## Requirements Overview:

**System Context:**

The system is a simulation-based drone dispatch and path-planning service operating over a constrained geographic area (Edinburgh). It does not control physical drones, therefore, requirements focus on logical safety, correctness, and performance properties within a simulated environment using synthetic data.

## Range of Requirements:

### Functional Requirements - Safety:

**Safety Requirement 1: No-Fly Zone Compliance**

**Priority:** Critical

**Stakeholders:** Edinburgh City Council, Civil Aviation Authorities, System Operators

**Description:** The system **shall not** generate flight paths that enter restricted airspace zones defined as no-fly zones:

- George Square Area (4-vertex polygon imported from the ILP REST API)
- Bristo Square Area (4-vertex polygon imported from the ILP REST API)

**Testing Approach:** Unit-level structural (white-box) testing of polygon intersection logic, integration testing of pathfinding with no-fly zone validation, system-level negative testing using restricted-area scenario

**Test Level:** System, Integration, Unit

**Safety Requirement 2: Corner Crossing Prevention**

**Priority:** Critical

**Stakeholders:** Safety Regulators, System Operators

**Description:**

Flight paths **shall not** cross corners (vertices) of no-fly zones or restricted areas.

**Detailed Specifications:**

- Any path segment that would pass through a polygon vertex is invalid
- Path planning must route around corners, not through them
- Applies to both entry and exit vertices of restricted polygons
- Minimum clearance from vertex: 0.00015° (one move distance)

**Testing Approach:** Unit testing with boundary value analysis around polygon vertices, integration testing to ensure path rerouting around corners, negative testing for vertex-crossing paths

**Test Level:** Unit, Integration

**Failure Mode:** Alternative path generation or dispatch rejection

**Safety Requirement 3: Capacity Constraint Enforcement**

**Priority:** Critical

**Stakeholders:** Drone Operators, Pharmacies, Safety Regulators

**Description:**

Drones **shall not** be assigned deliveries exceeding their maximum capacity in grams.

**Detailed Specifications:**

- Each drone has capability.capacity attribute (in grams)
- System must sum total weight of all items in assigned orders
- Total weight must be ≤ drone.capability.capacity
- Weight calculation must include packaging overhead (if specified)
- Batch assignments must validate cumulative weight before dispatch

**Testing Approach:** Unit testing using equivalence partitioning and boundary value analysis on weight calculations, integration testing of order assignment logic with cumulative weights

**Test Level:** Unit, Integration

**Test Cases:**

- Single order at max capacity (boundary value)
- Single order exceeding capacity (negative test)
- Multiple orders totaling max capacity
- Multiple orders exceeding capacity
- Empty order (0 grams)

**Failure Mode:** Order rejected with HTTP 400 and capacity violation message

**Safety Requirement 4: Maximum Flight Distance Enforcement**

**Priority:** High

**Stakeholders:** Drone Operators, Battery Management System

**Description:**

The total flight distance **shall not** exceed drone.capability.maxMoves for any dispatch.

**Detailed Specifications:**

- maxMoves is defined per drone in capability object

- Total path distance = (service point -> pickup) + (pickup -> dropoff1) + ... + (dropoffN -> service point)
- Each move has a distance of 0.00015°
- Path calculation must account for obstacle avoidance overhead
- System must reserve safety margin (suggested 5% of maxMoves)

**Testing Approach:** Unit testing of move-count calculations with boundary values, integration testing of path planning including obstacle overhead, system-level negative testing for over-limit paths

**Test Level:** System, Integration

**Test Cases:**

- Flight at exactly maxMoves (boundary value)
- Flight exceeding maxMoves (negative test)
- Multi-dropoff flight within maxMoves
- Single long-distance flight

**Safety Requirement 5: Return to Base Requirement**

**Priority:** High

**Stakeholders:** Drone Operators, Logistics Managers

**Description:**

All drones **shall** return to their originating service point after completing deliveries.

**Detailed Specifications:**

- Starting service point location from ServicePoint.location
- Final path coordinate must be within 0.00015° of service point location
- Return path must also respect no-fly zones
- Return distance must be included in maxMoves calculation
- Battery reserve must allow safe return

**Testing Approach:** System-level functional testing validating complete mission cycles, integration testing ensuring return paths respect no-fly zones and move limits

**Test Level:** System

**Test Cases:**

- Successful return to Appleton Tower
- Successful return to alternative service points
- Return path blocked by no-fly zone (fallback test)

**Failure Mode:** Path rejected if return not feasible within maxMoves

**Safety Requirement 6: Temperature-Controlled Item Safety**

**Priority:** High

**Stakeholders:** Pharmacies, Medical Authorities, Patients

**Description:**

Orders requiring the capabilities of cooling or heating **shall** only be assigned to drones with appropriate capabilities.

**Detailed Specifications:**

- Orders may specify cooling requirement (boolean)
- Orders may specify heating requirement (boolean)
- Drone capability.cooling must be true for cooling orders
- Drone capability.heating must be true for heating orders
- Mixed temperature requirements in batch are invalid

**Testing Approach:** Unit testing of capability-matching logic, integration testing of order assignment with compatible and incompatible temperature requirements, negative testing for mismatches

**Test Level:** Unit, Integration

**Test Cases:**

- Cooling order -> drone with cooling capability
- Cooling order -> drone without cooling (negative test)
- Heating order -> drone with heating capability
- Mixed batch with incompatible temperatures (negative test)

**Failure Mode:** Assignment rejection with capability mismatch error

## Functional Requirements - Correctness:

**CR1: Positioning Accuracy**

**Priority:** Critical

**Stakeholders:** Drone Operators

**Description:**

Flight paths **shall** terminate within 0.00015 degrees of target delivery coordinates.

**Detailed Specifications:**

- Precision requirement: +- (plus minus) 0.00015° (5 decimal places)
- Distance calculation through Pythagorean distance $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$
- Use a close enough test: check if drone distance from the delivery point is < 0.00015° (strictly less than)
- Applies to: pickup locations, drop-off locations, service point returns

**Testing Approach:** Unit testing using boundary value analysis on distance calculations, system-level functional testing of pickup, drop-off, and return accuracy

**Test Level:** Unit, System

**Test Cases:**

- Exact coordinate match (0.0° distance)
- Coordinate at 0.000149° distance (boundary - pass)
- Coordinate at 0.000150° distance (boundary - fail)
- Coordinate at 0.001° distance (clear fail)

**CR2: Order Validation Completeness**

**Priority:** High

**Stakeholders:** Pharmacies, Logistics Managers

**Description:**

System **shall** validate all order attributes inputted for each delivery before accepting an order for dispatch

**Detailed Specifications:**

Validation checks required:

1. Total weight of the order is ≤ available drone capacity
2. Delivery location is a valid LngLat coordinate
3. Pickup location matches restaurant location

4. Temperature requirements match drone capabilities
5. Order date/time within drone availability windows
6. Delivery location is reachable (not inside no-fly zone)

**Validation Sequence:**

1. Parse order JSON -> DTO object
2. Validate structure (required fields present)
3. Validate business rules (above checks)
4. Return validation result with specific error codes

**Testing Approach:** Unit testing of individual validation rules using equivalence partitioning, integration testing of full validation pipelines with systematic functional testing

**Test Level:** Unit, Integration

**Test Cases:**

- All validations pass
- Invalid order name
- Excessive weight
- Invalid coordinates
- Incompatible temperature
- Drone unavailable
- Unreachable location

**CR3: Cost Calculation Accuracy**

**Priority:** High

**Stakeholders:** Financial Controllers, Management

**Description:**

System **shall** accurately calculate total cost for each drone flight.

**Detailed Specifications:**

Cost formula: Total Cost = costInitial + (totalMoves × costPerMove) + costFinal

Where:

- costInitial: Fixed cost for takeoff (from drone capability)
- costPerMove: Variable cost per 0.00015° move (from drone capability)
- totalMoves: Actual path length in move units
- costFinal: Fixed cost for landing/return (from drone capability)

**Precision Requirements:**

- Currency precision: 2 decimal places (pence/cents)
- Intermediate calculations: Use BigDecimal to avoid floating-point errors
- Rounding: HALF_UP rounding mode

**Testing Approach:** Unit-level structural testing with fixed test oracles and boundary values, precision testing for rounding and decimal accuracy

**Test Level:** Unit

**Test Cases:**

Example:
costInitial = 3.40
costPerMove = 0.03
totalMoves = 1000
costFinal = 4.50
Expected: 3.40 + (1000 × 0.03) + 4.50 = £37.90

**CR4: Compass Direction Compliance**

**Priority:** Critical

**Description:**

Drone movements **shall** only use the 16 standard compass directions.

**Detailed Specifications:**

Allowed directions (22.5° intervals):

- Primary: N (90°), E (0°), S (270°), W (180°)
- Secondary: NE (45°), SE (315°), SW (225°), NW (135°)
- Tertiary: NNE (67.5°), ENE (22.5°), ESE (337.5°), SSE (292.5°), SSW (247.5°), WSW (202.5°), WNW (157.5°), NNW (112.5°)

Each direction has exact angle from East (0°):

E=0, ENE=22.5, NE=45, NNE=67.5, N=90, NNW=112.5, NW=135, WNW=157.5, W=180, WSW=202.5, SW=225, SSW=247.5, S=270, SSE=292.5, SE=315, ESE=337.5

**Testing Approach:** Unit-level structural testing with fixed test oracles and boundary values, precision testing for rounding and decimal accuracy

**Test Level:** Unit

**Test Cases:**

- Each of 16 directions produces correct change in longitude and latitude positions
- Invalid direction enum rejected
- Angle-to-direction conversion accuracy

**CR5: Move Distance Precision**

**Priority:** Critical

**Description:**

Each drone move **shall** be exactly 0.00015 degrees in length.

**Detailed Specifications:**

- Standard move distance: 0.00015° exactly
- Acceptable tolerance: +-$10^{-12}$ degrees (floating-point rounding)
- Applies to all 16 compass directions
- Hovering represented by two moves at same coordinate

**Testing Approach:** Unit testing using precise numerical assertions with tolerance thresholds, structural testing of movement calculation logic

**Test Level:** Unit

**CR6: Path Validity Complete Validation**

**Priority:** High

**Stakeholders:** All system users

**Description:**

Generated flight paths **shall** satisfy all geometric and constraint requirements.

**Detailed Specifications:**

A valid path must:

1. Start within 0.00015° of service point
2. Pass through pickup location within 0.00015°
3. Pass through all dropoff locations within 0.00015°
4. End within 0.00015° of service point (return)
5. Not intersect any no-fly zone polygons
6. Respect restricted zone altitude limits
7. Not cross polygon corners
8. Use only 16 compass directions
9. Have all moves exactly 0.00015° length
10. Total moves ≤ drone.maxMoves

**Testing Approach:** Integration and system testing using systematic functional testing, constraint-based validation with negative testing for individual rule violations

**Test Level:** Integration, System

**Test Cases:**

- Valid path through open space
- Path violating each constraint individually
- Path with multiple violations

**CR7: Multi-Order Batch Assignment**

**Priority:** High

**Stakeholders:** Operations, Efficiency Analysts

**Description:**

System **shall** correctly assign multiple compatible orders to single drone flights.

**Detailed Specifications:**

Batch compatibility criteria:

1. Same starting service point
2. Total weight ≤ drone capacity
3. All dropoffs reachable within maxMoves
4. All delivery requirements fulfilled by drones carrying out delivery

Batch optimisation objectives:

1. Minimise total moves across all flights
2. Minimise total cost across all flights
3. Maximise drone utilisation

**Testing Approach:** Integration testing of batch assignment logic using equivalence partitioning on compatibility constraints, system-level scenario-based testing for optimisation behaviour

**Test Level:** Integration, System

**Test Cases:**

- 2 compatible orders batched successfully
- 3 compatible orders batched successfully
- Incompatible temperature requirements separated
- Capacity-exceeding batch rejected
- MaxMoves-exceeding batch rejected

## Functional Requirements - Liveness

**LR1: Service Health and Readiness**

**Priority:** High

**Stakeholders:** Operations, DevOps

**Description:**
The system **shall** provide health check endpoints indicating service operational status.

**Detailed Specifications:**

- Health endpoint: GET /actuator/health returns {"status": "UP"}
- Ready endpoint indicates ILP REST API connectivity
- Startup time: < 30 seconds from container start to accepting requests

**Test Approach:** Integration and system testing using automated endpoint checks during startup and dependency availability testing

**Test Level:** Integration, System

**Test Cases:**

- Service starts successfully and health returns UP
- Health check fails when ILP API unreachable
- Ready probe succeeds only after dependencies available
- Restart completes within 30 seconds

**LR2: Deterministic Path Calculation**

**Priority:** High

**Stakeholders:** Testing Team, System Architects

**Description:**
Given identical inputs, pathfinding **shall** produce identical output paths.

**Detailed Specifications:**

- A* algorithm implementation is deterministic
- Tie-breaking in pathfinding uses consistent rules
- No random number generation in path calculation
- Same order + same drone + same restrictions leads to the same path
- Enables reproducible testing and debugging

**Test Approach:** Unit and integration testing using repeated execution with identical inputs; regression-style comparison of outputs for determinism

**Test Level:** Unit, Integration

**Test Cases:**

- Same inputs produce identical path
- Path hash matches expected value for known scenarios
- Different tie-breaking scenarios handled consistently

## Measurable Quality Attributes - Performance

**PR1: Pathfinding Response Time**

**Priority:** Critical

**Stakeholders:** Drone Operators

**Description:**
A* pathfinding algorithm SHALL complete within 2 seconds for typical Edinburgh routes.

**Detailed Specifications:**

- Simple path (no obstacles): < 200ms average
- Complex path (around George Square): < 2000ms
- Multi-dropoff optimization (2-3 locations): <5 000ms maximum

**Test Approach:** Unit-level performance testing using deterministic microbenchmarks; integration-level timing tests on representative synthetic workloads

**Test Level:** Unit, Integration

**Test Scenarios:**

1. Appleton Tower -> Informatics Forum (straight line): <200ms

2. Appleton Tower -> Business School (around George Square): <1500ms

3. Appleton Tower -> 3 Old Town locations (batch): <3000ms

**PR2: Order Validation Response Time**

**Priority:** High

**Stakeholders:** Pharmacies, Drone Operators

**Description:**

Order validation endpoint **shall** respond within 500ms for typical orders.

**Detailed Specifications:**

- Target response time: < 500ms
- Includes: menu validation, weight calculation, drone availability check
- Excludes: network latency to ILP REST API (measured separately)

**Test Approach:** Integration and system-level performance testing using timed request-response measurements under controlled load

**Test Level:** Integration, System

**PR3: Batch Optimisation Execution Time**

**Priority:** Medium

**Stakeholders:** Drone Operators

**Description:**
Multi-order batch optimisation **shall** complete within reasonable time based on batch size.

**Detailed Specifications:**

- 2-3 orders: < 3 seconds average
- 4-5 orders: < 8 seconds average
- 6-10 orders: < 20 seconds maximum
- Algorithm uses greedy heuristics for efficiency
- Early termination when acceptable solution found

**Test Approach:** Integration-level performance profiling with increasing batch sizes; execution time measurement under representative scenarios

**Test Level:** Integration

**PR4: ILP REST API Call Latency**

**Priority:** Medium
**Stakeholders:** Integration Team

**Description:**
ILP REST API calls SHALL have acceptable latency with proper timeout handling.

**Detailed Specifications:**

- Target latency: <500ms for successful responses
- Connection timeout: 2 seconds
- Read timeout: 5 seconds

**Test Approach:** Integration testing with mocked external services and injected latency; timeout and retry behaviour testing

**Test Level:** Integration

**Test Cases:**

- Successful call within 500ms
- Timeout after 5 seconds -> retry logic activates

**PR5: Memory Efficiency for Path Calculations**

**Priority:** Medium

**Description:**
Pathfinding algorithms **shall** operate within reasonable memory constraints.

**Detailed Specifications:**

- Single path calculation: < 50MB heap allocation
- Batch optimisation (5 orders): < 200MB heap allocation
- No memory leaks over sustained operation

**Test Approach:** Unit and integration-level resource testing using memory profiling and heap analysis during path computation

**Test Level:** Unit, Integration

## Measurable Quality Attributes - Reliability

**RR1: API Response Success Rate**

**Priority:** High

**Stakeholders:** API Consumers, Operations

**Description:**
Valid dispatch requests **shall** succeed with >99% success rate under normal conditions.

**Detailed Specifications:**

- Success = HTTP 200/201 with valid response payload
- Excludes client errors (HTTP 400, 422) from failure rate
- Includes server errors (HTTP 500, 503) in failure rate

**Test Approach:** Long-running integration and system testing measuring success rates across repeated valid request scenarios

**Test Level:** Integration, System

**RR2: Data Consistency for Order Processing**

**Priority:** Critical

**Stakeholders:** Data Team, Operations

**Description:**
Order processing **shall** maintain data consistency across all operations.

**Detailed Specifications:**

- Order weight calculation always matches sum of items
- Drone assignment is atomic (no double-assignment)
- Flight path waypoints match order pickup/dropoff coordinates
- Validation rules consistently enforced across all entry points

**Test Approach:** Integration testing with consistency assertions across order processing stages; transactional behaviour verification

**Test Level:** Integration

**RR3: Input Validation Completeness**

**Priority:** High

**Stakeholders:** Security, API Consumers

**Description:**
All invalid inputs **shall** be rejected with appropriate error messages.

**Detailed Specifications:**

- Errors to do with input data being sent to the API (null/empty required fields, misspelt fields etc.) -> HTTP 400
- Requests that are structurally sound but logically not (i.e: coordinates that are outside the range of valid coordinates) -> HTTP 200

**Test Approach:** Integration testing using systematic functional testing and negative testing for malformed and invalid inputs

**Test Level:** Integration

**Test Cases:**

- Missing name -> HTTP 400
- Longitude out of range (-5.0) -> HTTP 200

## Measurable Quality Attributes – Maintainability

**MR1: Code Modularity**

**Priority:** Medium
**Stakeholders:** Developers, Maintainers of the program

**Description:**
The system shall be structured into well-defined modules with clear responsibilities

**Detailed Specification:**

- Core subsystems (pathfinding, validation, dispatch, cost calculation) are separated into distinct packages
- No cyclic dependencies between packages
- Each class has a single primary responsibility

**Test Approach:**
Code inspection, dependency analysis tools

**Test Level:**
Unit (design-level), Integration

## Qualitative Quality Attributes - Usability

**UR1 – API Error Message Clarity**

**Priority:** High

**Stakeholders:** API Consumers, Operations

**Description:** API error responses shall provide clear, actionable error messages

**Detailed error specification:**

Error message guidelines:

- Use plain English, avoid technical jargon
- Include specific field names for validation errors
- Show both rejected value and expected constraint
- Suggest corrective actions where possible

**Test Approach:** Scenario-based integration testing with manual inspection against defined clarity and usability criteria

**Test Level:** Integration

**Test Cases:**

- Validation error includes field name and rejected value
- Capacity error message: "Total weight 3000g exceeds drone capacity 2500g. Try splitting order or selecting a larger drone"
- No-fly zone violation: "Flight path crosses restricted area: George Square. Choose different dropoff location"
- Missing required field: "Field 'customerName' is required but was not provided"
- Invalid coordinate range: "Longitude -5.0 is outside Edinburgh bounds (-4.0 to -3.0)"

**UR2 – Consistent Coordinate Format:**

**Priority:** Medium

**Stakeholders:** API Consumers, Frontend Developers

**Description:**
Geographic coordinates **shall** use consistent longitude and latitude ordering throughout the API implementation

**Detailed Specifications:**

Coordinate conventions:

- All coordinate pairs use [longitude, latitude] ordering
- Matches GeoJSON RFC 7946 standard
- Matches Leaflet.js mapping library convention

- Explicitly documented in API specification
- Schema includes min/max validation for Edinburgh bounds

Validation rules:

- Edinburgh longitude range: -4.0 to -3.0 degrees
- Edinburgh latitude range: 55.0 to 56.0 degrees
- Coordinates outside range rejected with HTTP 200
- Error message clarifies which coordinate is invalid

**Test Approach:** Integration testing using API contract tests; negative testing for reversed or out-of-range coordinates

**Test Level:** Integration

**Test Cases:**

- Valid Appleton Tower coordinates [-3.186874, 55.944494] accepted
- Reversed coordinates [55.944494, -3.186874] rejected with clear error
- Longitude out of range [-5.0, 55.944494] rejected: "Longitude -5.0 outside Edinburgh bounds"
- Latitude out of range [-3.186874, 57.0] rejected: "Latitude 57.0 outside Edinburgh bounds"

## Qualitative Quality Attributes – Robustness

**ROR1: Graceful Failure**

**Priority:** High
**Stakeholders:** API Consumers, Operations

**Description:**
The system shall fail gracefully when constraints cannot be satisfied.

**Detailed Specifications:**

- No uncaught exceptions exposed to API consumers
- Clear explanation of why dispatch failed
- System remains operational after failure

**Test Approach:**
Integration and system-level negative testing inducing constraint violations and failure conditions

**Test Level:**
Integration, System

**Robustness Requirement 2: External Dependency Isolation**

**Priority:** Medium

**Description:**
Failure of the ILP REST API shall not crash the system.

**Qualitative Criteria:**

- Timeouts handled cleanly
- Clear error returned to clients
- System continues to serve non-dependent endpoints

**Test Approach:**
Integration testing with fault injection and mocked external services to simulate dependency failure

**Test Level:**
Integration

## Qualitative Quality Attributes – Testability

**TR1: Deterministic Behaviour**

**Priority:** High

**Description:**
The system shall be deterministic to support repeatable testing.

**Detailed Specification:**

- No randomness in core algorithms
- Fixed tie-breaking rules
- Repeatable results for identical inputs

**Test Approach:**
Unit and integration testing using repeated execution and output comparison for identical inputs

**Test Level:**
Unit, Integration

**TR2: Observability**

**Priority:** Medium

**Description:**
The system shall expose sufficient internal state to support debugging and testing.

**Qualitative Criteria:**

- Path waypoints logged at debug level
- Validation decisions logged with rule identifiers
- Performance timings logged for critical operations

**Test Approach:**
Integration testing with log inspection to confirm availability of diagnostic and trace information

**Test Level:**
Integration