PizzaDronz is a pizza delivery service that takes in orders from students and utilises the drone that takes off from the rooftop of the Appleton Tower, picks up and delivers orders sequentially from a variety of restaurants across Edinburgh while avoiding the noFlyZones to bring them back to the tower. The service is supposed to be fast and reliable to handle a large number of orders in a timely fashion.
The following functional and non-functional requirements can be derived from the high-level specification:

Application Requirements:

- ## Non-Functional Requirements
    1. Provide good customer service
        1. Deliver pizzas in the most efficient manner.
        2. Log error messages verbosely for users to understand.
    2. Be robust and performant
        1. Performance (execution time under 60 seconds).
        2. In case of an error, exit the main application loop and inform the user about the error.
- ## Functional Requirements:
    1. The application shall take the date and the API URL as input.
    2. Fetch from the API:
        1. Orders.
        2. NoFlyZones.
        3. Restaurants.
        4. The Central Area.
    3. The application shall validate orders based on criteria:
        1. The order must have a unique identified assigned.
        2. Order is valid only if it's placed on the day.
        3. The order must have 1 to 4 pizzas inclusively.
        4. All pizzas have to come from the same restaurant and be present on the restaurant's menu.
        5. The restaurant has to be open on the day of the order.
        6. The total cost of the order must be equal to the sum of all pizza prices plus the delivery fee of £1.
        7. The order must have valid payment details:
            1. Card expiry date month and year must be no earlier than these of the order.
            2. The card number must be a 16-digit numeric value.
            3. The CVV of the card must be a 3-digit numeric value.
    4. The application shall ignore delivered or invalid orders after validating them as such.
    5. Every valid order shall be delivered in the exact order it was received.
    6. The application shall compute the shortest flight path for a delivery drone once the order was deemed "VALID_BUT_NOT_DELIVERED".
        - The flight path is a sequence of moves between adjacent coordinates that must take into account NoFlyZones provided by the API.
        - The flight is always to or from the top of Appletion Tower with fixed coordinates.
        - The destination is considered reached when the drone is 'close' to the point (within 0.00015 degrees)

- The drone must hover for 1 move when picking up or delivering the order.
- The drone must not enter a NoFlyZone at any point during the flight.
  Drone description:

1. A move is of 2 types:
    1. Hover
    2. Fly in a straight line for 0.00015 degrees in one of the 16 radially equidistant directions

7. After processing ordering and calculating flight paths the application shall create 3 files with the following output:
    1. A file that records both deliveries and non-deliveries made by the drone. Named `deliveries-YYYY-MM-DD.json` for a given day.
    2. A file that records the flight path of the drone move-by-move. Named `fligtpath-YYYY-MM-DD.json` for a given day.
    3. A file that records the flightpath in GeoJSON format. Named `drone-YYYY-MM-DD.json` for a given day.

The 3 levels of the bottom-up approach shall be used when devising a test-suite and should well-suited to the specification of requirements derived from the project behaviour description. Unit level tests will ensure that the standalone modules are working reliably and cover all edge cases minimizing or eliminating the possibility of unexpected behaviour. Integration tests will build on top of these units to make sure the interaction behaves as intended and produces desired results.
Qualitative requirements, such as robustness and performance will be quantified and assessed by introducing measurable benchmarks, such as execution time, relative test code coverage and, further in production, collected statistics about Mean-Time-To-Failure (MTTF).
Moreover, using this approach, should the system or integration requirements change or evolve, additional functional requirements and respective unit tests covering them may be easily introduced and integrated.

Some potential issues include unit-testing the external API, which is outside of our control, and therefore it's behaviour and responses cannot be guaranteed to stay the same even with testing on our end, impeding on robustness and reliability of the app.