LO1: Analyse requirements to determine appropriate testing strategies

Introduction

This document outlines the requirements for a medical dispatch delivery service that I have identified. It then identifies what level of requirement that each one is, if it is system, integration or unit. Then it identifies an appropriate test approach for a chosen selection of requirements. Then, finally, assessing the appropriateness of that test approach chosen. Many of these requirements align with the specification provided to me, which can be found in the Git repository, titled ILP CW2 - 3 - 4 Spec.pdf.

1.a. Range of requirements, functional requirements, measurable quality attributes, qualitative requirements

Functional requirements (FR)

FR1: The system shall provide an endpoint GET /api/v1/uid that returns the programmer's UID.

FR2: The system shall provide POST /api/v1/distanceTo that accepts a JSON body containing two coordinate pairs of longitude and latitude and returns the Euclidean distance between them.

FR3: The system shall provide POST /api/v1/isCloseTo that accepts two coordinates and returns a boolean indicating whether their distance is within 0.00015 degrees.

FR4: The system shall provide POST /api/v1/nextPosition that accepts a start coordinate and an angle, then returns the next coordinate after moving 0.00015 degrees. The angle must be one of the supported compass directions (multiples of 22.5 degrees).

FR5: The system shall provide POST /api/v1/isInRegion that accepts a coordinate and a Region polygon and returns whether the point is inside the region. The region must be a closed polygon (first vertex equals last vertex) with at least 4 vertices.

FR6: The system shall retrieve required planning inputs from an external ILP service:

- Drones
- service points
- drones for service points
- restricted areas

FR7: The system shall allow querying drones based on their attributes/capabilities, including:

- GET /api/v1/dronesWithCooling/{state} to list drone IDs matching a cooling capability state
- GET /api/v1/droneDetails/{id} to return full drone's details for a given ID (or an error if unknown)
- GET /api/v1/queryAsPath/{attribute-name}/{attribute-value} to filter drones by a named attribute/value pair
- POST /api/v1/query to filter drones based on a list of query constraints

FR8: The system shall provide POST /api/v1/queryAvailableDrones that accepts a list of medical dispatch requirement objects and returns drone IDs that can satisfy those requirements, given drone/service-point availability.

FR9: The system shall provide POST /api/v1/calcDeliveryPath that accepts a list of medical dispatch records and returns a calculated delivery flight path, a total cost and total moves.

FR10: The system shall provide POST /api/v1/calcDeliveryPathAsGeoJson that accepts a list of medical dispatch records completable by one drone and returns the calculated delivery path encoded as a GeoJSON object.

FR11: When generating a path from a start to a goal location, the system shall avoid restricted areas by producing a path that does not intersect restricted areas.

Measurable quality attributes (MR)

MR1: For invalid requests (e.g., null coordinates, out-of-range latitude/longitude, angle not a multiple of 22.5, non-closed polygon), endpoints shall return a client error response rather than producing undefined behaviour.

MR2: For valid inputs, distanceTo, isCloseTo, and nextPosition shall produce results to within 6 decimal places.

MR3: For typical requests, the API shall respond within a target response-time threshold (2000ms).

MR4: Given the same ILP inputs (drones, service points, restricted areas) and the same dispatch request payload, calcDeliveryPath/calcDeliveryPathAsGeoJson shall generate the same path outputs across repeated runs

MR5: Paths produced by the planner shall remain outside restricted areas by at least the distance equivalent to one movement step (0.00015 degrees).

MR6: Error responses shall not expose stack traces or internal class names to clients.

MR7: If the ILP service is unavailable or returns invalid data, the system shall fail gracefully with a clear 5xx/4xx response rather than timing out indefinitely or crashing.

Qualitative requirements (QR)

QR1: Domain logic (geometry validation, path finding, selection logic) shall be implemented outside the web controller layer so it can be tested independently of HTTP transport.

QR2: The system shall provide outputs in a format suitable for visualisation (coordinate pairs in a correct order), enabling integration with the frontend.

QR3: The base URL of the external ILP service shall be configurable at runtime via Spring configuration, allowing the system to run against different ILP endpoints without code changes.

1.b. Level of requirements, system, integration, unit

System-level requirements

FR7, FR9*, FR10*

MR3*, MR4*

Integration-level requirements

FR6*, FR8*, FR11

MR5*, MR7

QR1, QR2*

Unit-level requirements

FR1, FR2*, FR3, FR4*, FR5

MR1*, MR2*, MR6

QR3

1.c. Identifying test approach for chosen attributes

In 1.b above, the requirements marked with an asterisk (*) are the chosen attributes in this section for each level.

System-level chosen requirements (FR9, FR10, MR3, MR4)

FR9: calcDeliveryPath returns a plan with totals

System & acceptance testing: end-to-end tests using realistic dispatch scenarios (e.g., the visualiser's dispatch-scenario.json) and confirming the service provides the expected outputs.

Use-case–based testing: User submits a dispatch request -> system selects suitable drone -> system returns path + cost

Regression testing: Add these scenarios to an automated regression suite so changes don't break existing behaviour.

FR10: calcDeliveryPathAsGeoJson returns a valid GeoJSON suitable for visualisation

System testing + Interface-focused testing: Validate the GeoJSON structure and required fields for interoperability with the front-end.

Regression testing: Retest all, as changes to planners/serialisation may break existing behaviour.

MR3: Typical response time ≤ 2000ms

Non-functional testing: Performance testing (timing tests): repeated timed runs of key endpoints, reporting median for typical payload sizes.

MR4: Deterministic outputs for identical inputs

Regression testing: Use fixed inputs and assert output equality across runs.

System testing: same HTTP request -> same response body (or same key fields if minor ordering differences exist).

Integration-level chosen requirements and test approach (FR6, FR11, MR5)

FR6: ILP retrieval + mapping (drones, service points, restricted areas, drones-for-service-points)

Integration testing approaches: test ILPRestController + mappers first with mocked ILP responses, then integrate into DroneService.

Interface-focused testing: validate request/response assumptions between the system and the ILP API (field presence, array sizes, missing/empty fields).

FR11/MR5: Planner avoids restricted areas with a one-step buffer

Integration testing approaches - sandwich integration: test FlightPathAlgorithm with restricted areas in isolation, then integrate it into DroneService.calcDeliveryPath.

Interface-focused testing: verify restricted area geometry is correctly interpreted across DTO -> entity -> algorithm.

Use-case/scenario-based testing: straight line clear, straight line blocked by restricted area, no feasible route.

Fault-based - error guessing: likely faults include failing to buffer areas, failing intersection tests.

MR1: Invalid requests return client errors instead of undefined behaviour

Functional - decision table based testing: (valid/invalid coordinate × valid/invalid angle × valid/invalid region) -> expected 200 vs 4xx.

Boundary Value Analysis (BVA)/Equivalence Class Partitioning (ECP): nulls, NaNs, out-of-range values, malformed polygons.

Interface-focused testing at the API boundary: send invalid JSON and confirm consistent error behaviour.

Unit-level chosen requirements and test approach (FR2, FR4, FR5, MR1, MR2)

FR2/MR2: Distance calculation is correct (to 6 decimal places)

Functional - ECP: identical points, very close points, far-apart points, negative/positive coordinate mixes.

BVA: lat/lng at limits (±90, ±180), and values just inside the limits.

Structural (white-box): Statement + branch coverage over the distance implementation and validation paths (ensures all branches for invalid/valid coordinates are exercised).

Fault-based - error guessing: likely faults include swapped lat/lng, incorrect subtraction order, or wrong tolerance usage

FR4/MR2: nextPosition step and valid angle handling (multiples of 22.5)

Functional - ECP: valid angles (multiples of 22.5) vs invalid angles; valid coords vs invalid coords.

BVA: angles at 0 and 360, and just outside range (-1, 361); also check near coordinate bounds.

Decision table based testing: (angle validity × coordinate validity) -> expected outcome (Position returned vs client error).

Structural: Branch/condition coverage of the validation logic (ErrorHandler.angle, null/NaN checks).

Error guessing: typical faults are wrong step size, radians/degrees mix-up

FR5/MR1: isInRegion correctness and polygon validation

Functional - ECP: point inside/outside/on boundary; valid polygon/invalid polygon.

BVA: points on edges/vertices; minimum polygon size (4 vertices closing the polygon).

Structural: Branch coverage over validation rules (null checks, vertex count, closed polygon check).

Error guessing: off-by-one when iterating vertices, boundary handling mistakes, and incorrect closure check.

1.d. Assess the appropriateness of your chosen testing approach.

The chosen strategy is well matched to the requirements because it combines:

Specification-based unit tests for mathematical/validation correctness

Interface-focused integration tests for ILP dependency and planner pipeline correctness

Scenario-based system tests for end-to-end service delivery and GeoJSON interoperability,

The main weaknesses are dependency realism (mock ILP vs real ILP), difficulty proving geometric avoidance in all cases, and limited realism in performance testing. These limitations are acceptable for the portfolio as long as they are acknowledged and mitigated with clear documentation of test assumptions and environments.