***ILP Project requirements***

This is the official Software Requirements Specification (SRS) for the PizzaDronz app (ILP project for the current year). This document covers the following aspects related to the app: the ***stakeholders*** involved, ***functional requirements***, ***measurable attributes***, as well as ***clarifying/ refining specifications***.

# **Stakeholders**

In this section, we outline all the relevant stakeholders for the PizzaDronz app, along with reasoning as to why these stakeholders are relevant in the context of the project. We will define the notion of a stakeholder to be that of someone who is directly impacted by our PizzaDronz app. The list of relevant stakeholders is as follows:

1. **The university students:** The students are the ones placing delivery orders to different restaurants and receive their ordered pizzas via our drone. The students are a relevant stakeholder because they are the ***end users*** for the PizzaDronz app: they constitute the customer that the whole system is being designed for.
2. **The pizza providers:** these represent all the restaurants that will be available on the PizzaDronz app. They share their menus and allow orders to be placed, collected by our drone, and delivered to Appleton Tower for the relevant customers. The pizza providers are a relevant stakeholder because they rely on our drone delivery service in order to make more pizza sales.
3. **The drone manufacturers:** these represent a third-party company that will create the drone. They are a relevant stakeholder since the drone is the crucial component to our PizzaDronz app, as it will be integrated with the delivery-making algorithm in order to deliver as many orders as possible in a single day.
4. **The software designers:** this is the person responsible for delivering the end-product (in this case, me). The reason why software designers are a relevant stakeholder is because they are directly involved in implementing all the necessary software to make the PizzaDronz app fully functional.
5. **Staff of the School of Informatics:** these are mainly staff in higher positions in the School of Informatics, and are meant to serve as regulators of the PizzaDronz app. The reason they are a relevant stakeholder is because they are directly involved in overseeing how the PizzaDronz app is performing once it has been released to the end users.
6. **Other members of society:** this group essentially constitutes most of the population currently living in Edinburgh. The reason they are a relevant stakeholder is that there are certain safety and privacy issues that need to be considered when designing our system (examples of which will be given later in the document) and these issues directly impact this group.
7. **The government:** this represents the Scottish government. The main reason they are a relevant stakeholder is because they act as a regulating body over the School of Informatics, and they try to mitigate the risks our drone poses to the other members of society.

**Functional Requirements**

In this section, we outline all the functional requirements for the PizzaDronz app, along with any additional details, such as the category they pertain to. We will define the notion of a functional requirement as “what the system should do”. The list of functional requirements can be seen below:

1. **The system should be able to store locations on the map (such as restaurants, Appleton Tower, etc.) as a pair of (longitude, latitude) co-ordinates**, where the **longitude** is the measurement east or west of the prime meridian and the **latitude** is the measurement of distance north or south of the Equator. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
2. **The system should be able to compute the distance between any two given locations using the Pythagorean distance formula**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
3. **The system should not allow any point which is further away than the *distance tolerance* of 0.00015 degrees to be considered close**. This requirement falls under ***safety***, since it would be very problematic if the system fails to ensure this: if a point that isn’t meant to be labelled as close is in fact labelled as close, then this could severely impact the path finding algorithm and either result in the drone dropping off the pizzas too early, not getting close enough to collect the pizzas, making illegal moves, etc.
4. **The system should not allow the drone to make more than a total of 2000 moves**. This requirement falls under ***safety***, since it would be very problematic if the system fails to ensure this: if the drone tries to make more than 2000 moves, it will run out of battery and fall from the sky, risking potentially injuring unsuspecting civilians.
5. **The system should only allow the drone to make two types of moves: either *fly* or *hover***. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
6. **The system should record every *fly* move the drone makes as a straight line of length 0.00015 degrees**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
7. **The system should only allow the drone to fly in one of the 16 directions specified in the coursework specification document**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
8. **The system should impose that the drone makes a hover move when collecting pizza orders and dropping them off**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
9. **The system should ensure that the drone starts from Appleton Tower each day**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
10. **The system should ensure that the drone returns to a location close to Appleton Tower before it runs out of battery**. This requirement falls under ***safety***, since it would be very problematic if the system fails to ensure this: if the drone runs out of battery before reaching a point close to Appleton Tower, it will fall from the sky, risking potentially injuring unsuspecting civilians. By ensuring this happens close to Appleton Tower, we provide a safe location where the risk of injuring civilians is mitigated.
11. **The system should ensure that each move made by the drone consumes one unit of battery**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
12. **The system should ensure that, once the drone enters the University’s Central Area, it does not leave this area until the current order is delivered**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
13. **The system should ensure that the drone never enters a no-fly-zone**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
14. **The system should be able to retrieve data about different components (such as location of the University’s Central Area, locations of No-Fly-Zones, restaurant locations and order information) from a given REST server**. This requirement falls under ***liveness***, since it refers to the fact that the system eventually gets all the required information after reading in the URL for the REST server.
15. **The system should allow the user to input exactly 3 command line arguments, which are to be validated before the program runs**. These 3 arguments are: a URL (to server as an address for the REST server), a date (to be used for retrieving all the order information for the day), and a random seed generator. This requirement falls under ***safety***, since failing to validate the input could result in retrieving incorrect information, causing severe issues (the drone making illegal moves, going to the wrong locations, incorrect order information being read, etc.).
16. **The system should store the 16 possible compass locations that the drone can move in**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
17. **The system should store the 10 possible outcomes for a given order**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
18. **The system should validate each order for the given date, and, in case some of the order information is not valid, label it with the appropriate order outcome depending on which information was invalid**. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
19. **The system should only consider orders labelled as valid when planning a route for the drone**. This requirement falls under ***safety***, since allowing invalid orders to be processed by our system as valid could have severe consequences, such as restaurants not getting paid due to invalid card details or customers paying more or less than necessary due to invalid total calculations to name a few.
20. **The system should output user friendly messages in case a validation check for the command line input fails**. In this case, we will consider user friendly to mean a message that conveys why the error has occurred in a very clear, easy to understand manner. This requirement falls under ***liveness***, since it will only occur in case of a failed validation check, and not otherwise.
21. **After the drone has done all the possible deliveries for a day, the system should produce 3 output files for that date.** These files include two JSON files, one storing order information, and the other information about the flight path, and one GeoJSON file storing additional information about the flight path. This requirement falls under ***correctness***, since it is strictly related to how the system should operate.
22. **The system should be able to plan and plot a flightpath for delivering orders in 60 seconds or less**. This requirement falls under ***safety***, since we would consider the failure of meeting the 60 seconds timer as problematic for the PizzaDronz service.
23. **The drone should aim to deliver as many orders as possible in a given day.** This requirement falls under ***correctness***, since it is strictly related to how the system should operate.

**Measurable attributes**

In this section, we outline all the measurable attributes for the PizzaDronz app, along with any additional details needed for further clarity. The list of measurable attributes can be seen below:

1. **Performance**: the performance of the system can be measured as the average number of pizza orders delivered in a single day.
2. **Fairness:** the fairness of the system can be measured as whether, for each individual day, at least a certain number of orders from every restaurant on the app is being delivered (i.e., there is not one restaurant from where not enough orders are being delivered).
3. **Efficiency:** the efficiency of the system can be measured as how quickly the flightpath for the drone is planned and the output files are being produced.
4. **Usability:** the operability of the system can be measured as how easy it is for the end users to work with our product.
5. **Attractiveness:** the attractiveness of the system can be measured as how popular/ appreciated the system is among the end users.
6. **Reliability:** the reliability of the system can be measured as the number of faults that occur in a given period of time (e.g., a day).
7. **Recoverability:** the recoverability of the system can be measured as the time taken to resume functionality once a failure has occurred.
8. **Maintainability:** the maintainability of the system can be measured by the ease with which new features can be added (the time taken to get the system back up to full functionality once a change is made).
9. **Functionality:** the functionality of the system can be measured as how well the system has met all the necessary requirements.

**Clarifying/Refining Specifications**

In this section, we address some of the ambiguities that may prevent someone from verifying the specification, as well as any additional details. These comments can be found below:

* In order to verify the specification, one must be aware that the system will be developed in Java. Therefore, in order to run the app on a local device, one must ensure that their device has a Java Virtual Machine (JVM) installed in order to launch the PizzaDronz app.
* In addition, one must also know that the measurable attributes missing from the specification, as well as the altered definitions, are related to the stage of development that the product is meant to be in. The final product is not meant to be released to the users, but rather be passed as a beta version, with the user interface and other features being implemented by the new developers taking over the project once fully implemented.
* Lastly, one must also know that the PizzaDronz app is a command line app and should therefore be launched from the command line (using the device’s Terminal).