflet offers lightweight geographic visualisation with precise control over markers, paths and restricted zones. The frontend is built using a modern Node.js toolchain(Vite), ensuring fast development and dependency management. Spring Boot ensures reliable REST and WebSocket support while integrating seamlessly with CW2 logic.

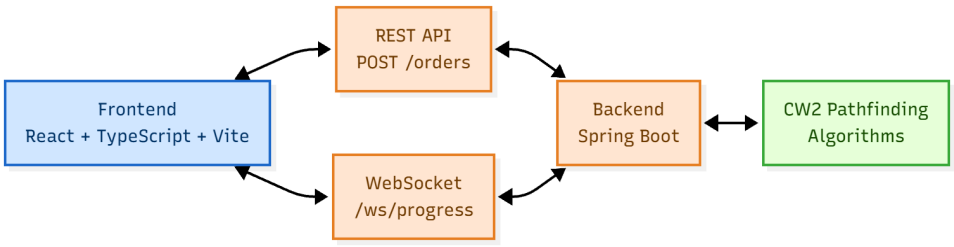

Figure 1: System Architecture Overview

2 Implementation Details

2.1 Frontend Implementation

The frontend is organised into a set of modular, reusable React components that map directly to core functional areas of the system. The Map module handles all geographic elements, including drone animation, explored nodes, restricted areas and dynamically updated flight paths rendered through Leaflet. The Control module manages user inputs such as order creation, demo scenario loading and calculation triggers, providing a structured interface for dispatch planning. The Monitor module visualises backend progress through a live event stream, presenting node exploration logs, path-found notifications and batch completion messages. Finally, the Layout module defines the overall page composition, ensuring a coherent and responsive interface across the entire application.

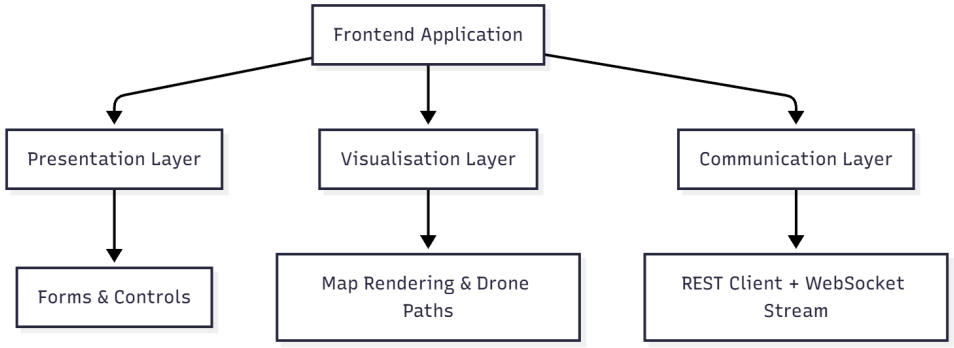

Figure 2: Frontend layered architecture

State is managed using React hooks, with local component state used for UI interaction and shared state held in higher-level containers. Derived values such as computed metrics, animation frames and filtered order lists are memoised using useMemo and useCallback to reduce unnecessary recomputation. WebSocket updates are processed incrementally and merged into an immutable state structure, ensuring predictable rendering behaviour. Event throttling and selective map re-rendering are applied to avoid performance bottlenecks during high-frequency pathfinding updates, enabling smooth animation even when hundreds of nodes are explored.

Real-time behaviour is achieved through a WebSocket client that subscribes to progress messages streamed by the backend. Message types include node exploration, path-found events, batch-start and batch-complete notifications. The frontend updates visual elements incrementally, including appending explored nodes to the map, extending drone trajectories and refreshing metrics panels. Rendering is optimised by limiting the number of visible nodes and grouping updates into small batches, preventing excessive DOM operations while maintaining accurate, step-by-step animation of the A* search process.

2.2 Backend Implementation

The backend extends the CW2 Spring Boot service and exposes endpoints that encapsulate drone availability checks, constraint validation and delivery path computation. The primary endpoint, /api/v1/calcDeliveryPathAsGeoJson, triggers CW2's A* pathfinding logic and returns results in a structured GeoJSON format suitable for map rendering. Other endpoints provide service point information, restricted areas, drone configurations and order submission. DTO classes isolate transport formats, while the service layer encapsulates domain logic behind stable interfaces. This layered structure enforces clear boundaries between API, domain and infrastructure concerns, improving maintainability and clarity.

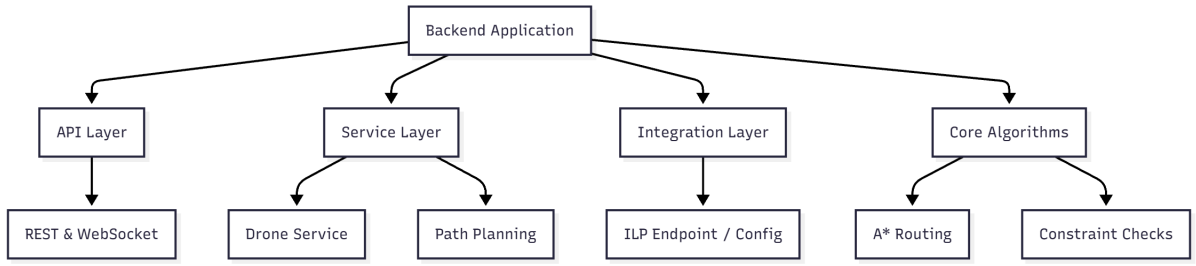

Figure 3: Backend layered architecture

The backend includes a dedicated WebSocket module that implements an event-driven channel for incremental pathfinding progress. As the A* algorithm expands nodes, events are published through a PathfindingProgressHandler, which pushes structured messages to all subscribed clients. Progress updates include node coordinates, cost accumulations, batch boundaries and path-found signals. This push-based model eliminates polling and greatly improves responsiveness, enabling the frontend to animate algorithmic behaviour in real time. The WebSocket channel is lightweight, stateless and efficient, supporting concurrent visualisation sessions without interfering with core computation.

3 Main Features

The system provides an interactive and real-time visualisation environment for analysing drone deliveries and CW2 pathfinding behaviour. Users can create delivery orders, load preset scenarios, and trigger route computation directly through a structured control panel. A dynamic map displays service points, restricted areas, explored nodes, and multi-drone flight paths using smooth Leaflet-based animations. Real-time progress is

streamed via WebSocket, enabling detailed inspection of A* exploration patterns and batch-level execution. Complementary performance metrics summarise total cost, moves, node expansions, and per-drone delivery statistics. Together, these features transform CW2's previously opaque internal processes into an intuitive, observable and explorable workflow.As shown in Figure 4.

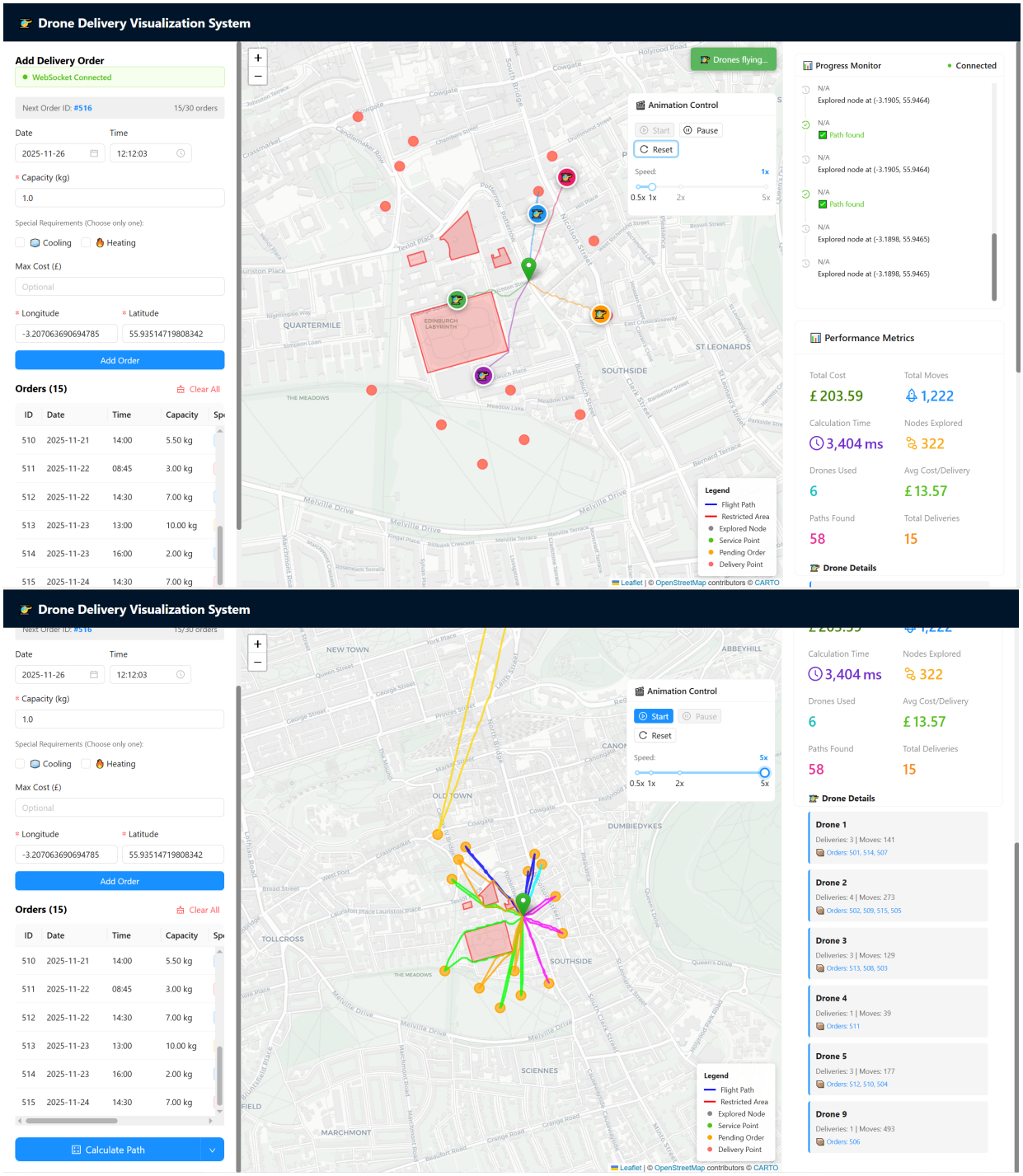

Figure 4: Drone Delivery Visualization System

4 Error Handling and Robustness

This system applies input validation and runtime safeguards to ensure reliable operation. Invalid delivery locations trigger immediate warnings, and communication failures surface through clear UI feedback. Additional guidance is provided for map or data issues, helping maintain stable real-time visualisation.As shown in Figure 5.

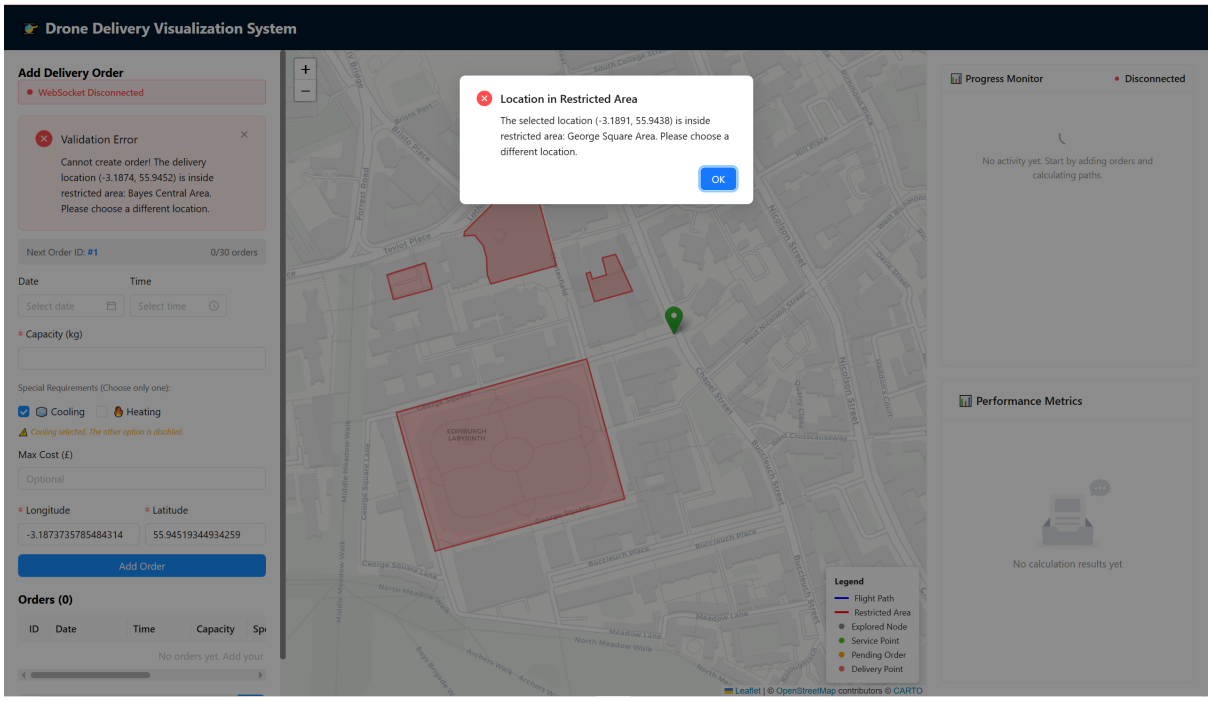

Figure 5: Error Feedback

5 Conclusion

This project delivers an interactive, real-time visualisation system that extends the CW2 backend with a map-based interface for analysing drone deliveries. Users can add delivery orders directly through an integrated control panel, enabling immediate computation and visual feedback of A* pathfinding, drone allocation, explored nodes and operational constraints. This transforms CW2's previously opaque logic into a transparent and engaging workflow.

Future improvements may include predictive routing, support for 3D terrain data. Additional enhancements such as historical replay, richer analytics dashboards and machine-learning-based demand forecasting could further extend the system to large-scale, city-level logistics environments.