CW3 Explanation

Innovation / Idea / Benefit

The system is an interactive web-based visualisation and scenario-planning tool for the ILP drone planner. Its key innovation is that it converts hard to read large JSON coordinate outputs, into an animated, step-based, geospatial playback. This allows users to understand how routes evolve over time and whether they adhere to the coursework's specification.

The tool targets two groups:

- **Lecturers/markers,** who need an intuitive way to verify specification constraints such as restricted areas, service points, delivery order, and drone behaviour.
- **Students/developers,** who need fast visual feedback when debugging their software.

The planner page lets users define dispatch scenarios, submit them to the backend, and visualise how routes change. This supports experimentation and highlights correctness issues that are difficult to detect in raw ILP output.

Compared with static GeoJSON maps, console logs, or non-interactive images the main novelty is temporal animation, stepwise rendering, and a scenario-creation frontend integrated with the solver. Together, these features make the path finding behaviour transparent in a way that static visualisation tools do not. This gives the student faster debugging cycles and allows them to

catch subtle edge cases. It also allows the markers to grade route behaviour, not just seeing an auto marked result.

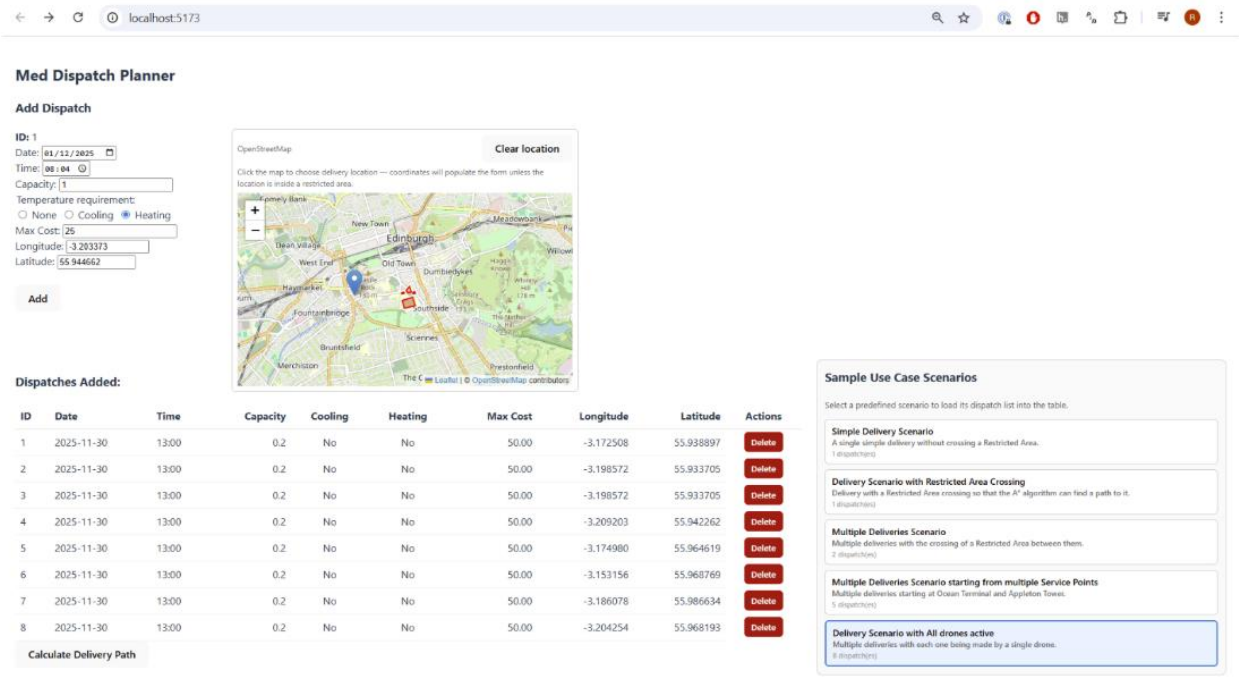
Execution / Implementation

The frontend uses React + Vite, react-leaflet, and OpenStreetMap.

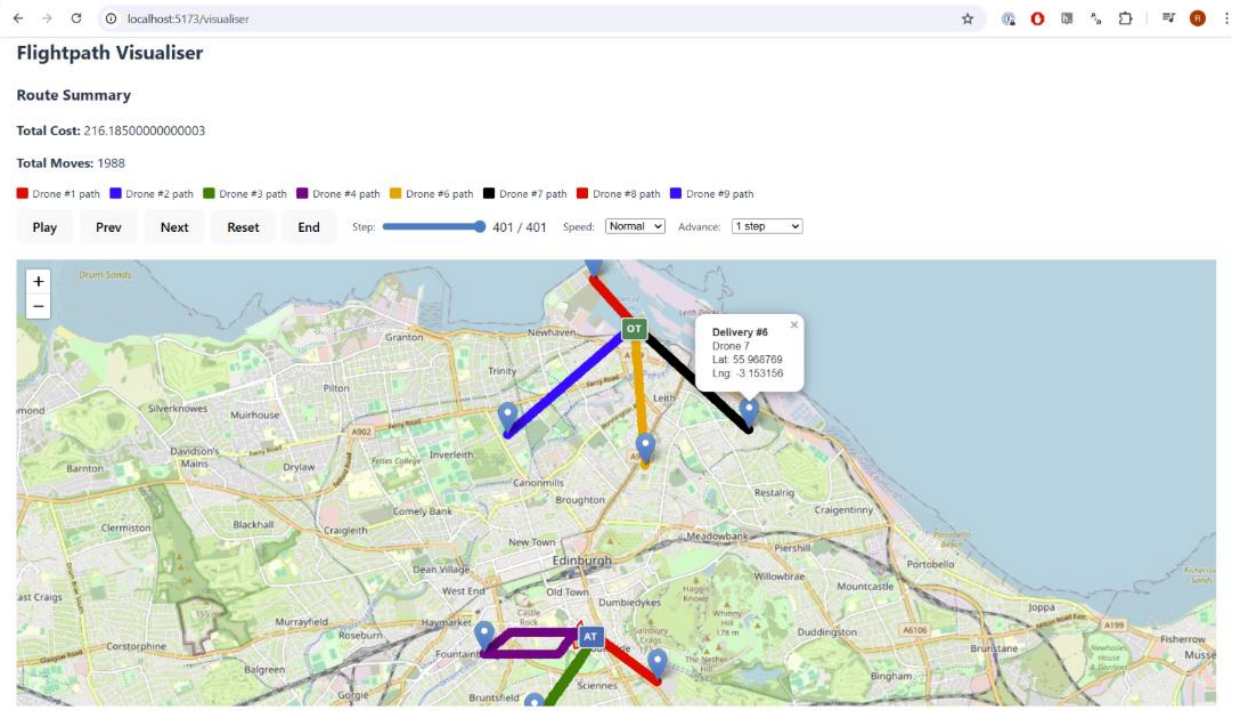
React's component model and hooks support the playback UI; Vite enables fast development; react-leaflet provides map rendering. JSON is used for data exchange. AI assistance was used during planning due to my unfamiliarity with the tools used.

Planner -> Backend -> Visualiser workflow

The planner page allows users to add and delete dispatch requests. Each field entered into the dispatch form has error checking to ensure correct user input is added. A selection of predefined scenarios exists, implemented as buttons with a small description, once pressed the planner page receives and then displays that scenario from dispatch-scenario.json.



When submitted, the dispatch list is sent via Rest to the backend solver. After the backend returns its solution, the app navigates to the Visualiser page and passes the payload. This cleanly separates scenario creation from result interpretation.



Core implementation details

Visualisation logic is in Visualiser.jsx and FlightpathLayer.jsx. Results are normalised per drone for flight paths and delivery points. The normalisation attempts to show as much data as possible, so if given a malformed JSON, it will still show some of the intended flightpaths.

Visualiser playback engine

This manages the:

- Step, how many points of each path are currently visible
- Playing, whether the animation is running
- speedMs, delay between ticks
- stepDelta, number of steps per tick

A useEffect with a setTimeout advances the animation when playing, the step is advanced by stepDelta at the chosen speed. timerRef keeps track of the timer to ensure proper cleanup on pause.

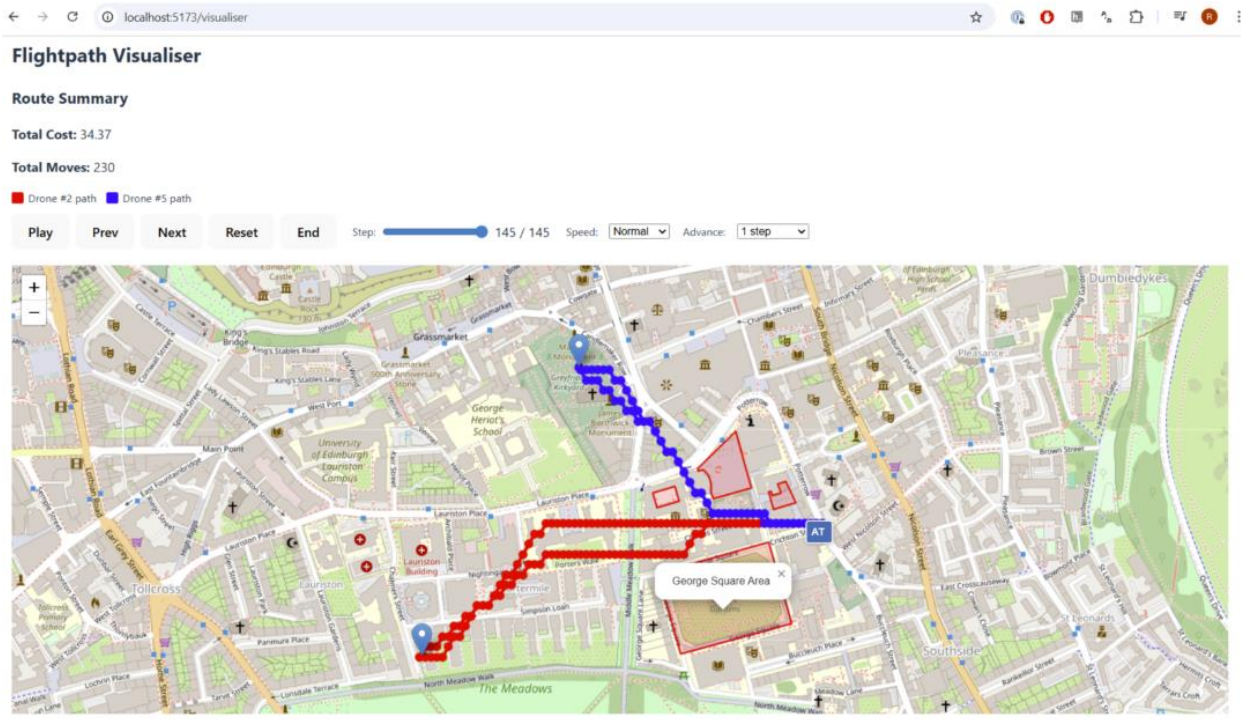
User controls

A toolbar provides:

- Play/Pause, Prev, Next, Reset, End buttons
- A range slider of the step
- A speed selector (Slow to Very fast) and step-advance options

These controls give the user full control over how they explore each route, which is particularly useful when verifying flight paths.

Flightpathlayer renders polylines for each drone, coloured per drone; delivery points are highlighted as markers. Restricted areas are rendered from a GeoJSON FeatureCollection and shown visually for constraint checking. This visually emphasises where the ILP constraints should prevent flight, directly supporting debugging and assessment.



The separation of concerns between data normalisation, playback logic and map rendering demonstrates consistent use of modern web development practices. If the given ILP solver fails to provide a solution a page showing No results loaded is shown.

Completeness

The tool supports the complete workflow:

1. The user plans a dispatch scenario
2. The backend ILP returns total cost, total moves, and each drone's routes
3. The frontend receives this result via routing state and opens the Visualiser page which displays totals, legend, restricted areas, service points, delivery points, and animated paths
4. The user can play, step through, and inspect final flight paths

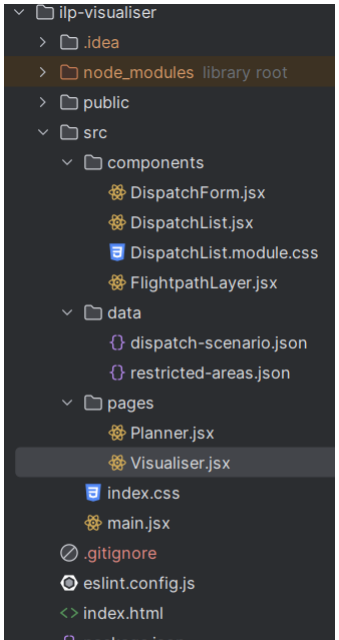
This covers many aspects of an anticipated scenario: multi-drone routing, restricted areas, drone service points, and time-based visualisation. The tool is deliberately focused on route understanding rather than full mission planning UI. All key data relevant to spatial reasoning is visualised and shown.

The end-to-end flow starts in the Planner page, where the user creates one or more dispatch requests, or selects a predefined scenario. The backend solves the dispatches and returns a structured result. The application then routes the user to the Visualiser with this result, where they can inspect total cost, moves, and animated paths. This workflow covers the full anticipated use case: defining a scenario, running the optimisation, and analysing the resulting planned flight paths.

Style and Architecture

The frontend is structured into small, focused files:

- pages/Planner.jsx and pages/Visualiser.jsx for main pages
- components/Flightpathlayer.jsx, DispatchForm.jsx, DispatchList.jsx for reusable components



- data/... for static JSON files
- index.html, main.jsx, index.css for application bootstrap and

  global styling

This separation adheres to the standard React architecture: page-level components handle routing and high-level state, while presentational components focus on rendering.

The architecture separates scenario input from result inspection. Planner components deal with form handling, validation, and REST calls to the Spring Boot backend, while Visualiser components deal exclusively with rendering and playback of already-computed results. This separation improves testability and keeps each part of the UI focused and simpler.

The frontend is decoupled from the backend. It expects the specification defined JSON result but does not depend on a specific server implementation. This design makes it easy to test the visualiser independently with sample data, and it would be straightforward to plug into different ILP solvers.

Overall, the project demonstrates a coherent style, clear component boundaries, and appropriate use of modern frontend tools to support the course's optimisation scenario.