S1973227

ILP Coursework 2- Implementation Report

Software Architecture Description

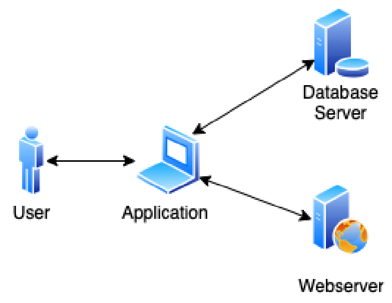
Introduction
I have been tasked by the school of informatics to devise and implement an algorithm to control the flight of the drone as it makes its deliveries while respecting the constraints on drone movement specified

Purpose
This document provides an architectural description of the Informatics Large Practical Coursework 2 – Drone Delivery Service. This is used to convey architectural decisions that have been made about the system.

Scope
This document provides an architectural overview of the ILP drone project developed by James Keane

Non-Functional Requirements
1. High Performance – The Runtime of the application should be less than 60 seconds, with an average of 10-20 seconds
2. Error Management – The application should be able to handle all errors, exiting the application when appropriate
3. Secure – When possible private functions should be used in the application

Architecture Summary



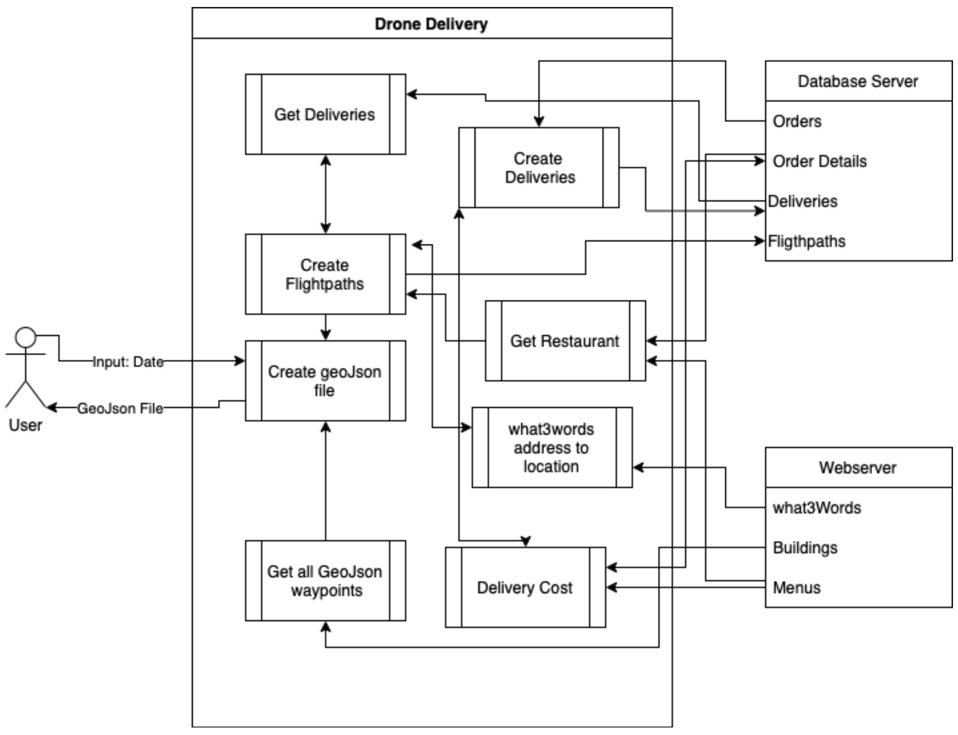
The system is comprised of two servers the apache derby database server and a Jibble webserver. These both interact with the application. The database server holds several tables mainly pertaining to orders, deliveries and the drone flightpaths. The webserver contains what3word addresses and locations, building json information and menus information. The user interacts with the application by running it with the port numbers for each server and the date for which they want to create a GeoJson file of the drone's flightpath.

Constraints

Both servers must be hosted locally on the machine for the application to run correctly. This is a limitation of the coursework piece as it would be unproductive to rely on servers hosted on the internet as they may be subject to downtime.

Use Case Overview

The only architecturally significant use case is to generate a geojson document of the drone's flightpath for a specified date and to calculate the cost of every delivery and output it to the database for the same specified date.



1. Get Deliveries
    - This process fetches all the deliveries from the database server.
2. Create Deliveries
    - If a *deliveries* table already exists in the database, it is dropped and a new one is created.
    - Order information are taken from the database server, the delivery cost is calculated via Delivery Cost and deliveries are appended to the *deliveries* in the database.
3. Create GeoJson file
    - Flightpaths are converted to GeoJson objects and are combined with the GeoJson waypoints.
    - The combined GeoJson is output as file type .geojson
4. Get Restaurant
    - Order items in order details table are taken from database using the order number.

    - Restaurant menus are taken from the webserver
    - Order items are compared to menu items and restaurant(s) for the order are deduced.
5. Location to what3words address
    - What3words is taken from the delivery object passed to the Create Flightpaths process.
    - Location is fetched using the address directory in the webserver and then pushed to the Create Flights Process.
6. Get all GeoJson waypoints
    - Building json data is fetched from the webserver
    - Json data is parsed as either type Location or type NoFlyZone depending on the geojson type.
7. Delivery Cost
    - Menu items are fetched from webserver as type Item
    - Order item names are fetched from database
    - Delivery cost is accumulated comparing order item names to menu items.
    - Delivery fee is added to delivery cost

Webserver

The webserver is a Jibble webserver, it contains:

- Json data for Buildings
- A directory for converting what3words to coordinates
- Menu information for each restaurant

Setup

The webserver is hosted locally on the client machine on a chosen port

Usage

The webserver is accessed using the java application by running;
"java -jar target/ilp-1.0-SNAPSHOT.jar *day month year webserverPortNumber databasePortNumber"* where italics mark place holders for integer values.

Database Server

The database server is an apache derby database server and contains the following tables:

- Orders
    - Order Number
        - Order ID
        - Character, Length 8
    - Delivery Date
        - Date the order is delivered
        - Date
    - Customer
        - Customer ID, uses student number
        - Character, Length 8
    - Deliver To
        - What3words address of delivery location
        - Alphanumeric Length 8

- Order Details
    - Order Number
        - Order ID
        - Type Character, Length 8
    - Item
        - List of items in written English format
        - Alphanumeric, Length 58
- Deliveries
    - Order Number
        - Order ID
        - Character, Length 8
    - Delivered To
        - What3words address of delivery location
        - Alphanumeric, Length 19
    - CostInPence
        - Cost of delivery in pence
        - Integer
- Flightpaths
    - fromLongitude
        - Start position Longitude
        - Decimal Number
    - from Latitude
        - Start position Latitude
        - Decimal Number
    - angle
        - bearing at which drone moves, specified below.
        - Integer
    - toLongitude
        - End position Longitude
        - Decimal Number
    - toLatitude
        - End position Latitude
        - Decimal Number

Angle System
The drone can only travel in angles of multiples of 10. The angle 0 means to face due East, moving anticlockwise such that 180 means to face West.

Setup
The database server is hosted locally on the client machine on a chosen port

Usage
The databse is accessed using the java application by running;
"java -jar target/ilp-1.0-SNAPSHOT.jar *day month year webserverPortNumber databasePortNumber"* where italics mark place holders for integer values.

Java Application

The java application is comprised of the following classes:

1. App
    - Runtime process for JAR file
2. Buildings
    - This class is used to handle the geojson files in the webserver under "/buildings/...", It was named Buildings for ease of understanding.
    - Holds methods for getting GeoJson points for map attributes
    - Holds methods to change map attributes to Locations
3. Database
    - Class manages IO for the Apache Derby Database
4. Delivery
    - Class holds Delivery Datatype
    - Holds order number, delivery destination and cost in pence.
5. Flightpath
    - Class holds one move made by the drone
    - Contains order number, start point, angle of movement and end point
6. implementationTask
    - This class holds the main methods to be implemented as outlined in the coursework document
7. Item
    - Holds datatype item
    - Contains item name and cost in pence
8. Location
    - Holds Location datatype
    - Contains location name and coordinates
9. LongLat
    - Holds LongLat datatype, Longitude and Latitude
    - Holds methods pertaining to calculation with LongLat Points
10. Map
    - Class holds GeoJson for all attributes on map
    - Holds methods for adding flightpaths to GeoJson
11. Menus
    - Class used to handle menu data
    - Holds methods to find restaurants for order and calculate delivery costs
12. NoFlyZone
    - Holds datatype for no-fly zone
    - Contains name and GeoJson polygon, for the zone.
13. Order
    - Holds Order datatype
    - Contains:
        i. Order Number
        ii. Delivery Date

        iii. Customer ID
        iv. Deliver to

14. Restaurant
    - Holds Datatype Restaurant
    - Contains:
        i. Name of Restaurant
        ii. Location
        iii. Menu as list of Item
15. Webserver
    - Holds Webserver Information
    - Contains methods for changing w3w to LongLat.

Drone Control Algorithm

Algorithm Requirements

The drone control algorithm has these main requirements:

- Start and return the Appleton tower
- Hover at when picking up or dropping off deliveries
- Do not fly through No-Fly Zones
- Complete no more than 1500 moves in a given day
- The drone must remain within the confinement area at all times

Goal of Algorithm

The goal of the algorithm is to create a list drone movements for all orders on a certain date that maintains all of the above requirements

Stages of Algorithm

1. Create flightpath table in database
2. For each order in database
    a. Calculate flightpath for each order
3. Combine all flightpaths
4. Convert flightpath to GeoJson

Algorithm Subprocesses

Creating Flightpath from start location to another location (flightpathLongLat)

This function would take two points and create a flightpath for the drone moving between them. This would be needed because the drone can only move a set distance each move. The process would loop through creating a list of flightpaths.

Creating Flightpath for delivery (flightpathDelivery)

This function would take a starting position and the delivery information, it would return a list of drone flightpaths for the pickup and drop off the delivery.

Creating Flightpath for a list of deliveries (flightpathDeliveries)

This function would take a list of deliveries and a starting position and create a list of flightpaths for all deliveries

Pseudocode

flightpathLongLat

```
INPUT startPos
INPUT endPos

INITIALISE Array Journey
INITIALISE Array FligthpathList
INITIALISE currentPos
Journey.add(startPos, endPos)

FOR EACH IN Journey:
    WHILE currentPos is NOT CloseTo endPos:
        INITIALISE INT Angle
        Angle = angleBetween(currentPos, endPos)
        INITIALISE nextPos(angle)
        INITIALISE Flightpath(currentPos, angle, nextPs)
        FlightPathList.add(Flightpath)
        currentPos = nextPos

    IF currentPOS CloseTo endPos:
        INITIALISE Flightpath(Hover)
        FlightPathList.add(Flightpath)

RETURN FlightPathList
```

flightpathDelivery

```
INPUT startPos
INPUT Delivery

INITIALISE Landmarks
INITIALISE No-Fly Zones
INITIALISE Array Journey
INITIALISE Array FlightpathListDelivery
INITIALISE currentPos
Journey.add(startPos)
Journey.add(resaurants)
Journey.add(endPos)

FOR x IN Journey:
    INITIALISE Angle
    INTIALISE Boolean validPath
    INITIALISE Array FligthpathList

    WHILE currentPos is NOT CloseTo endPos:
        INITIALISE INT Angle
        Angle = angleBetween(currentPos, endPos)
        INITIALISE nextPos(angle)
        IF nextPos is CONFINED AND NOT closeTo No-Fly Zones:
            INITIALISE Flightpath(currentPos, angle, nextPos)
            FlightPathList.add(Flightpath)
            currentPos = nextPos

        ELSE IF nextPos closeTo No-Fly Zones:
            validPath = FALSE
            INSERT closestLandmark, Journey at X
            REPEAT iteration at X

        ELSE IF currentPOS CloseTo endPos:
            INITIALISE Flightpath(Hover)
            FlightPathList.add(Flightpath)


        IF validPath is FALSE:
            EMPTY FlightpathList

    IF validPath is TRUE:
        APPEND flightpathListDelivery, flightpathList


RETURN flightpathListDelivery
```

flightpathDeliveries

```
INPUT startPos
INPUT Array Deliveries
INITIALISE Array Flightpaths

Flightpaths.add( flightpathDelivery(startPos, delivery[0]) )
Deliveries.remove(0)

FOR EACH in Deliveries:
    newStart = Deliveries.Last.EndLocation
    Flightpaths.add(fligthpathDelivery(newStart, Delivery)


RETURN Flightpath
```

Full Process

The final process for the drone control algorithm takes the date as an input and produces a GeoJson output of all deliveries on that date. The pseudocode below outlines this process.

Full Implementation

```
INPUT Date
FETCH Database Deliveries for Date as databaseDeliveries
INITIALISE Array Flightpaths
INITIALISE Location Appleton Tower
Flightpaths = flightpathsDeliveries(databaseDeliveries, Appleton Tower)

IF Flightpaths.size() > 1500:
    TRIM Flightpaths [0:1500]
    lastPos = Flightpaths.last.endPoint
    returnHome = flightpathLongLat(lastPos, AppletonTower)
    Flightpaths.add(returnHome)
ELSE:
    lastPos = Flightpaths.last.endPoint
    returnHome = flightpathLongLat(lastPos, AppletonTower)
    Flightpaths.add(returnHome)

GENERATE GeoJson for Flightpaths

RETURN GeoJson
```

Implementation

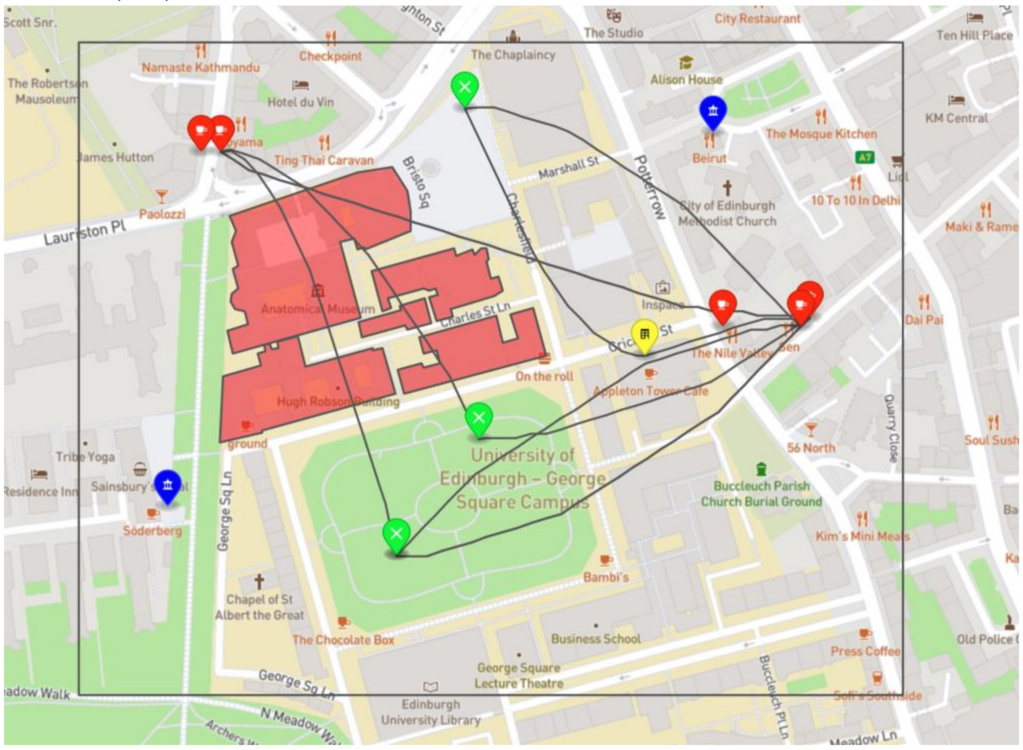
Unfortunately, due to time requirements I was unable to implement the following requirement for the algorithm:

- Do not fly through No-Fly Zones

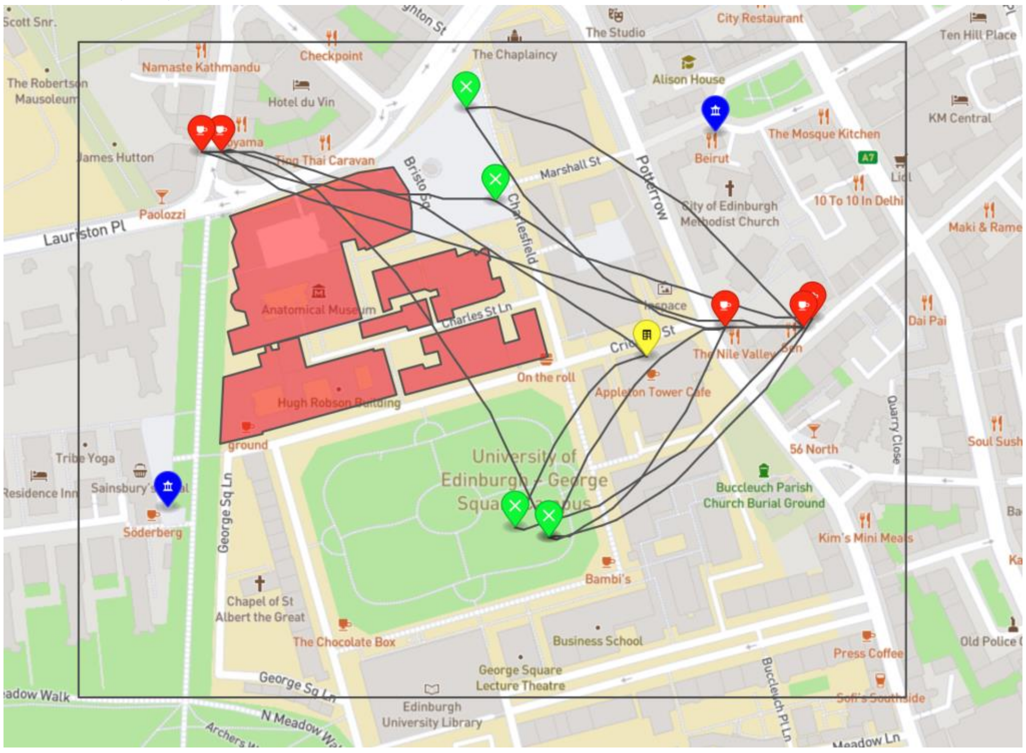
If more time was allowed I would have spent time implementing this and making the algorithm more efficient.

GeoJson Examples

GeoJson for 01/01/2022



GeoJson for 01/02/2022


Namaste Kathmandu
Checkpoint
The Chaplaincy
The Studio
City Restaurant
Ten Hill Place
Alison House
The Mosque Kitchen
KM Central
Hotel du Vin
Ting Thai Caravan
Marshall St
Potterrow
Beirut
10 To 10 In Delhi
City of Edinburgh Methodist Church
Maki & Ramen
Lauriston Pl
Paolozzi
James Hutton
Anatomical Museum
Charles St Ln
Inspace
The Nile Valley
Dai Pai
On the roll
Appleton Tower Cafe
Hugh Robson Building
56 North
Soul Sushi
University of Edinburgh – George Square Campus
Buccleuch Parish Church Burial Ground
Kim's Mini Meals
Tribe Yoga
Residence Inn
Sainsbury's
Söderberg
George Sq Ln
Bambi's
Chapel of St Albert the Great
The Chocolate Box
Business School
Press Coffee
Old Police
George Square Lecture Theatre
Edinburgh University Library
Buccleuch Pl Ln
Sofi's Southside
Meadow Walk
N Meadow Walk
Meadow Ln