# PizzaDronz specification Document

## 1. Introduction

### 1.1 Purpose of Document

This is the specification document of a new PizzaDronz system for the School of Informatics. The school of Informatics is a department under the College of Science and Engineering at the University of Edinburgh. The new system will allow students to order pizzas via an app and have them delivered directly via drone to the top of Appleton Tower for collection. Students are then able to collect the pizzas and enjoy their pizzas at any time of the day. This document describes the system stakeholders and their respective main requirements and covers the functional requirements and non-functional requirements of the system. The document also briefs through the test approach for each requirement category and an assessment of its appropriateness and limitations.

### 1.2 System Stakeholders and Requirements

- School of informatics
    - The system generates a legal flight path to deliver pizza orders every day.
    - Flight paths generated by the system can maximize the number of orders delivered each day.
- Customers
    - System can process secure and safe transactions during order payment
    - The system can process orders rapidly to ensure the orders are delivered promptly
    - The system should be reliable and accessible throughout the day, for the use of customers at any time.
- Restaurant
    - The system should process orders in a reasonably short time so that restaurants can serve up orders as soon as possible.
- General public
    - The drone should fly safely through Edinburgh when delivering orders and avoid populated areas.
    - The drone should avoid residential areas when delivering orders to respect the privacy of the residents.
- Regulators
    - The drone should fly safely through Edinburgh when delivering orders by avoiding no-fly zones and adhering to drone flight laws.
- Software developers
    - The system should be easily maintainable and scalable in the long term.

## 2. Functional Requirements

1. The system shall generate a flight path for a singular drone to deliver a pizza order.
   **[high priority system level requirement]**
2. The system shall not deliver more than one order at a time to avoid wrong deliveries. For example, a non-vegetarian pizza to a vegetarian.
   **[high priority integration level requirement]**

3. The system shall only deliver orders when it has a sufficient battery for the day. In this case, the drone can make at most 2000 moves before it runs out of battery.
   **[high priority integration level requirement]**
4. The system shall validate all orders received before generating a flight path for the drone. This must include the following checks:
   **[high priority integration level requirement]**
   a. Each order must contain between 1 to 4 pizzas.
   b. Each order must only contain pizzas ordered from the same restaurant. Orders containing pizzas from more than 1 restaurant will be invalidated.
   c. Each order must contain pizzas from the menu of the given list of restaurants.
   d. Each order must contain valid transactional details. The number, expiry date and CVV of the credit card must all be validated.
   e. Each order must contain the correct stated total cost. This includes the total price of all pizzas ordered and the delivery fee of £1.
      **[high priority unit level requirements]**
5. The system shall fetch all data required by the drone from a REST API, which stores all orders made by users each day. This also includes the no-fly zones, central area, participating restaurants and their respective menus. The system shall also verify the REST API whenever it fetches data from the API. This includes validating the URL to the REST API, the server itself, and the data fetched.
   **[high priority integration level requirement]**
6. The system shall return JSON files containing the following daily results:
   a. The outcome of all orders in a day, recording the deliveries and non-deliveries made.
   b. The flight path of the drone, recording each move taken by the drone.
   **[high priority integration level requirement]**
7. The system shall generate flight paths such that the number of pizza orders delivered each day is maximized.
   **[high priority system level requirement]**
8. The system shall avoid generating flight paths where the drone leaves the central area of the University of Edinburgh. If the drone does leave the central area, it should only do so once in a single flight and should return to the central area in the shortest route possible.
   **[high priority integration level requirement]**

# 3. Non-Functional Requirements

## 3.1 Performance Requirements

- The mean time for the system to generate flight paths for all orders in a day and produce the resulting files in JSON format shall not exceed 60 seconds.
  **[medium priority system level requirement]**
- The system should be able to support 50 simultaneous users.
  **[medium priority system level requirement]**

## 3.2 Quality Attributes

- **Availability**
  - The system should be available 24 hours a day but must be available from 11 am to 2 pm each day as that is the peak lunchtime for students.
    **[medium priority system level requirement]**

- **Security**
    - The system shall ensure transactional data used in payments must be transmitted in encrypted form.
      **[high priority system level requirement]**
- **Reliability**
    - The system shall be completely operational for a minimum of 95% of the time.
      **[medium priority system level requirement]**
- **Maintainability**
    - The mean time between failure following a system failure must not be greater than 30 minutes. This includes all corrective maintenance time and delay time.
      **[medium priority system level requirement]**

## 3.3 Qualitative Requirements

- The system should have an application interface that is neat and user-friendly. It should be simple and quick for users to learn how to navigate the system.
  **[medium priority system level requirement]**
- The system should support most devices running across the latest operating system versions.
  **[medium priority system level requirement]**
- The system shall ensure that all user data are protected.
  **[high priority system level requirement]**
- The system should always ensure the safety and privacy of all stakeholders when operating.
  **[high priority system level requirement]**

# 4. Testing Approach

## 4.1 Functional Requirements

There is a mixture of system and integration level requirements in the points listed in section 2. Thus, to ensure the testing analysis is complete, unit level tests will be used throughout the development of the system to ensure the correctness of each module's component. Integration level tests will also be used during the integration phase to verify if each module integrates properly. Additionally, system level tests will be essential for validating that the system produces an optimum flight path to deliver orders to users. Analysis and testing principles from chapter 3 largely influence the testing approach is taken: The higher level test will be partitioned into smaller simple components for easier testing, static type checking and asserting properties in the code are redundant but will help to increase confidence and the specifications will be limited to reduce difficult problems into simpler ones. Additionally, code coverage will also be used as a metric to evaluate the completeness of the test analysis, ensuring all areas of the source code are executed during tests. Lastly, scaffolding and print statements will be used throughout the development to simplify the testing process.

## 4.2 Non-Functional Requirements

There is a variety of system level requirements covered in section 3. These will be tested via black box testing and non-functional system level testing for verification after the system has been completed. For example, the response time of the system will be measured to check the performance of the system.

# 5. Testing Approach Assessment

## 5.1 Appropriateness

Unit tests are primarily used to isolate small components of code and check if it works as expected. It is necessary as it helps detect bugs early in the development of the system, makes it simplifies refactoring code, and thus makes integration safer and easier. It also eases the debugging process and ensures the code written has high cohesion since it encourages modular programming. However, unit tests are costly as it is time-consuming and is not guaranteed to catch all bugs.

Integration tests mainly check the interfacing between modules. It helps catch integration problems between connected modules, ensures each module works properly before developing the complete system and improves code coverage, giving the code another level of reliability. On the downside, integration tests could be difficult to incorporate and expensive to do so, as they often require much scaffolding and can end up being time-consuming.

System tests are a vital part to verify the complete system. It is relatively simple to execute since in-depth knowledge of the system is not required. It also tests the entire system and captures any bugs missed during the unit and integration testing. Thus, system tests ensure the end product has covered almost all bugs and increases confidence in the resulting system. However, the testing process is the most expansive and time-consuming, as it covers the testing of the entire system.

## 5.2 Limitations

Unfortunately, the scope of the project and access to resources are very limited. Thus, many essential tests cannot be executed. If there were no restrictions on resources, further test approaches would be executed. For example, stress tests to check reliability, load tests to further evaluate system performance, penetration testing and vulnerability scans to assess security, etc.

Additionally, if there were access to real user feedback, acceptance testing would be conducted to validate the system developed. This would include alpha and beta testing.