Requirements Supporting Document LO1

s2298559

December 2025

1 Functional Requirements

1.1 FR1 - Order Validation

The system shall validate each order retrieved from the `/orders` REST endpoint and classify it as `DELIVERED`, `VALID_BUT_NOT_DELIVERED`, or `INVALID` with an appropriate validation code. Validation includes checks on:

- Credit card number, expiry date, and CVV (the card number being 16 digits, expiry date being of MM/YYYY format and before today and CVV being 3 digits)
- Correct total price including delivery charge
- Pizza count between 1 and 4
- Pizzas belonging to a single restaurant
- Restaurant being open on the order date

This requirement is deterministic and rule-based, making it suitable for unit testing with controlled inputs and expected outputs. Boundary cases (e.g., exactly 4 pizzas, invalid totals by 1 penny) can be tested thoroughly.

1.2 FR2 - Flight-Path Calculation

For each valid order, the system shall compute a drone flight path from the restaurant to Appleton Tower using:

- Fixed-length moves of 0.00015 degrees
- One of 16 permitted compass directions (0.0 to 337.5 degrees in increments of 22.5 degrees)
- Mandatory hover actions at pickup and delivery points

This requirement depends on geometric calculations and movement constraints, which can be partially unit-tested (e.g., next-position calculation) but require integration testing to ensure consistency across components.

1.3 FR3 - No-Fly Zone Avoidance

The computed flight path shall never intersect any polygon defined as a no-fly zone. Other checks include:

- Verifying that the no-fly zone is an enclosed polygon
- Ensuring the destination and restaurant are not inside a no-fly zone, returning “No path found” if they are
- Ensuring that no-fly zones do not completely block access to the pickup or delivery point

Geometric intersection checks can be unit-tested, but verifying full paths against real no-fly zone data requires integration testing.

1.4 FR4 - Central Area Constraint

Once the drone enters the Central Area, it shall not leave until delivery is complete. The Central Area must be validated as an enclosed polygon.

This safety-related requirement depends on path history and therefore requires integration and system testing.

1.5 FR5 – REST Endpoint Behaviour

The service shall expose REST endpoints that return correct HTTP status codes and valid JSON/GeoJSON outputs.

This requirement is best verified using black-box system testing over HTTP.

2 Measurable Quality Attribute Requirements

2.1 QR1 - Performance (Runtime)

Flight-path calculations shall complete within approximately 60 seconds on target hardware.

This requirement necessitates performance testing under representative workloads.

2.2 QR2 – Numerical Accuracy

All coordinate calculations shall respect:

- Distance tolerance of 0.00015 degrees
- Floating-point precision constraints ($\pm 10^{-12}$ degrees)

Accuracy checks are well suited to unit testing with tolerance-based assertions.

2.3 QR3 - Robust Error Handling

The system shall never crash due to malformed requests or inconsistent data and must always return a defined HTTP response.

This is validated through system-level robustness testing and malformed-input generation.

3 Qualitative Requirements

3.1 NR1 - Reliability

The service shall behave consistently across repeated executions given identical input data. Reliability is assessed via repeated integration and system tests.

3.2 NR2 - Maintainability

The system shall be modular, enabling isolated testing of components (validation, geometry, routing). This supports a layered testing strategy.

3.3 NR3 - Data-Driven Design

All dynamic data (restaurants, no-fly zones, orders, Central Area) shall be retrieved from the external REST service.

Integration and system testing validate correct handling of dynamic data.

4 Level of Requirements and Corresponding Testing

Requirement Level	**Examples**	**Testing Strategy**
Unit	Order validation rules, distance calculations	Unit testing
Integration	Flight-path generation with zones and Central Area	Integration testing
System	REST endpoints, JSON/GeoJSON output, robustness	System testing
Quality Attributes	Runtime, numerical tolerance	Non-functional testing

Table 1: Level of requirements description

5 Justification of Test Approaches per Requirement Category

5.1 Deterministic Functional Requirements (FR1)

FR1 is deterministic and rule-based, making it ideal for:

- equivalence partitioning,
- boundary-value analysis,
- negative testing for malformed or contradictory inputs.

Unit tests provide high confidence, but system-level robustness testing is still required for malformed REST-layer inputs.

5.2 Geometric and Safety-Critical Requirements (FR2–FR4)

Path-related requirements involve continuous geometric spaces and interactions between multiple constraints. Appropriate techniques include:

- unit tests for geometric primitives,
- integration tests for full-path legality and interaction with real zone data,
- system tests for externally observable path validity.

Exhaustive geometric testing is infeasible; tests therefore provide probabilistic confidence.

5.3 Interface and Protocol Requirements (FR5)

FR5 concerns observable external behaviour. System-level black-box tests are the most appropriate:

- verifying status codes,
- validating JSON/GeoJSON schemas,
- testing robustness under malformed inputs.

6 Quality Attributes and Non-Functional Testing Rationale

6.1 Performance (QR1)

Runtime is an emergent property and can only be meaningfully tested at system level via repeated timed executions under representative workloads.

6.2 Numerical Accuracy (QR2)

Accuracy requirements suit unit tests with tolerance-based assertions. However, accumulated error over long paths remains a known limitation.

6.3 Robust Error Handling (QR3)

Robustness motivates negative testing, fuzzing, and malformed-input injection at system level.

7 Requirement-to-Test Traceability Matrix

Requirement	**Type**	**Level**	**Primary Testing Technique**
FR1 Order validation	Functional	Unit	Boundary + rule-based unit tests
FR2 Flight-path calculation	Functional	Unit/Integration	Numeric unit tests + path integration tests
FR3 No-fly zone avoidance	Safety	Integration	Polygon intersection + feasibility testing
FR4 Central Area constraint	Safety	Integration/System	Path-history constraint testing
FR5 REST behaviour	Interface	System	Black-box HTTP tests
QR1 Runtime	Performance	System	Timed regression tests
QR2 Numerical accuracy	Accuracy	Unit	Tolerance-based numeric tests
QR3 Robustness	Reliability	System	Malformed-input and negative testing
NR1 Reliability	Qualitative	System	Repeated execution consistency tests
NR2 Maintainability	Qualitative	Code-level	Review + modular test structure
NR3 Data-driven design	Qualitative	Integration	Dynamic REST data testing

Table 2: Requirement-to-test traceability mapping

8 Trade-offs, Limitations, and Testing Appropriateness

A layered testing strategy is the most appropriate compromise:

- unit tests provide precision for deterministic logic,

- integration tests validate emergent geometric behaviour,
- system tests validate correctness and robustness under realistic client interactions.

Alternative approaches (formal verification, large-scale statistical testing) could provide higher assurance but are impractical within resource constraints.