Level of Requirements

This document will assign each class and method that is assessable in my ILP project to the appropriate level of testing for their requirements outlined in the **Range of Requirements** document. The levels include unit, integration, and system testing.

Further details of the individual tests can be found in the **Range of Requirements** document and evidence of the tests can be found in the **Test Cases** document.

Requirement ID from **Range of Requirements**	Test Level
FR-OV-01 – FR-OV-11	Integration
FR-N-01, FR-N-02	Unit
FR-N-03 – FR-N-06	Integration
FR-N-07, FR-N-08	System
MA-03	Unit
MA-01, MA-02	System

Unit Testing

Due to the tightly coupled nature of the controller logic, unit testing was intentionally limited to pure helper methods. Validation logic was tested at integration level where its behaviour is observable and meaningful.

Class	Requirements	Reference to **Test Cases** ID
LngLat	Represent a coordinate in terms of its longitude and latitude values. Calculate its Euclidean distance to another object of the same class. Able to check whether an object of this class is close to (less than 0.00015 degrees) another object of the same class.	UT-GEO-01 UT-GEO-02 UT-GEO-03

Rather than test the classes directly, my controller is structured so that the classes rely on individual helper methods which I can then perform unit testing on.

Method	Requirement	Reference to **Test Cases** ID
isValidCoordinate	Reject coordinates outside valid latitude/longitude bounds	UT-GEO-01
arePositionsClose	Return true when coordinate differences are less than 0.00015	UT-GEO-03

calculateEuclideanDistance	Return zero distance for identical coordinates	UT-GEO-02
calculateEuclideanDistance	Be symmetric with respect to argument order	UT-GEO-02

Integration Testing

Integration tests revealed several null handling and boundary condition defects in Order Validation (FR-OV-01 – FR-OV-11), including order date parsing errors that resulted in internal server errors. These issues were resolved by introducing defensive validation and verified through regression tests.

System Testing

Tests included:

- container startup and service availability checks (/isAlive)
- end-to-end order validation using realistic ILP sample data
- flight-path generation under operational constraints

Additional validation was required for generated flight paths as not only did the system need to return a flight path for a valid order, but the path also had to meet the following conditions:

- avoid all defined no-fly zones
- do not exit the central area after entry
- include required hover steps at delivery locations

GeoJSON outputs produced by /calcDeliveryPathAsGeoJson were manually inspected using **geojson.io** to provide visual confirmation of path legality and specification compliance.