SOFE 3700U: Data Management Systems

Final Report

Supervised By: Dr. Khalid A. Hafeez and TA Heyam Abunaseer

"WeDrone" – Drone Delivery System Application

Group 24 Members and IDs

Daniel Grewal 100768376

Usman Mahmood 100349839

Karanvir Bhogal 100748973

Mohammed Adnan Hashmi 100753115

Table of Contents

Introduction

The WeDrone Delivery System is a web application and database supporting any drone delivery service. To demonstrate a practical use case for using the service, we have implemented a prototype for a self-serve, person to person service for sending packages within Greater Toronto and surrounding area. WeDrone will enable users of the application to request pick-up and delivery of items after providing source and destination addresses. The WeDrone application utilizes real world geocode data using Google Maps API and Google Places/AutoComplete API to provide verified address data for the origin and destination addresses.

WeDrone will enable users to fulfil their automated delivery needs via a web-based application. For this implementation the scope of the delivery service will be limited geographically to the Greater Toronto and surrounding areas. Furthermore, this project will act as a proof-of-concept of the web application, so it will not use real deliveries, financial transactions, or actual drones. Additionally, the interface/communication protocols between drones and the application will also be out of scope of this project.

The WeDrone application is designed such that it can be scaled, adapted, and integrated into virtually any other delivery service system. Our application will provide a means to conveniently operate and manage a drone delivery service. The WeDrone service can be seamlessly deployed into an existing database platform to enable a convenient drone delivery option for your business or organization in any location. The WeDrone Delivery System will have several advantages over typical delivery services. For instance, automation allows for more economical and efficient delivery solutions. Utilizing drone technology allows for fast, safe, eco-friendly and reliable 24/7 point to point delivery capabilities in a convenient, self-managed and user-friendly delivery service application.

Background and Motivation

Over the past decade, online delivery services experienced a large boom in the industry, with millions of items being ordered and shipped everyday. However, these deliveries are always made using the standard modes of transportation which include automobiles, ships, and planes. Ships and planes are likely going to be the most reliable mode of transportation for overseas deliveries for a while, but automobiles have certain limitations currently when it comes to efficiency. For example, some of the issues with automobile transportation include traffic, fuel consumption, environmental damage, and human error. Many companies have tried to come up with solutions for these issues such as electric cars and self-driving cars, but they are usually not very common or haven't been implemented into the real world yet due to ethical and safety concerns. Another solution that is currently on the rise is the use of drones to complete deliveries.

Drones are an appealing concept to many companies as they can avoid traffic rules and the use of fossil fuels to run. The downside to using drones is how far they can travel in one flight and their carrying capacity for both size and weight. Those issues can be mitigated if you understand the limitations of drones and how to use them, which is exactly what our service is selling. Instead of worrying about one drone being unable to make a long-distance journey, we will set up multiple nodes around our service area with drones in place. If a drone is unable to make it the full distance and back, then we will have it so that the drone drops off it's product at the nearest node and a fully charged drone will take its place to complete the rest of the delivery. That way, deliveries can be made over longer distances without being limited by the flight distance of a single drone. Regarding the weight and size limits, we have set a cap so that only orders of a certain weight and size limit can be sent out.

Some current examples of similar services include Wing, Amazon Prime Air, OPS Flight Forward, and Flytrex. All these services use drones as a method of delivering packages to customers for

commercial use. The way our service will mainly differ from current services is using strategically placed nodes to greatly enhance the operating area of our drones. Currently, all the services listed above have a small flight distance and carrying capacity, with the higher end ones being able to carry 6.6 pounds and travel up to 6.2 miles [1]. Our drones will be able to travel 50 km on one charge and have a carrying capacity of 50 Kg, to ensure most ordered products can be delivered. Our service is also available for use by other companies as well, not just commercial use. That way we can expand the reach of drone usage in modern day deliveries. Overall, our main motivator for our project is to provide a service which is more practical than other drone services and more eco-friendly than current automotive delivery services.

System Design

Design Process

The WeDrone system was designed using the Attribute Driven Design (ADD) process. The ADD process takes a set of requirements as inputs and expects an architecture design and description as output. For this project specifically we utilized the ADD 3.0 process for a greenfield system in a mature domain using a web application architecture. We followed the 7-step ADD process and used 3 iterations to produce our final design.

Table 1: Use cases and descriptions

Use Case	Description
UC-1: Log in	A user logs into the system through a login/password page. Upon successful login, the user is presented with a different landing page depending on their role (e.g. user or admin).
UC-3: Submit a delivery order	A user enters the weight and dimensions of their delivery item. If it is overweight or oversize the user is prompted with an error. Otherwise, the user can proceed to provide a pickup/drop-off address and submit the delivery order. Once addresses are verified (using Google Maps/Places Autocomplete API service), the user is then given a quote for their delivery order with cost and time details. If the user accepts, the delivery order is submitted. If the user cancels, the order is abandoned, and fields reset.
UC-4: View order history	A user views their own order history with details about every delivery they have been successfully submitted and fulfilled. Order history can be filtered by date, status, distance and so on.
UC-6: Monitor deliveries	A user can view the deliveries in progress.
UC-7: Collect delivery data	The system will collect and store delivery order data into a database.
UC-9: View order invoice	A user can enter their order ID or click through from the order history page to see the invoice and details for that order.

Table 2: Quality attributes with scenarios, along with associated use cases

ID	Quality Attribute	Scenario	Associated Use Case
QA-1	Availability	All operations provided by the application must be available 24/7.	ALL
QA-2	Performance	All pages on the website must load within 3 seconds.	ALL
QA-3	Performance, Usability	Order history must load quickly and by default show all orders fulfilled.	UC-4,6,9
QA-4	Security	All orders are associated to a particular user, so it is known who made the order. All logins are protected by username and password.	UC-1
QA-6	Usability	The application will have an intuitive and easy to use UI.	UC-3,4,6,9
QA-7	Interoperability	The application will use the Google Maps/Places/AutoComplete APIs to validate user input and use real world geocode data for calculating drone metrics and flight paths.	UC-3, 9

Table 3: System constraints with descriptions

Constraints	Description
CON-1:	The system must be accessed through the latest version of either web browser (Chrome, Firefox, V4, IE8) in different platforms: Windows, OSX, and Linux.
CON-2:	The delivery must meet less than 50 kg in weight.
CON-3:	The delivery must be less than 1.5 cubic meters.
CON-4:	Users must enter a valid address for both pickup and drop off within the GTA and surrounding area.
CON-5:	There must be an established internet connection during selection and confirmation.

This project is specifically for providing a nimble self-serve delivery service to users, allowing the customer to have direct point-to-point management of their own deliveries. This approach minimizes the need for multiple parties to be involved in parcel delivery. Therefore, the only stakeholders involved

are the customers in our self-serve application. The following diagram outlines the general context for the WeDrone Delivery system:

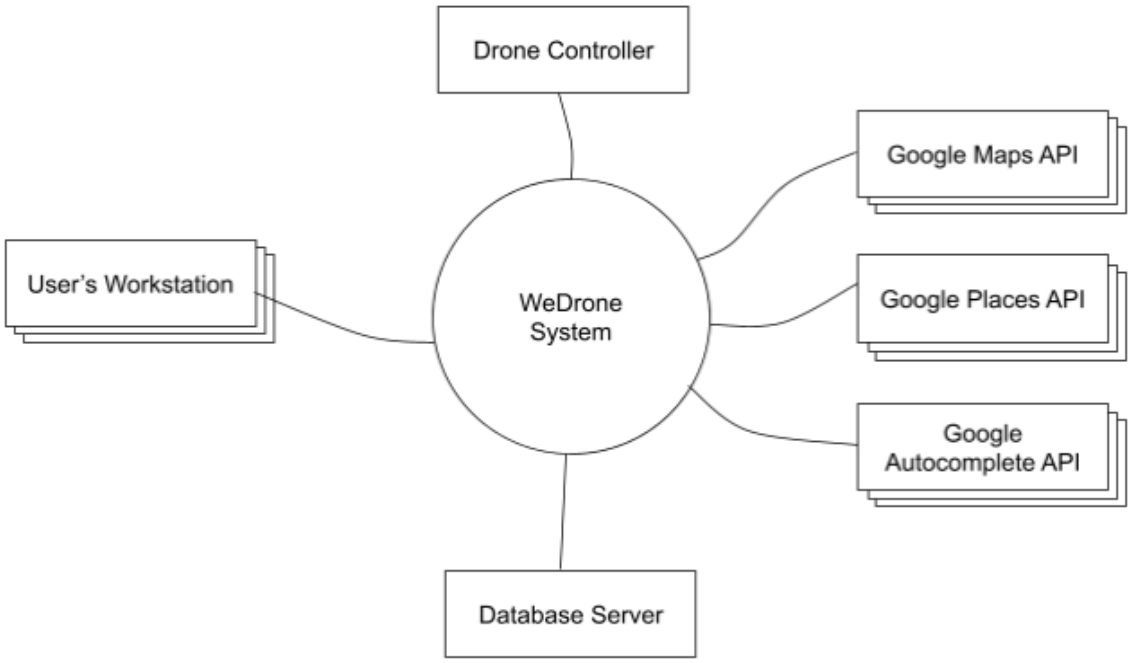


Figure 1: Context diagram for the WeDrone delivery system

In our scenario, the entire system except for the Drone Controller is considered. The scope of our application does not handle how the drones themselves operate. The bulk of the application resides on the server and is composed of a layered architecture: presentation, business and data layers. ASP.NET Core is naturally adapted to the Web Applications reference architecture. We chose .NET Core because it is cross platform and satisfies our constraints for being accessed in a web browser and using an existing database. Another advantage to using the .NET Core and Microsoft SQL is its scalability and performance over other frameworks. Most large organizations are using Microsoft .NET based systems application due to their performance, so integrating into their existing platforms and database servers will be seamless while also providing maximum performance for many concurrent users making many

requests. The WeDrone delivery system must maintain its integrity and performance to provide a dependable and reliable service.

Google Maps API Source Data and Implementation

Aside from using the Google API suite to verify user provided addresses, a critical part of our application is the implementation of the Google Maps API to provide geocode data to the database (latitude/longitude) in order to calculate the best route for a drone to fulfil a delivery. The WeDrone system depends on a network of nodes that will serve as hubs for drones to relay packages. Due to the distance and battery constraints of each drone, longer distance deliveries will require drones to pass packages off to one another at these nodes to complete a delivery. For our scenario, we have selected seven fixed locations as our nodes and entered them into our database. The database is designed to allow for nodes to be added or removed if necessary, depending on the use case and location of the delivery system.

The Google Maps API is used to determine all possible routes and distance vectors between the nodes (red markers in Figure 2). These flight legs are stored in the database and the pathfinding algorithm will then find the optimal route for the delivery.

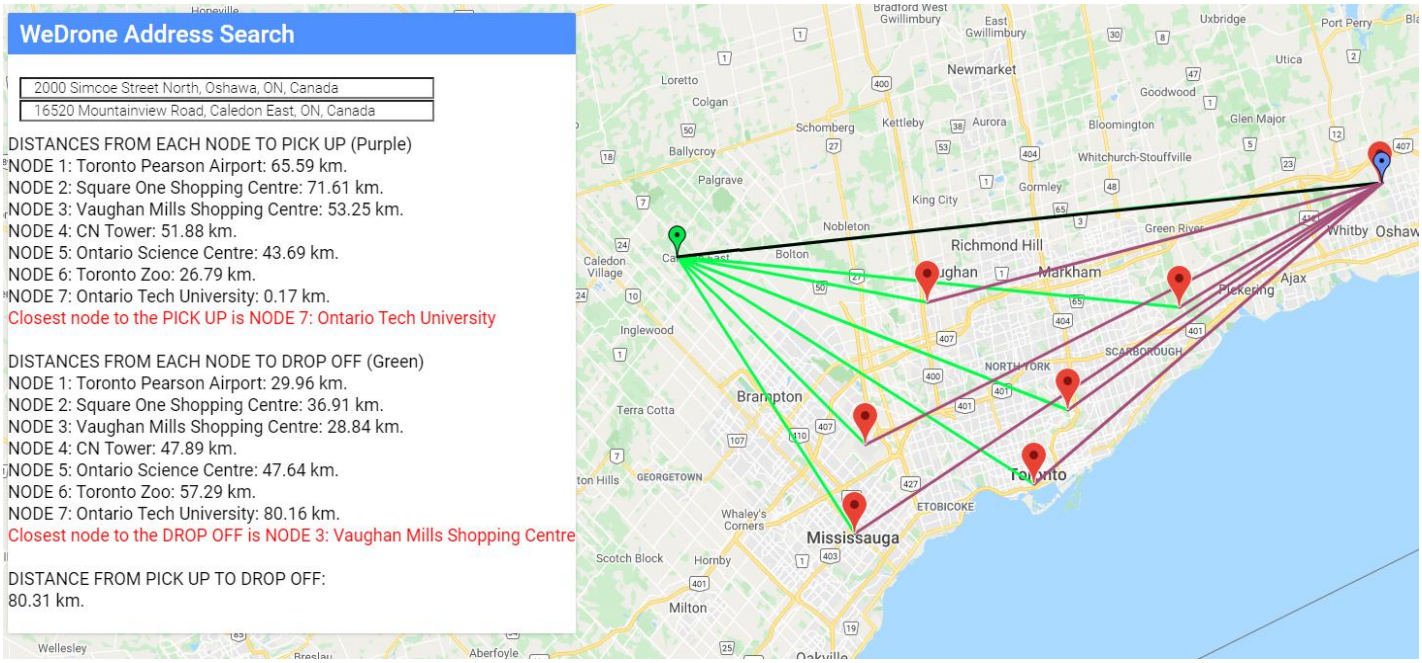


Figure 2: Google Maps API implementation for finding all flight paths and legs

The green and blue markers represent the pickup and drop-off locations specified by the user. The system will determine which node in the network is closest to the origin and destination, and therefore find the appropriate path to connect them. If the origin and destination locations are within range of each other without the use of the network nodes, the application will determine that the drone can make a direct route which bypasses the need for the network of nodes to relay the package. The Microsoft SQL statements for producing the database tables for flightLegs and flightRoutes (which is the series of flight legs that connects the nodes) can be seen in Appendix A later in this report, along with the source data fetched from the Google Maps API. Details for the code and how this was implemented in the application is all found on GitHub (link to repository files is available in the Appendix later in the report). Once the route is determined and the path is chosen from the database, a quote for the delivery can be presented to the user based on distance and cost. All configurations for deliveries can be edited in a config JSON file (“appsettings.json” as shown in Appendix C) where there are parameters for drone constraints (i.e. weight capacity, speed, cost per kilometer etc.). The external configuration in the design allows for

the parameters of the delivery to be configured based on what business use case is desired. This is a key feature for WeDrone to be conveniently adaptable and modifiable for any use case scenario or business application.

Test Cases

Successfully integrating the Google APIs for the address and geocode data was a challenging task for this project. We had to learn how to use the asynchronous features in C# to use the Google Maps API. In order to make sure we are fetching the API data correctly in our application and interfaces, a series of test cases for the APIs was created (source code for the test cases can be seen in Appendix D and in file "APIClientTests.cs").

`GetQueryURI_ShouldReturnURIGivenValidQuery()`

This test is for making sure the Google API returns a valid URL with the address information once input string is provided.

`GetQueryURI_ShouldReturnURIAfterTrimmingLeadingAndTrailingSpaces()`

This test is for making sure the Google API returns a valid URL even if the user input has leading or trailing spaces in the string query.

`GetAddresses_ShouldReturnNullGivenEmptyString()`

This test is for making sure the Google API returns null result if given an empty string to ensure there is no garbage data or error produced in the application.

`GetAddresses_ShouldReturnValidAddressResponse()`

This test is for making sure the Google API returns a valid address response in the correct postal address format (street number, street type, street direction, city, province, postal code, country: e.g. 101 Bloor St E, Oshawa, ON L1H 3M3, Canada).

Note: All instructions to set up the project and run it locally is available in the readme file shown in Appendix 0. Also, link to Github repository and project files can be found in Appendix 1 (link is https://github.com/danielgrewal/WeDrone)

Database Schema and Views

We developed our tables and schema as seen in Appendix A (SQL statements from "WeDrone.sql" file) and Appendix E (E-R Schema Diagram with key relationships). The C# application logic (in the FlightPlanner class) determines the possible flight legs to find the best delivery route based on the pickup and drop off locations provided by the user. The following is a brief walkthrough of the SQL statements for the WeDrone application (Appendix A).

Once the database is created the first step is to create the necessary tables and key relationships using SQL. Tables for the Locations, Delivery Status, Users, Flightlegs, FlightRoutes, FlightRouteSteps, Orders and OrderHistory are all created. Foreign key relationships are defined as seen in the attached E-R Schema Diagram (Appendix E). Subsequently the precomputed data required for the application to function is now prepopulated into the WeDrone database. Geocode data for the node locations, text for the drone statuses, test usernames/passwords, all possible combinations of FlightsRoutes, FlightLegs and FlightRouteSteps are all inserted into their respective tables. Data for the orders will be stored into the Orders and OrderHistory tables once deliveries are created by users. The next section of the SQL file is for creating indexes to optimize queries. Although indexes were not required for this project, we decided to follow best design practices and adhere to one of our top

priority design goals for enabling the application to be scalable and reliable. Indexes will ensure that the queries are efficient and less costly once there is an abundance of records in the database. Indexes minimize the need to excessively read the database (and storage media) when a query is processed resulting in unnecessary full table scans.

Furthermore, we have implemented transactions into our database design. As seen in the beginning of the SQL script, we have initiated a transaction after the database is created, and we have a Commit after the last index is created. This block of the script is part of a single transaction and ensures this unit of work in the database creation is treated in a coherent and reliable manner. Breaking the script (work) into transaction sections ensures processes are done reliably and systematically. Transactions maintain integrity and durability in the database and therefore the performance of the application. From the ACID database rules, a transaction must be:

1. Atomic - it is one unit of work and does not dependent on previous and following transactions.
2. Consistent - data is either committed or roll back, no “in-between” case where something has been updated and something hasn’t.
3. Isolated - no transaction sees the intermediate results of the current transaction.
4. Durable - the values persist if the data had been committed even if the system crashes right after.

A new transaction is started for when we create the Views. The 1st view (vw_ShowAllOrders) computes a join of at least three tables. This view gets all orders in the WeDrone system, who ordered them, their current status, and their origin/destination and presents it in a user readable manner. The 2nd view (vw_OrdersWithDistanceNotCancelled) uses nested queries with the ANY or ALL operator and uses a GROUP BY clause. This view returns all orders that were not cancelled and also provides the aggregate sum of the distance travelled in each order. The 3rd view (vw_CustomersWithFilledOrders) is a correlated nested query. This view returns user records for all users that have created orders that have

been delivered. The 4th view (vw_AllUsersAndTheirOrders) uses a full join. This view returns all users and all of their associated orders, if they have made any. The 5th view (sp_GetFlightPlan) was converted to a stored procedure to allow the recursive CTE to be passed in initial values to build from user input parameters for the RouteStart and RouteEnd. We changed the 5th view (vw_FlightPlanCTE) from deliverable Phase II to a stored procedure (sp_GetFlightPlan) because it required input parameters to work properly, which a view cannot do. This view is critical to the application and used to generate the flightpaths for the drones. The 6th view (vw_OrdersWithWeightOver10) returns all orders that are over 10 kilograms in weight. The 7th view (vw_OrdersWithVolumeOver1) returns all orders that are over 1 cubic meter in volume. The 8th view (vw_ShowFacilityNodes) returns all location nodes that are marked as "facility nodes". These are the nodes that are fixed and represent where drones are stored and where payloads are relayed. Later in the application life cycle when more nodes are added, we want to easily be able to find these locations from the stored user locations. The 9th view (vw_FlightLegsLessThan10) returns all flight legs that are less than 10 km. These are the shortest distance routes between nodes. Finally, the 10th view (vw_OrdersDelivered) returns all orders that were successfully delivered. All these views are displayed to the user using a dashboard webpage using HTML and CSS components. Another commit is then used to close off the transaction specifically for creating the views, which concludes the SQL script. All SQL statements for creating the views specified in Phase II can be found in Appendix A. It is also worth mentioning that at first it was planned to de-normalize a static reference to historical records for the OrderHistory. However, it was decided to instead use normalized IDs as this method preferred in a more traditional database design for this course.

Observations and Analysis

There are endless applications for automated drone deliveries in many high growth markets and industries. For example, Food and Grocery, Retail, Medical and Pharmaceuticals, Business-to-Business, Agricultural/Industrial, Hazardous/Waste Materials and so on. As the drone technology and communication networks mature, automated drone delivery services will be increasingly energy efficient and cost effective. This will empower the WeDrone service to inherently become more affordable and accessible to all users, businesses and organizations. WeDrone is an adaptable and scalable system design that provides a time and cost-efficient method to conveniently operate and manage a drone service for any business. WeDrone takes a step into the future by incorporating an external API allowing for a much more efficient point to point drone delivery system. Our automation allows for a more fast and safe economical delivery solution by decreasing the amount of people interacting with the package itself. Having direct customer interaction with the drone system that will reduce labour cost, accidents, and package mishaps.

The future growth of drones and its many applications are rapidly rising more so now than ever. It is quite apparent that conventional delivery systems hold many flaws that can be eliminated with the implementation of automated drones. For instance, an automated drone delivery system would attract greater investment in response and readiness. Allowing for ease of medical personnel and rapid transportation of medical supplies where needed, this will alleviate pressure on the public health system. The popularity of drones has skyrocketed due to their low prices, and many uses over different industries and environments. Implementation of drones within certain industries have helped in reducing costs for both the customer and the company [2].

Although drones are incredibly beneficial with their many practical uses, there are regulations and obstacles that prevent free use of the transportation vehicle. Drone pilots must follow the rules

stated in the Canadian Aviation Regulations. Any drones up to 25 kg must be marked and registered with the pilot who must carry a valid drone pilot certificate [3]. Certain minimum distances must be met in which the drone must be flying within certain bounds of other objects. Drones must fly below 122 meters in the air and away from bystanders at a minimum horizontal distance of 30 meters [3]. Alongside these rules, drones must avoid certain areas such as forest fires, airports, advertised events and far away from other aircrafts.

Conclusion

Over the course of this project, we have come across many challenges. The first obstacle was to build a novel delivery system that challenged the status quo. The second obstacle was to effectively determine the optimal flight path which we resolve by implementing a package relay network across the nodes and pathfinding algorithm. Additionally, learning how to integrate the Google APIs into our codebase and securing the API key in order to share the project on GitHub was a new challenge and learning experience. Finally, working amongst a team with varying experience and skill sets was demanding.

The WeDrone database system will empower consumers and organizations to conveniently manage their delivery needs while being designed to seamlessly integrate and grow with the business. The future of WeDrone would be to integrate the system to include conventional means of transportation rather than limit the option to drones itself. Vehicles such as autonomous trucks, cars, ships and planes are a few examples of how our API and database could integrate with other methods of delivery. Additionally, expanding our project and allowing for deliveries to be made across a larger service area is a potential next step for the WeDrone application. Drones are projected to become a major vehicle for the transportation of goods while cutting costs and simultaneously being eco-friendly, helping in reducing the effects of carbon emission found in traditional delivery services.

References

[1] S. Ueland, “8 commercial drone delivery companies,” *Practical Ecommerce*, 25-Jan-2021. [Online]. Available: https://www.practicalecommerce.com/8-commercial-drone-delivery-companies. [Accessed: 30-Nov-2021].

[2] “Why are drones suddenly so popular? - panda security,” *Panda Security Mediacenter*, 28-Mar-2018. [Online]. Available: https://www.pandasecurity.com/en/mediacenter/news/what-is-the-future-of-drones/. [Accessed: 30-Nov-2021].

[3] Transport Canada, “Flying your drone safely and legally,” *Transport Canada*, 01-Sep-2020. [Online]. Available: https://tc.canada.ca/en/aviation/drone-safety/learn-rules-you-fly-your-drone/flying-your-drone-safely-legally. [Accessed: 30-Nov-2021].

Appendices

Appendix 0

Installation Instructions

Set up ASP.NET Core

1. Navigate to https://dotnet.microsoft.com/download.
2. Download **.NET SDK x64.**
3. Install the downloaded executable file.

Set up Microsoft SQL Server

1. Navigate to https://www.microsoft.com/en-us/sql-server/sql-server-downloads.
2. Download the **SQL Server 2019 Developer Edition** by clicking on the Download Now button under "Developer".
3. Run the downloaded executable file and follow the instructions to perform a **Basic Installation.**

Set up Visual Studio (Note: this is a different product than Visual Studio Code)

1. Navigate to https://visualstudio.microsoft.com/ and download **Visual Studio Community 2022.**
2. Run the downloaded executable file and follow the instructions; please ensure you minimally select the select the **ASP.NET and web development** workload.

Clone Repository and Open the Project

1. Using git command-line tools, clone the https://github.com/danielgrewal/WeDrone.git repository. The repository can also be found on https://github.com/danielgrewal/WeDrone if the git command-line tools are not available.
2. Open the project folder and run **WeDrone.sln** under the **src** sub-folder. This will open Visual Studio. This window can be minimized for now to set up the database.

Set up Database and API Connection

1. Open Microsoft SQL Server Management Studio and run **WeDrone.sql** in a new query window to create the initial database. This file can be found in the root directory of the repository.
2. Once the database is created, open up the minimized Visual Studio window and under the **WeDrone.Web project**, open the **appsettings.json** file.
3. Add the database connection string between the double quotes on the line that says **"WeDroneContext": ""** to point to the database.
4. For the application to function a Google API key will need to be provided on the line that says **"Key": "InsertYourAPIKeyHere"** by replacing *InsertYourAPIKeyHere* with the actual key. The application will not function if the correct connection string or an API key are not provided.

Run Application

1. At the top of the Visual Studio IDE, press the play button and a browser window will launch with the application running. Use the username **"usman"** and the password "**password"** for testing purposes.

Appendix 1

Link to GitHub repository and project files: https://github.com/danielgrewal/WeDrone

This has the most up to date and used code for the implementation.

Appendix A

Microsoft SQL file commands for creating the schema and tables, populating location data from the Google Maps API and for all views and precomputed queries used by the application.

```
CREATE DATABASE WeDrone;
GO

IF OBJECT_ID(N'[__EFMigrationsHistory]') IS NULL
BEGIN
    CREATE TABLE [__EFMigrationsHistory] (
        [MigrationId] nvarchar(150) NOT NULL,
        [ProductVersion] nvarchar(32) NOT NULL,
        CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
    );
END;
GO

BEGIN TRANSACTION;
GO

CREATE TABLE [Locations] (
    [LocationId] int NOT NULL IDENTITY,
    [Name] nvarchar(255) NULL,
    [Latitude] decimal(17,14) NOT NULL,
    [Longitude] decimal(17,14) NOT NULL,
    [Address] nvarchar(255) NULL,
    [IsDroneFacility] bit NOT NULL,
    CONSTRAINT [PK_Locations] PRIMARY KEY ([LocationId])
);
GO

CREATE TABLE [Status] (
    [StatusId] int NOT NULL IDENTITY,
    [Name] nvarchar(50) NULL,
    CONSTRAINT [PK_Status] PRIMARY KEY ([StatusId])
);
GO

CREATE TABLE [Users] (
    [UserId] int NOT NULL IDENTITY,
    [Username] nvarchar(50) NOT NULL,
    [Password] nvarchar(50) NOT NULL,
    [Name] nvarchar(255) NOT NULL,
    CONSTRAINT [PK_Users] PRIMARY KEY ([UserId])
);
GO

CREATE TABLE [Flightlegs] (
    [FlightlegId] int NOT NULL IDENTITY,
    [FromId] int NOT NULL,
    [ToId] int NOT NULL,
```

```
    [Distance] decimal(10,2) NOT NULL,
    CONSTRAINT [PK_Flightlegs] PRIMARY KEY ([FlightlegId]),
    CONSTRAINT [FK_Flightlegs_Locations_FromId] FOREIGN KEY ([FromId]) REFERENCES
[Locations] ([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_Flightlegs_Locations_ToId] FOREIGN KEY ([ToId]) REFERENCES [Locations]
([LocationId]) ON DELETE NO ACTION
);
GO

CREATE TABLE [FlightRoutes] (
    [FlightRouteId] int NOT NULL IDENTITY,
    [RouteStartId] int NOT NULL,
    [RouteEndId] int NOT NULL,
    CONSTRAINT [PK_FlightRoutes] PRIMARY KEY ([FlightRouteId]),
    CONSTRAINT [FK_FlightRoutes_Locations_RouteEndId] FOREIGN KEY ([RouteEndId]) REFERENCES
[Locations] ([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_FlightRoutes_Locations_RouteStartId] FOREIGN KEY ([RouteStartId])
REFERENCES [Locations] ([LocationId]) ON DELETE NO ACTION
);
GO

CREATE TABLE [FlightRouteSteps] (
    [FlightRouteId] int NOT NULL,
    [CurrentLegId] int NOT NULL,
    [NextLegId] int NULL,
    [IsInitialLeg] bit NOT NULL,
    CONSTRAINT [PK_FlightRouteSteps] PRIMARY KEY ([FlightRouteId], [CurrentLegId]),
    CONSTRAINT [FK_FlightRouteSteps_Flightlegs_CurrentLegId] FOREIGN KEY ([CurrentLegId])
REFERENCES [Flightlegs] ([FlightlegId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_FlightRouteSteps_Flightlegs_NextLegId] FOREIGN KEY ([NextLegId])
REFERENCES [Flightlegs] ([FlightlegId]) ON DELETE NO ACTION
);
GO

CREATE TABLE [Orders] (
    [OrderId] int NOT NULL IDENTITY,
    [UserId] int NOT NULL,
    [OriginId] int NOT NULL,
    [DestinationId] int NOT NULL,
    [FlightRouteId] int NULL,
    [Weight] decimal(10,2) NOT NULL,
    [Volume] decimal(10,2) NOT NULL,
    [Cost] decimal(10,2) NOT NULL,
    [OrderCreated] datetime2 NULL,
    [OrderFilled] datetime2 NULL,
    CONSTRAINT [PK_Orders] PRIMARY KEY ([OrderId]),
    CONSTRAINT [FK_Orders_FlightRoutes_FlightRouteId] FOREIGN KEY ([FlightRouteId])
REFERENCES [FlightRoutes] ([FlightRouteId]),
    CONSTRAINT [FK_Orders_Locations_DestinationId] FOREIGN KEY ([DestinationId]) REFERENCES
[Locations] ([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_Orders_Locations_OriginId] FOREIGN KEY ([OriginId]) REFERENCES
[Locations] ([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_Orders_Users_UserId] FOREIGN KEY ([UserId]) REFERENCES [Users] ([UserId])
ON DELETE CASCADE
);
GO

CREATE TABLE [OrderHistory] (
    [Orderid] int NOT NULL,
    [ValidFrom] datetime2 NOT NULL,
```

```
    [ValidTo] datetime2 NOT NULL,
    [FromId] int NULL,
    [ToId] int NULL,
    [StatusId] int NOT NULL,
    [Distance] decimal(10,2) NULL,
    CONSTRAINT [PK_OrderHistory] PRIMARY KEY ([Orderid], [ValidFrom]),
    CONSTRAINT [FK_OrderHistory_Locations_FromId] FOREIGN KEY ([FromId]) REFERENCES
[Locations] ([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_OrderHistory_Locations_ToId] FOREIGN KEY ([ToId]) REFERENCES [Locations]
([LocationId]) ON DELETE NO ACTION,
    CONSTRAINT [FK_OrderHistory_Orders_Orderid] FOREIGN KEY ([Orderid]) REFERENCES [Orders]
([OrderId]) ON DELETE CASCADE,
    CONSTRAINT [FK_OrderHistory_Status_StatusId] FOREIGN KEY ([StatusId]) REFERENCES
[Status] ([StatusId]) ON DELETE CASCADE
);
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'LocationId',
N'Address', N'IsDroneFacility', N'Latitude', N'Longitude', N'Name') AND [object_id] =
OBJECT_ID(N'[Locations]'))
    SET IDENTITY_INSERT [Locations] ON;
INSERT INTO [Locations] ([LocationId], [Address], [IsDroneFacility], [Latitude],
[Longitude], [Name])
VALUES (1, N'6301 Silver Dart Dr, Mississauga, ON L5P 1B2', CAST(1 AS bit),
43.68232877980147, -79.62661047567958, N'Toronto Pearson International Airport'),
(2, N'100 City Centre Dr, Mississauga, ON L5B 2C9', CAST(1 AS bit), 43.5932704575367, -
79.6418354765629, N'Square One Shopping Centre'),
(3, N'1 Bass Pro Mills Dr, Vaughan, ON L4K 5W4', CAST(1 AS bit), 43.82549302652521, -
79.5381447146669, N'Vaughan Mills Mall'),
(4, N'290 Bremner Blvd, Toronto, ON M5V 3L9', CAST(1 AS bit), 43.64272145936629, -
79.38704607234244, N'CN Tower'),
(5, N'770 Don Mills Rd., North York, ON M3C 1T3', CAST(1 AS bit), 43.71718851953277, -
79.33851344772478, N'Ontario Science Centre'),
(6, N'2000 Meadowvale Rd, Toronto, ON M1B 5K7', CAST(1 AS bit), 43.82096417612802, -
79.1812287514316, N'Toronto Zoo'),
(7, N'2000 Simcoe St N, Oshawa, ON L1G 0C5', CAST(1 AS bit), 43.94565647325994, -
78.89679613001036, N'Ontario Technology University');
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'LocationId',
N'Address', N'IsDroneFacility', N'Latitude', N'Longitude', N'Name') AND [object_id] =
OBJECT_ID(N'[Locations]'))
    SET IDENTITY_INSERT [Locations] OFF;
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'StatusId', N'Name') AND
[object_id] = OBJECT_ID(N'[Status]'))
    SET IDENTITY_INSERT [Status] ON;
INSERT INTO [Status] ([StatusId], [Name])
VALUES (1, N'Pending Pick-up'),
(2, N'Order Retrieved'),
(3, N'In Transit'),
(4, N'Pending Drop-off'),
(5, N'Delivered'),
(6, N'Cancelled');
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'StatusId', N'Name') AND
[object_id] = OBJECT_ID(N'[Status]'))
    SET IDENTITY_INSERT [Status] OFF;
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'UserId', N'Name',
N'Password', N'Username') AND [object_id] = OBJECT_ID(N'[Users]'))
```

```
    SET IDENTITY_INSERT [Users] ON;
INSERT INTO [Users] ([UserId], [Name], [Password], [Username])
VALUES (1, N'Usman Mahmood', N'password', N'usman'),
(2, N'Daniel Grewal', N'password', N'daniel'),
(3, N'Mohammed Adnan Hashmi', N'password', N'adnan'),
(4, N'Karanvir Bhogal', N'password', N'karanvir');
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'UserId', N'Name',
N'Password', N'Username') AND [object_id] = OBJECT_ID(N'[Users]'))
    SET IDENTITY_INSERT [Users] OFF;
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'FlightRouteId',
N'RouteEndId', N'RouteStartId') AND [object_id] = OBJECT_ID(N'[FlightRoutes]'))
    SET IDENTITY_INSERT [FlightRoutes] ON;
INSERT INTO [FlightRoutes] ([FlightRouteId], [RouteEndId], [RouteStartId])
VALUES (1, 2, 1),
(2, 1, 2),
(3, 3, 1),
(4, 1, 3),
(5, 4, 1),
(6, 1, 4),
(7, 5, 1),
(8, 1, 5),
(9, 6, 1),
(10, 1, 6),
(11, 7, 1),
(12, 1, 7),
(13, 3, 2),
(14, 2, 3),
(15, 4, 2),
(16, 2, 4),
(17, 5, 2),
(18, 2, 5),
(19, 6, 2),
(20, 2, 6),
(21, 7, 2),
(22, 2, 7),
(23, 4, 3),
(24, 3, 4),
(25, 5, 3),
(26, 3, 5),
(27, 6, 3),
(28, 3, 6),
(29, 7, 3),
(30, 3, 7),
(31, 5, 4),
(32, 4, 5),
(33, 6, 4),
(34, 4, 6),
(35, 7, 4),
(36, 4, 7),
(37, 6, 5),
(38, 5, 6),
(39, 7, 5),
(40, 5, 7),
(41, 7, 6),
(42, 6, 7);
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'FlightRouteId',
N'RouteEndId', N'RouteStartId') AND [object_id] = OBJECT_ID(N'[FlightRoutes]'))
    SET IDENTITY_INSERT [FlightRoutes] OFF;
```

```
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'FlightlegId',
N'Distance', N'FromId', N'ToId') AND [object_id] = OBJECT_ID(N'[Flightlegs]'))
    SET IDENTITY_INSERT [Flightlegs] ON;
INSERT INTO [Flightlegs] ([FlightlegId], [Distance], [FromId], [ToId])
VALUES (1, 9.98, 1, 2),
(2, 9.98, 2, 1),
(3, 17.43, 1, 3),
(4, 17.43, 3, 1),
(5, 19.77, 1, 4),
(6, 19.77, 4, 1),
(7, 23.48, 1, 5),
(8, 23.48, 5, 1),
(9, 38.95, 1, 6),
(10, 38.95, 6, 1),
(11, 65.47, 1, 7),
(12, 65.47, 7, 1),
(13, 27.13, 2, 3),
(14, 27.13, 3, 2),
(15, 21.24, 2, 4),
(16, 21.24, 4, 2),
(17, 28.02, 2, 5),
(18, 28.02, 5, 2),
(19, 44.85, 2, 6),
(20, 44.85, 6, 2),
(21, 71.51, 2, 7),
(22, 71.51, 7, 2),
(23, 23.67, 3, 4),
(24, 23.67, 4, 3),
(25, 20.05, 3, 5),
(26, 20.05, 5, 3),
(27, 28.64, 3, 6),
(28, 28.64, 6, 3),
(29, 53.11, 3, 7),
(30, 53.11, 7, 3),
(31, 9.15, 4, 5),
(32, 9.15, 5, 4),
(33, 25.81, 4, 6),
(34, 25.81, 6, 4),
(35, 51.8, 4, 7),
(36, 51.8, 7, 4),
(37, 17.11, 5, 6),
(38, 17.11, 6, 5),
(39, 43.6, 5, 7),
(40, 43.6, 7, 5),
(41, 26.68, 6, 7),
(42, 26.68, 7, 6);
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'FlightlegId',
N'Distance', N'FromId', N'ToId') AND [object_id] = OBJECT_ID(N'[Flightlegs]'))
    SET IDENTITY_INSERT [Flightlegs] OFF;
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CurrentLegId',
N'FlightRouteId', N'IsInitialLeg', N'NextLegId') AND [object_id] =
OBJECT_ID(N'[FlightRouteSteps]'))
    SET IDENTITY_INSERT [FlightRouteSteps] ON;
INSERT INTO [FlightRouteSteps] ([CurrentLegId], [FlightRouteId], [IsInitialLeg],
[NextLegId])
VALUES (1, 1, CAST(1 AS bit), NULL),
```

```
(2, 2, CAST(1 AS bit), NULL),
(3, 3, CAST(1 AS bit), NULL),
(4, 4, CAST(1 AS bit), NULL),
(5, 5, CAST(1 AS bit), NULL),
(6, 6, CAST(1 AS bit), NULL),
(7, 7, CAST(1 AS bit), NULL),
(8, 8, CAST(1 AS bit), NULL),
(7, 9, CAST(1 AS bit), 37),
(37, 9, CAST(0 AS bit), NULL),
(8, 10, CAST(0 AS bit), NULL),
(38, 10, CAST(1 AS bit), 8),
(7, 11, CAST(1 AS bit), 37),
(37, 11, CAST(0 AS bit), 41),
(41, 11, CAST(0 AS bit), NULL),
(8, 12, CAST(0 AS bit), NULL),
(38, 12, CAST(0 AS bit), 8),
(42, 12, CAST(1 AS bit), 38),
(2, 13, CAST(1 AS bit), 3),
(3, 13, CAST(0 AS bit), NULL),
(1, 14, CAST(0 AS bit), NULL),
(4, 14, CAST(1 AS bit), 1),
(15, 15, CAST(1 AS bit), NULL),
(16, 16, CAST(1 AS bit), NULL),
(15, 17, CAST(1 AS bit), 31),
(31, 17, CAST(0 AS bit), NULL),
(16, 18, CAST(0 AS bit), NULL),
(32, 18, CAST(1 AS bit), 16),
(15, 19, CAST(1 AS bit), 31),
(31, 19, CAST(0 AS bit), 37),
(37, 19, CAST(0 AS bit), NULL),
(16, 20, CAST(0 AS bit), NULL),
(32, 20, CAST(0 AS bit), 16),
(38, 20, CAST(1 AS bit), 32),
(15, 21, CAST(1 AS bit), 31),
(31, 21, CAST(0 AS bit), 37),
(37, 21, CAST(0 AS bit), 41),
(41, 21, CAST(0 AS bit), NULL),
(16, 22, CAST(0 AS bit), NULL),
(32, 22, CAST(0 AS bit), 16),
(38, 22, CAST(0 AS bit), 32),
(42, 22, CAST(1 AS bit), 38);
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CurrentLegId',
N'FlightRouteId', N'IsInitialLeg', N'NextLegId') AND [object_id] =
OBJECT_ID(N'[FlightRouteSteps]'))
    SET IDENTITY_INSERT [FlightRouteSteps] OFF;
GO

IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CurrentLegId',
N'FlightRouteId', N'IsInitialLeg', N'NextLegId') AND [object_id] =
OBJECT_ID(N'[FlightRouteSteps]'))
    SET IDENTITY_INSERT [FlightRouteSteps] ON;
INSERT INTO [FlightRouteSteps] ([CurrentLegId], [FlightRouteId], [IsInitialLeg],
[NextLegId])
VALUES (7, 23, CAST(1 AS bit), NULL),
(8, 24, CAST(1 AS bit), NULL),
(25, 25, CAST(1 AS bit), NULL),
(26, 26, CAST(1 AS bit), NULL),
(27, 27, CAST(1 AS bit), NULL),
(28, 28, CAST(1 AS bit), NULL),
(27, 29, CAST(1 AS bit), 41),
```

```
(41, 29, CAST(0 AS bit), NULL),
(28, 30, CAST(0 AS bit), NULL),
(42, 30, CAST(1 AS bit), 28),
(31, 31, CAST(1 AS bit), NULL),
(32, 32, CAST(1 AS bit), NULL),
(31, 33, CAST(1 AS bit), 37),
(37, 33, CAST(0 AS bit), NULL),
(32, 34, CAST(0 AS bit), NULL),
(38, 34, CAST(1 AS bit), 32),
(31, 35, CAST(1 AS bit), 37),
(37, 35, CAST(0 AS bit), 41),
(41, 35, CAST(0 AS bit), NULL),
(32, 36, CAST(0 AS bit), NULL),
(38, 36, CAST(0 AS bit), 32),
(42, 36, CAST(1 AS bit), 38),
(37, 37, CAST(1 AS bit), NULL),
(38, 38, CAST(1 AS bit), NULL),
(37, 39, CAST(1 AS bit), 41),
(41, 39, CAST(0 AS bit), NULL),
(38, 40, CAST(0 AS bit), NULL),
(42, 40, CAST(1 AS bit), 38),
(41, 41, CAST(1 AS bit), NULL),
(42, 42, CAST(1 AS bit), NULL);
IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CurrentLegId',
N'FlightRouteId', N'IsInitialLeg', N'NextLegId') AND [object_id] =
OBJECT_ID(N'[FlightRouteSteps]'))
    SET IDENTITY_INSERT [FlightRouteSteps] OFF;
GO

CREATE INDEX [IX_Flightlegs_FromId] ON [Flightlegs] ([FromId]);
GO

CREATE INDEX [IX_Flightlegs_ToId] ON [Flightlegs] ([ToId]);
GO

CREATE INDEX [IX_FlightRoutes_RouteEndId] ON [FlightRoutes] ([RouteEndId]);
GO

CREATE INDEX [IX_FlightRoutes_RouteStartId] ON [FlightRoutes] ([RouteStartId]);
GO

CREATE INDEX [IX_FlightRouteSteps_CurrentLegId] ON [FlightRouteSteps] ([CurrentLegId]);
GO

CREATE INDEX [IX_FlightRouteSteps_NextLegId] ON [FlightRouteSteps] ([NextLegId]);
GO

CREATE UNIQUE INDEX [IX_Locations_Address] ON [Locations] ([Address]) WHERE [Address] IS NOT
NULL;
GO

CREATE INDEX [IX_OrderHistory_FromId] ON [OrderHistory] ([FromId]);
GO

CREATE INDEX [IX_OrderHistory_StatusId] ON [OrderHistory] ([StatusId]);
GO

CREATE INDEX [IX_OrderHistory_ToId] ON [OrderHistory] ([ToId]);
GO
```

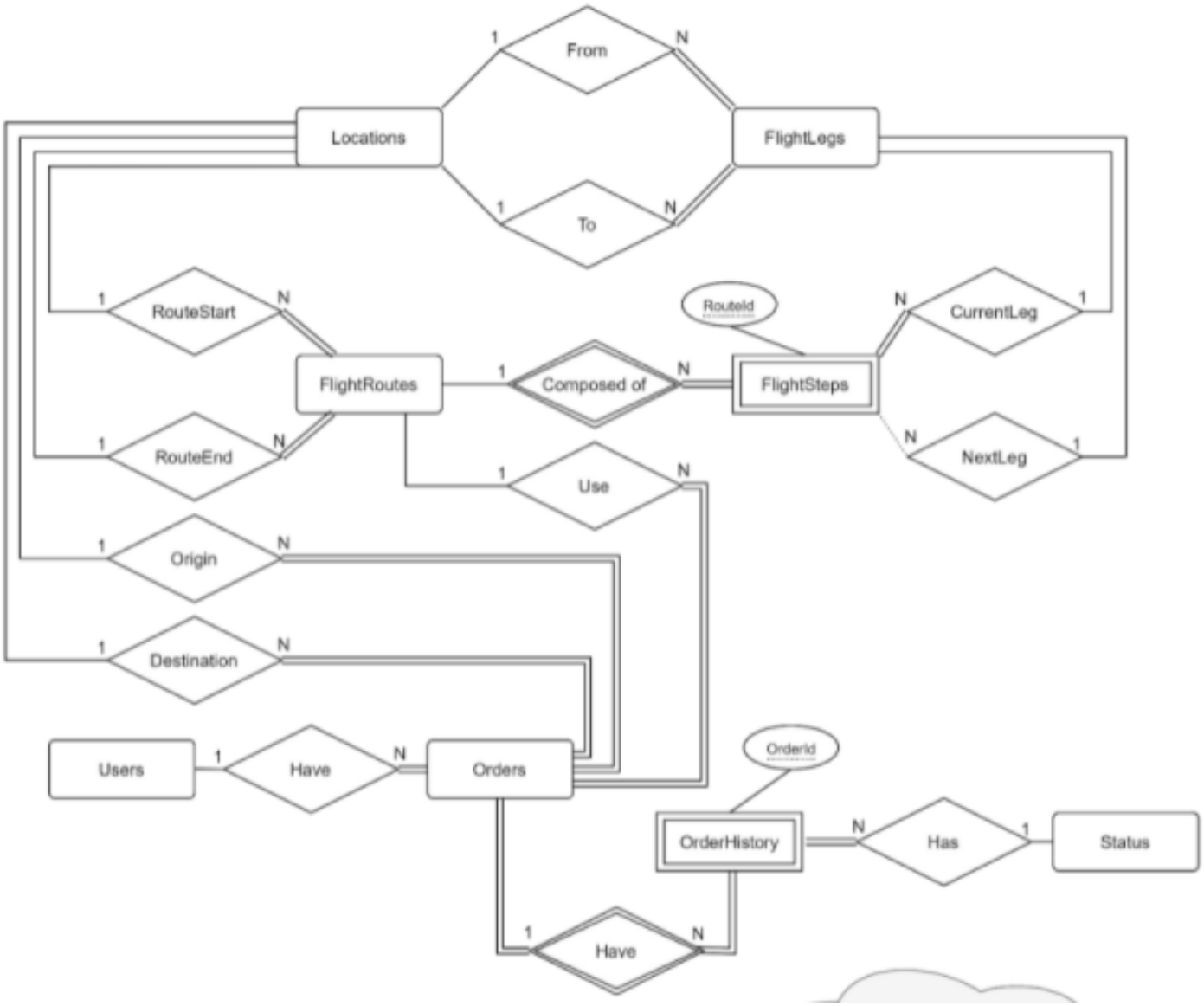
```
CREATE INDEX [IX_Orders_DestinationId] ON [Orders] ([DestinationId]);
GO

CREATE INDEX [IX_Orders_FlightRouteId] ON [Orders] ([FlightRouteId]);
GO

CREATE INDEX [IX_Orders_OriginId] ON [Orders] ([OriginId]);
GO

CREATE INDEX [IX_Orders_UserId] ON [Orders] ([UserId]);
GO

CREATE UNIQUE INDEX [IX_Users_Username] ON [Users] ([Username]);
GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20211202045809_initial', N'6.0.0');
GO

COMMIT;
GO

BEGIN TRANSACTION;
GO

USE WeDrone
GO

CREATE VIEW vw_ShowAllOrders AS
-- View 1: Computes a join of at least three tables
-- This view gets all orders in the WeDrone system, who ordered them, their
-- current status, and their origin/destination
-- and presents it in a user readable manner.

SELECT o.OrderId AS 'Order Id', u.Name AS
'Ordered By',
lorigin.Address AS 'Package Pick-up',
ldestination.Address AS 'Package Destination', s.Name AS 'Current Status',
oh.ValidFrom AS 'Last Update',
o.OrderCreated AS 'Ordered On'

FROM Orders o
INNER JOIN Users u ON o.UserId = u.UserId
INNER JOIN OrderHistory oh ON oh.OrderId = o.OrderId
INNER JOIN Status s ON s.StatusId = oh.StatusId
INNER JOIN Locations lorigin ON lorigin.LocationId = o.OriginId
INNER JOIN Locations ldestination ON ldestination.LocationId = o.DestinationId
WHERE oh.ValidFrom < GETDATE() AND oh.ValidTo > GETDATE()
GO

CREATE VIEW vw_OrdersWithDistanceNotCancelled AS

-- View 2: Uses nested queries with the ANY or ALL operator and uses a GROUP BY clause
-- Returns all orders that were not cancelled and also provides the aggregate sum of the
distance travelled in each order.

SELECT o.*, od.Distance
FROM Orders o
INNER JOIN (
       SELECT OrderId, SUM(distance) AS 'Distance'
```

```
        FROM OrderHistory oh
        GROUP BY OrderId) AS od ON od.OrderId = o.OrderId
WHERE NOT (o.OrderId = ANY(SELECT DISTINCT OrderId FROM OrderHistory WHERE
OrderHistory.StatusId = 6));
GO

CREATE VIEW vw_CustomersWithFilledOrders AS
-- View 3: A correlated nested query
-- Returns user records for all users that have created orders that have been delivered.

SELECT * FROM Users u
WHERE EXISTS (
        SELECT UserId FROM Orders o
        WHERE o.UserId = u.UserId AND o.OrderFilled IS NOT NULL
);
GO

CREATE VIEW vw_AllUsersAndTheirOrders AS
-- View 4: Uses a FULL JOIN
-- Returns all users and all of their associated orders, if they have made any.

SELECT u.*, o.OrderId, o.OriginId, o.DestinationId, o.Weight, o.Volume, o.OrderCreated,
o.OrderFilled
FROM Users u
FULL JOIN Orders o ON o.UserId = u.UserId;
GO

CREATE VIEW vw_OrdersWithWeightOver10 AS
-- View 6: Returns all orders that are over 10 kilograms in weight.

SELECT OrderId
From Orders o
WHERE o.Weight > 10;
GO

CREATE VIEW vw_OrdersWithVolumeOver1 AS
-- View 7: Returns all orders that are over 1 cubic feet in volume.
SELECT OrderId
From Orders o
WHERE o.Volume > 1;
GO

CREATE VIEW vw_ShowFacilityNodes AS
-- View 8: Returns all location nodes that are marked as "facility nodes".
-- These are the nodes that are fixed and represent where drones are stored
-- The opposite is when DroneFacility = 0, these are pickup/drop off locations
-- Later in the application when more nodes are added, we want to easily be able
-- to find these locations from the stored user locations.
SELECT l.*
From Locations l
WHERE l.IsDroneFacility = 1;
GO

CREATE VIEW vw_FlightLegsLessThan10 AS
-- View 9: Returns all flight legs that are less than 10 km
-- These are the shortest distance routes between nodes
SELECT fl.*, l.Address 'FromAddress', l2.Address 'ToAddress'
From FlightLegs fl
INNER JOIN Locations l on l.LocationId = fl.FromId
INNER JOIN Locations l2 on l2.LocationId = fl.ToId
```

```
WHERE fl.Distance < 10;
GO

CREATE VIEW vw_OrdersDelivered AS
-- View 10: Return all orders that were successfully delivered
SELECT Count (*) 'OrdersDelivered'
From Status s
INNER JOIN OrderHistory oh on oh.StatusId = s.StatusId
WHERE s.Name = 'Delivered'
AND oh.ValidFrom < GETDATE() AND oh.ValidTo > GETDATE();
GO

CREATE PROCEDURE sp_GetFlightPlan
-- View 5: This view was converted to a stored procedure to allow the recursive CTE
-- to be passed in initial values to build from

	@RouteStartId		INT,
	@RouteEndId			INT
AS
BEGIN

	WITH FlightPlanCTE (FlightRouteId, RouteStartId, RouteEndId, CurrentLegId, NextLegId)
	AS
	(
		-- Initial leg
		SELECT fr.FlightRouteId, fr.RouteStartId, fr.RouteEndId, s.CurrentLegId,
s.NextLegId
		FROM FlightRouteSteps s
		INNER JOIN FlightRoutes fr ON fr.FlightRouteId = s.FlightRouteId
		WHERE fr.RouteStartId = @RouteStartId AND fr.RouteEndId = @RouteEndId AND
s.IsInitialLeg = 1

		UNION ALL

		-- Recursive call to next leg
		SELECT fr.FlightRouteId, fr.RouteStartId, fr.RouteEndId, s.CurrentLegId,
s.NextLegId
		FROM FlightRouteSteps s
		INNER JOIN FlightRoutes fr ON fr.FlightRouteId = s.FlightRouteId
		INNER JOIN FlightPlanCTE ON s.CurrentLegId = FlightPlanCTE.NextLegId
		WHERE fr.RouteStartId = @RouteStartId AND fr.RouteEndId = @RouteEndId
	)

	SELECT fl.*
	FROM FlightPlanCTE fp
	INNER JOIN FlightLegs fl ON fp.CurrentLegId = fl.FlightLegId;

END


GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20211202045833_runSQL', N'6.0.0');
GO

COMMIT;
GO
```

Appendix B

Relational Diagram for the WeDrone delivery system database:



Appendix C

Appsettings.json file used to define the parameters for the drone deliveries, such as rate, speed, capacity, distance etc. This is also where different API keys for google maps can be stored and changed.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "WeDroneContext": ""
  },
  "GoogleAPI": {
    "Key": "InsertYourAPIKeyHere"
  },
  "DroneSettings": {
    "MaxVolume": 1.5,  // cubic meters
    "MaxWeight": 50,   // kg
    "MaxDistance": 50, // km
    "Speed": 0.5,        // km/s
    "Rate": 1          // $/km
  }
}
```

Appendix D

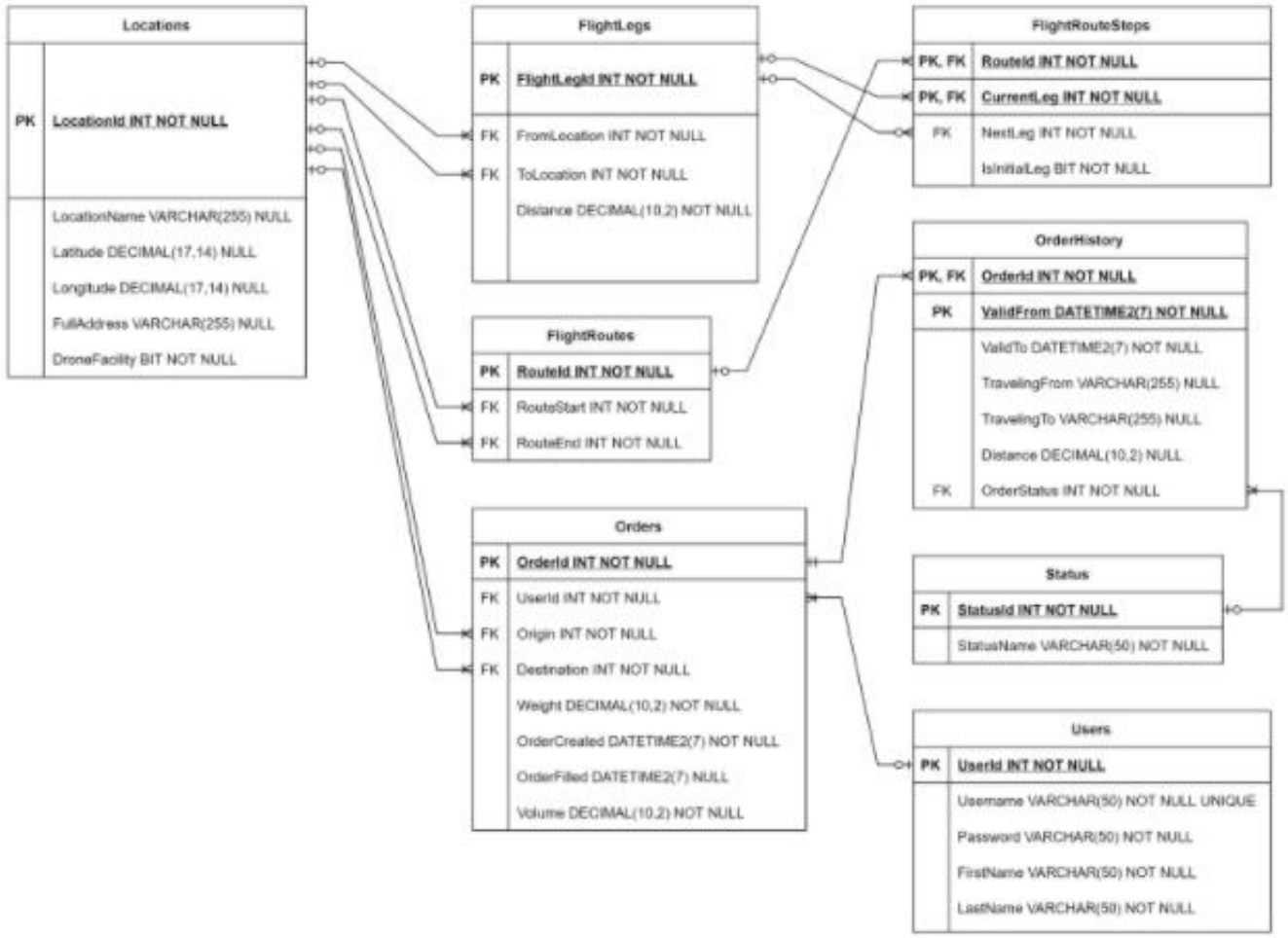
Source code for our test cases using the Google Maps/Autocomplete/Places APIs in ASP.NET C# implementation (found in file APIClientTests.cs)

```
namespace WeDrone.Test.Infrastructure
{
    public class APIClientTests
    {
        private readonly APIClient mockClient;
        private readonly APIClient client;

        public APIClientTests()
        {
            this.mockClient = new APIClient("testkey");
            this.client = new APIClient("realkey");
        }

        [Fact]
        public void GetQueryURI_ShouldReturnURIGivenValidQuery()
        {
            //Arrange
            string query = "101 Bloor Street";

            //Act
            string actual = this.mockClient.GetAddressQueryURI(query, false);

            //Assert
            string expected =
"https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=101%20Bloor%20S
treet&inputtype=textquery&fields=formatted_address,place_id&key=testkey";
            actual.Should().Be(expected);
```

```
        }

        [Fact]
        public void GetQueryURI_ShouldReturnURIAfterTrimmingLeadingAndTrailingSpaces()
        {
            //Arrange
            string query = " 101 Bloor Street ";

            //Act
            string actual = this.mockClient.GetAddressQueryURI(query, false);

            //Assert
            string expected =
"https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=101%20Bloor%20S
treet&inputtype=textquery&fields=formatted_address,place_id&key=testkey";
            actual.Should().Be(expected);
        }

        [Fact]
        public void GetQueryURI_ShouldReturnNullGivenEmptyString()
        {
            //Arrange
            string query = "   ";

            //Act
            string actual = this.mockClient.GetAddressQueryURI(query, false);

            //Assert
            actual.Should().BeNull("because the query is empty");
        }

        [Fact]
        public void GetAddresses_ShouldReturnNullGivenEmptyString()
        {
            //Arrange
            string query = "   ";

            //Act
            string actual = this.mockClient.GetAddressQueryURI(query, false);

            //Assert
            actual.Should().BeNull("because the query is empty");
        }

        [Fact]
        public async void GetAddresses_ShouldReturnValidAddressResponse()
        {
            //Arrange
            string query = " 101 Bloor Street East ";

            //Act
            var actual = await client.GetAddress(query, false);

            //Assert
            //actual.candidates.First().formatted_address.Should().Be("101 Bloor St E,
Oshawa, ON L1H 3M3, Canada");
        }
    }
}
```

Appendix E

E-R Schema Diagram showing tables and key relationships



Contribution Matrix

Link to GitHub WeDrone Project KanBan board with all tasks (To Do, In Progress, Completed):

https://github.com/users/danielgrewal/projects/2

(Note that all tasks are now completed)

Tables with contributions for each Phase from each member:

Project Title	WeDrone Delivery System Application
Group Number	Group 24
Team Leader	Daniel Grewal 100768376
Date	Nov 29, 2021

Milestone 1: Oct 14: Phase 1: Project Proposal

Group Member Name	**Student #**	**% Contributed**
Daniel Grewal	100768376	**40**
Usman Mahmood	100349839	**40**
Karanvir Bhogal	100748973	**10**
Mohammed Adnan Hashmi	100753115	**10**
TOTAL %		**100**

Milestone 2: Nov 7: Phase 2: Project Design

Group Member Name	**Student #**	**% Contributed**
Daniel Grewal	100768376	**20**
Usman Mahmood	100349839	**80**
Karanvir Bhogal	100748973	**0**
Mohammed Adnan Hashmi	100753115	**0**
TOTAL %		**100**

Milestone 3: Nov 26 Phase 3: Final Report and PowerPoint Presentation

Group Member Name	Student #	% Contributed
Daniel Grewal	100768376	**30**
Usman Mahmood	100349839	**10**
Karanvir Bhogal	100748973	**30**
Mohammed Adnan Hashmi	100753115	**30**
TOTAL %		**100**

Milestone 4: Nov 29: Phase 3: Implementation of Design and Project Code

Group Member Name	Student #	% Contributed
Daniel Grewal	100768376	**25**
Usman Mahmood	100349839	**75**
Karanvir Bhogal	100748973	**0**
Mohammed Adnan Hashmi	100753115	**0**
TOTAL %		**100**

Note, work was assigned to members based on availability, experience and confidence in the skill set required to complete work.