LO4 - Evaluate the limitations of a given testing process, using statistical methods where appropriate, and summarise outcomes:

4.1 – Identifying Gaps and Omissions in the Testing Process:

Logical Coverage and Condition Depth
The testing strategy achieved high statement and branch coverage in important classes such as DistanceService, RegionService, and RestrictedAreaService. This ensured that major decision paths were explored and that error-handling branches were reached during testing. However, branch coverage alone did not fully validate the correctness of complex logical expressions. In the route planning logic, multiple boolean conditions are combined to determine whether a path is valid, including coordinate range checks, no-fly zone intersections, and distance constraints. These conditions were mostly tested by checking the final outcome, rather than testing each logical condition separately. This means some true/false combinations of rules may not have been individually checked.

To address this limitation, condition level testing would be introduced so that each rule within a compound expression is tested separately in both its true and false states. This would provide stronger assurance that all constraint interactions behave correctly, particularly in edge cases where multiple rules conflict.

Test Data Diversity and Input Representation
Most test cases were based on manually generated inputs designed to target known error conditions, such as invalid orders or out of range coordinates. While effective for checking boundary cases, this approach limited the range of data being tested and reduced the chance of uncovering unexpected interactions between components. For example, scenarios combining orders with missing fields with otherwise valid geographic data were not systematically explored, leaving some request processing and validation paths undertested.

This gap could be reduced by automating test data. Generating structured variations of orders and coordinate sets would allow systematic coverage of high risk input combinations and better reflect the variety of patterns likely to appear in real world use when deployed.

External Data and Interface Reliability
The system verified that incoming data from external sources were of expected format and value ranges, but it did not evaluate how internal logic responded when external services behaved unreliably. Classes responsible for order processing and route calculation were not tested against delayed responses, inconsistent data, or partial service failures. As a result, the system's fault tolerance under realistic network and service conditions remains largely unverified.

This limitation could be addressed by introducing service simulation tools, such as mock endpoints, to reproduce conditions like timeouts, missing fields, and contradictory responses. Testing against these scenarios would strengthen confidence in the system's ability to recover from faults within external services.

Automation, Repeatability, and Process Limitations
Several robustness and exploratory tests were conducted manually, which supported targeted fault discovery but reduced consistency and repeatability across development iterations. Because these

tests were not part of an automated regression suite, it was difficult to measure defect detection rates or assess whether system quality was improving over time.

Expanding automated test coverage to include these robustness scenarios would allow regular re-execution and better tracking of test yield. This would support trend analysis across builds and provide more objective evidence of testing effectiveness as the system evolves.

4.2 – Identifying Target Coverage and Performance Levels:

Logical Coverage Targets
Objective:
Ensure that internal decision logic is fully tested at the level of individual conditions, not just final outcomes.

Target Level:

- Achieve 100% condition coverage for modules that contain complex Boolean logic, particularly in RestrictedAreaService, and DistanceService.
- Maintain at least 95% branch coverage across all core service classes being tested (RestrictedAreaService, DistanceService, RegionService)

Explanation:
Testing each logical condition as both true and false ensures that all rule combinations, such as coordinate limits, no-fly zone intersections, and distance constraints, are verified independently. This reduces the risk of hidden logic errors that may not be revealed through branch coverage alone.

Performance and Load Targets
Objective:
Confirm that the system performs reliably under both expected and heavy usage conditions.

Target Level:

- Process 95% of requests within 60 seconds under peak load scenarios.
- Sustain a test workload of at least 1,000 concurrent order requests without timeouts or system crashes.

Explanation:
These targets provide a clear benchmark for acceptable system performance and help identify scalability limits as demand increases.

Stress and Resilience Targets
Objective:
Define the system's behaviour at and beyond its operational limits.

Target Level:

- System should continue operating without crashing during double of the normal peak load.
- All failure cases should return meaningful error responses (e.g., HTTP 4xx/5xx) rather than invalid data or silent failures.

Explanation:
This ensures the system fails safely and predictably when overwhelmed or given extreme inputs.

Data Accuracy and Interface Validation Targets
Objective:
Ensure reliable handling of external and internal data sources.

Target Level:

- Validate 100% of incoming external responses against expected schemas and value ranges.
- Simulate fault conditions (delays, missing fields, incorrect values) in at least 30% of integration tests using mocked services.

Explanation:
These targets ensure the system can detect and handle unreliable data without passing errors to dependent components, improving reliability in real deployment environments.

4.3 – Comparison of Achieved Testing Levels Against Targets:

The testing process reached a strong level of assurance for both functional correctness and structural coverage across the system's core services. High unit and integration test coverage was achieved in critical components such as RestrictedAreaService, RegionService, and DistanceService, with method and line coverage reaching 100% and branch coverage exceeding 85% in all three. This confirms that most major execution paths and error-handling branches were exercised, particularly in geometric validation, distance calculations, and restricted area detection.

However, while branch coverage targets were largely met, the intended condition-level coverage was not fully achieved. Complex logical decisions within route validation, such as combined checks for compass direction accuracy, boundary tolerances, and no-fly zone intersections, were mainly verified through final path outcomes rather than by testing each logical condition independently. The system testing phase highlighted this limitation when all PathValiditySystemTest cases initially failed due to differences between test validation logic and production algorithms. This demonstrated that although paths were being generated correctly, deeper condition-level testing would have helped detect inconsistencies in rule interpretation earlier.

Performance testing met and exceeded baseline performance targets under controlled conditions. The PathfindingPerformanceTest showed that both simple and complex routes completed in under 2 milliseconds on average, far below the defined thresholds of 200ms and 2000ms. Scalability tests across multiple delivery counts showed linear performance trends. However, these results were obtained using synthetic, generated workloads and execution environments which were ideal and had no lag/latency, meaning the system's behaviour under sustained peak traffic or subpar network conditions are only partially validated.

Data validation targets were met at the format and range level, as inputs not matching the expected form and invalid coordinates were consistently rejected across unit, integration, and system tests. However, because external service failures were not simulated, the system was not fully tested against delayed, missing, or inconsistent data. This means that while the system works reliably with valid internal inputs, its behaviour in real-world conditions with unreliable external services is less certain.

4.4 – Actions Required to Achieve Target Testing Levels:

Strengthening Logical Validation
To reach the target coverage levels, the test suite should be extended from branch-level checks to full condition testing. This would involve designing tests that independently exercise each logical clause within key decision-making components, such as route constraint evaluation, boundary enforcement, and order validation. By isolating individual true and false conditions, the system's behaviour under complex and conflicting rules can be verified more reliably.

Expanding Performance and Scalability Testing
Achieving the desired performance targets would require a more structured load and stress testing framework. Automated workload generation should be introduced to simulate sustained high request rates, large delivery sets, and expanded restricted zone configurations. Monitoring response times, memory usage, and failure rates under these scenarios would help identify performance bottlenecks and help define the system's operational limits.

Improving External Service Resilience Testing
Testing of external dependencies should be enhanced using service mocking tools such as WireMock. By simulating delayed, incomplete, and inconsistent API responses, the system's ability to detect, handle, and recover from external services in components like order processing and route planning can be evaluated more thoroughly.

Increasing Automation and Test Yield Tracking
Automating more robustness and exploratory tests would make them easier to repeat after each code change. Running them as part of a continuous testing process would make it possible to track how many defects are found over time and show whether system quality is consistently improving.