Functional requirements:
Based on the level of requirements: system(s), integration(I), and unit test(U).

System:

1. The system shall take 3 arguments: the base URL and the keyword with a specific date of the order.
2. The REST service should retrieve the following information:
    - Order information. (U)
    - Restaurant information. (U)
    - None-fly zone location. (U)
    - Central area location. (U)
3. A valid order must meet the following:
    - Correct card numbers with 16 numbers. (U)
    - Order numbers with a mixture of 8 letters and numbers uniquely. (U)
    - Valid expiration date of card that is in the same order date or after the order date. (U)
    - Valid CVV with 3 digits. (U)
    - Each order is between 1 to 4 pizzas. (U)
    - Check if the ordered pizzas are in the same restaurant. (U)
    - Check the pizza names match the names on the menu. (U)

- Check that the total price is added correctly, equal to the pizza price plus a one-pound delivery fee. (U)
- Check the date the restaurant is opening. (U)

4. 3 files needed to be created:
    - The first file records both the deliveries and non-deliveries made by the drone.
    - The second file records the flight path of the drone move-by-move.
    - The third file is the drone's flight path in GeoJSON format.

Correctness:

Success: the drone is consistent and meets the specifications.

- The drone will start delivering at Appleton Tower(U).
- Once the order is sent the drone will go back to Appleton Tower. (U)
- The drone can only fly in 16 compass directions. These are the primary directions of North, South, East and West, and the secondary directions between those of North East, North West, South East and South West, and the tertiary directions between those of North North East, East North East, and so forth. (U)
- The drone needs to avoid flying into the none-fly zone. (I)
- The drone has unique order numbers. (U)
- When hovering for delivery, the drone must be close to the destination

- Once the drone has entered the Central Area, it cannot leave it again until it has delivered the ordered pizzas to Appleton Tower. (I)
- The distance that the drone moves is 0.0015 degrees.
- The angle at which the drone hovers is 999.
- The delivery cost for the order fee is £1.
- Once an order with a valid status is shown the drone should find an optimal path.
- After the drone sends the delivery, the order status should be [delivered].
- While entering invalid order details, the order status should be [invalid].

Safety:

Prevent hazards: The drone completes the delivery successfully without crashes.

- The drone should be flying over the roofs of the buildings.
- Drone can only send one delivery at a time with a maximum of 4 pizzas.
- Drones must not enter the no-fly zone.
- Drones should not crash during flying.

Non-functional requirements:

Accessibility and useability:

- The system is designed for the students of the University of Edinburgh and should be easy to use.
  Performance:

- the URL won't cause any system error such as a 404 error.
- The whole program executes and generates 3 files within 60 seconds.
- The memory for storing files should be low.

Robustness:

- The application must be capable of handling any runtime errors gracefully.

Security:

- All the users' details must be stored safely.
- The drone must avoid populated areas since some students won’t like their photos to be captured or minimise the incident of the drone falling into the populated areas.

Efficiency:

- Since there is only one drone for delivery, we need to ensure it chooses the optimal path every time.

Extensibility:

- Prepare for future growth, such as adding more drones or expanding distribution areas. This involves creating tests and documentation that help with scalability and ease of modification.

Level of requirements:

In the previous section, all functional requirements are labelled with specific symbols - S, I or U - which represent system-level, integration-level and unit-level requirements. Of these, system-level requirements are the most macro, describing the functionality that an application should have. System-level requirements often have several conditions that they fulfil. These conditions are usually categorised as unit-level and sometimes integration-level requirements. This depends on the nature of

these conditions: if they specify a specific functional requirement, then they are considered unit-level; if they cover multiple unit-level requirements, then they are considered integration-level. All non-functional requirements are system-level requirements, as they are only valid for the complete application.