2.1 Test Plan

This document outlines the test strategy and plan for the Pizza Drone Service, which includes validation of the order system, path calculation, performance, security, and robustness testing of the system.

Test Strategy

- A Test-Driven Development (TDD) is used, starting with unit tests for smaller components, followed by integration testing, system testing and performance testing. Security and robustness tests will be executed after initial implementation phases.

Test Objectives

- To ensure PizzaDronz performs as expected under normal and edge cases.
- To verify that the system meets all functional, performance, security, and robustness requirements.
- To ensure the system is adoptable to dynamic changes in operational environments (external API failures)

Test Scope

- Validating order details
- Pathfinding and GeoJSON conversion
- API and performance validation
- Input validation
- Robustness tests for handling edge cases

Test Lifecycle Fit (where testing fits in the development process)

- Functional Testing
    - Order Validation: Unit testing → Integration testing → System testing
    - Flight Path Calculation: Unit testing → Performance testing → System testing
    - API Endpoints: Unit testing → Integration testing → Security testing → System testing
- Performance Testing
    - Following unit and integration testing, ensuring the system responds under 500ms and flight paths are computed within 200ms.
- Security Testing: After system and integration testing, focus on preventing vulnerabilities like input validation and improper data handling
- Robustness Testing: Performed at the system level to handle invalid GPS coordinates, failures in API interactions, and resilience against no-fly zone changes.

Test Methodology

- Unit Testing
    - Validate individual functions and components (order validation, pathfinding algorithm)
- Integration testing
    - Validate interactions between components like the API and external systems
- System Testing
    - Validate the complete workflow, including order submission, payment processing, and drone delivery path calculation
- Performance Testing
    - Test the system under normal load conditions to ensure performance targets are met (response time under 500ms)

- Robustness Testing
    - Ensure the system can handle invalid GPS coordinates and API failures

Test Criteria

- Functional requirements
    - System correctly validates credit card details, order size, restaurant hours, and menu
    - Pathfinding algorithm calculates an accurate and efficient route.
- Performance requirements
    - API response time within 500ms under normal load
    - Pathfinding algorithm completes in under 200ms
    - System can handle at least 10 concurrent orders
- Security Requirements
    - No sensitive data is logged or stored
    - All endpoints reject improperly formatted requests
- Robustness Requirements
    - System rejects invalid GPS coordinates
    - An order is either fully processed or fully rejected
    - Handles external API failures

Tools

- Junit for unit testing
- Junit, Postman, SpringBoot Test Framework for API testing
- Mockito for mocking APIs for external restaurant data and no-fly zones

Test Cases

Test Case	Requirement	Description	Input	Expected Outcome
TC001	Order Validation (valid)	Verify that a valid order returns a valid validation result	Valid date following all the validation rules	Responses should return a 200 OK status code and the result of the order validation.
TC002 TC003 TC004	Order validation (Credit card details)	Validate credit card number and CVV are valid and full, and card is not expired	Order with invalid credit card number, invalid CVV, and expired credit card	Return the appropriate error code and a 400 BAD REQUEST status code
TC005	Order Validation (Pizza limits)	Validate that the number of pizzas is not more than 4	Order data with invalid pizza count.	Return MAX_PIZZA_COUNT_EXCEEDED and 400 BAD REQUEST.
TC006	Order Validation (Restaurant open)	Validate the restaurant is opened on the day of the order date	Order ordering from a restaurant which is not opened.	Return RESTAURNT_CLOSED and 400 BAD REQUEST
TC007 TC008	Order Validation (Restaurant)	Validate all pizzas on the order are from the same restaurant and on the restaurant's menu	Order with pizzas from multiple restaurant, and order with a non-existent pizza name.	Return PIZZAS_FROM_MULTIPLE_RESTAURANTS if from multiple restaurants, or PIZZA_NOT_DEFINED if not on the restaurant's menu, and 400 BAD REQUEST.

TC009 TC010	Order Validation (Price)	Verify individual pizza prices are correct, and the total price is sum of pizza prices plus a 100 pence delivery charge	Order data with invalid pizza price, and order data with invalid total price.	Return PRICE_FOR_PIZZA_INVALID if any of the pizza(s) price is incorrect, or TOTAL_INCORRECT if total price is incorrect, and 400 BAD REQUEST
TC011	Order Validation (Empty order)	Validate order is not empty	Empty order	Return EMPTY_ORDER and 400
TC012	Flight Path (Valid order)	Validate path is calculated correctly for a valid order	Order with valid data	Responses should return an array of valid Lng/Lat coordinates, and 200 OK.
TC013	Flight Path (No-Fly zones)	Verify flight paths do not cross no-fly zones	Valid order	Paths avoid no fly zones
TC014	Flight Path (Central Area)	Check the flight path does not deviate from the central area after entering	Valid order	Flight path does not leave central area once entered.
TC015	Flight Path (GeoJSON)	Verify that the flight path is returned as a valid GeoJSON	Valid order	Response should return GeoJSON formatted output for path, and 200 OK.
TC016	REST Endpoint for Flight Paths	Test that the flight path retrieval endpoint returns correct data	JSON output includes all required fields with valid data	
TC017	REST API Performance	Measure API response time for flight path calculation	Valid order	Each API request responds within 500ms
TC018	Pathfinding Algorithm Speed	Ensure paths are computed within 200ms.	Valid order	Flight path computation time is less than 200ms
TC019	Input Validation	Check that system rejects improperly formatted order requests	Orders with improper formats	400 BAD REQUEST
TC020	Input Validation	Check that system rejects invalid GPS coordinates	Orders with extreme latitude and longitude values	400 BAD REQUEST

Target Code Coverage: 85% - 90%