ILP CW2 Implementation Essay

Guolong Tang, s2286997

1 Introduction

This assignment extends the CW1 Java REST service by adding three main endpoints: `/validateOrder`, `/calcDeliveryPath`, and `/calcDeliveryPathAsGeoJson`. Unit and integration tests were enhanced, and the final project is packaged in a Docker image (`amd64`) to ease deployment across systems.

2 What Was Implemented

1. **Order Validation** (`/validateOrder`): Examines each order's pizzas, restaurant constraints (including opening days), and credit card information. Returns a JSON payload with an `orderStatus` and `orderValidationCode`.
2. **Delivery Path Calculation** (`/calcDeliveryPath`): Given a valid order, computes a drone flight path from the restaurant to Appleton Tower, avoiding no-fly zones and restricting the drone to remain within the central area if it ever enters it. Hover steps are appended at origin and destination.
3. **GeoJSON Delivery Path** (`/calcDeliveryPathAsGeoJson`): Provides the same path as above but in GeoJSON `FeatureCollection` format (excluding hover steps).

3 Why It Was Done

Each endpoint fulfills a clear specification requirement: precise validation ensures correct error reporting; path computation addresses geographical constraints (no-fly zones, central area rule); and GeoJSON output caters to standard mapping tools. Minimal overhead is added to maintain performance.

4 How It Was Done (Technical Details)

Overall Architecture: Implemented with Spring Boot. Two controllers handle new endpoints: `OrderValidationController` and `DeliveryPathController`. Each delegates complex logic to dedicated services for clear separation of concerns.

Order Validation: Encapsulated in `OrderValidationService`, it checks `pizzasInOrder` for matching restaurant menus, ensures the order date corresponds to restaurant openings, verifies prices (including delivery fee), and validates credit card fields (length, expiry parsing, CVV check). If any check fails, an `OrderStatus.INVALID` and relevant `OrderValidationCode` is returned.

Pathfinding (A* Search): Implemented in `CalcDeliveryPathService`. The restaurant location is determined by verifying which single restaurant can supply all pizzas. We then run A* on a discretized grid around Edinburgh, using 16 directions and a step size of 0.00015 degrees. Each potential move is validated:

- It must not cross no-fly zones, which are stored and checked via `NoFlyZoneService`.
- Once inside the central area (checked via `CentralAreaService` and a point-in-polygon test), the path cannot exit again.

A small tolerance ensures the drone can reach Appleton Tower if it is within 0.00015 degrees. Hover steps are inserted at the start and end positions.

GeoJSON Construction: For `/calcDeliveryPathAsGeoJson`, the path from the A* result is converted into a GeoJSON `FeatureCollection` containing a single `LineString`. The primary difference is the exclusion of hover duplicates.

5 How It Was Tested

Integration Tests: In `EndpointsIntegrationTest`, test orders are fetched from `https://ilp-rest-2024.azurewebsites.net/orders` and sent to `/validateOrder`, `/calcDeliveryPath`, and `/calcDeliveryPathAsGeoJson`. Responses are checked for:

- Correct status codes (`200` for valid inputs, `400` for invalid).
- Matching `orderStatus` and `orderValidationCode`.
- Timely completion (under five seconds).

6 Conclusion

The final system satisfies CW2 requirements: detailed validation, robust path planning, and GeoJSON output. By isolating logic in service classes and ensuring high coverage via integration tests, the application remains modular, testable, and scalable. Docker packaging further simplifies deployment and consistency across different environments.