OntarioTech UNIVERSITY

WeDrone Delivery System Application

SOFE3650 – Final Project Report

December 5, 2021

Final Project Group 30 Members

Usman Mahmood 100349839

Karanvir Bhogal 100748973

Daniel Grewal 100768376

Mohammed Adnan Hashmi 100753115

Table of Contents

1.1 System Requirements

1.1.1 Use Case Model

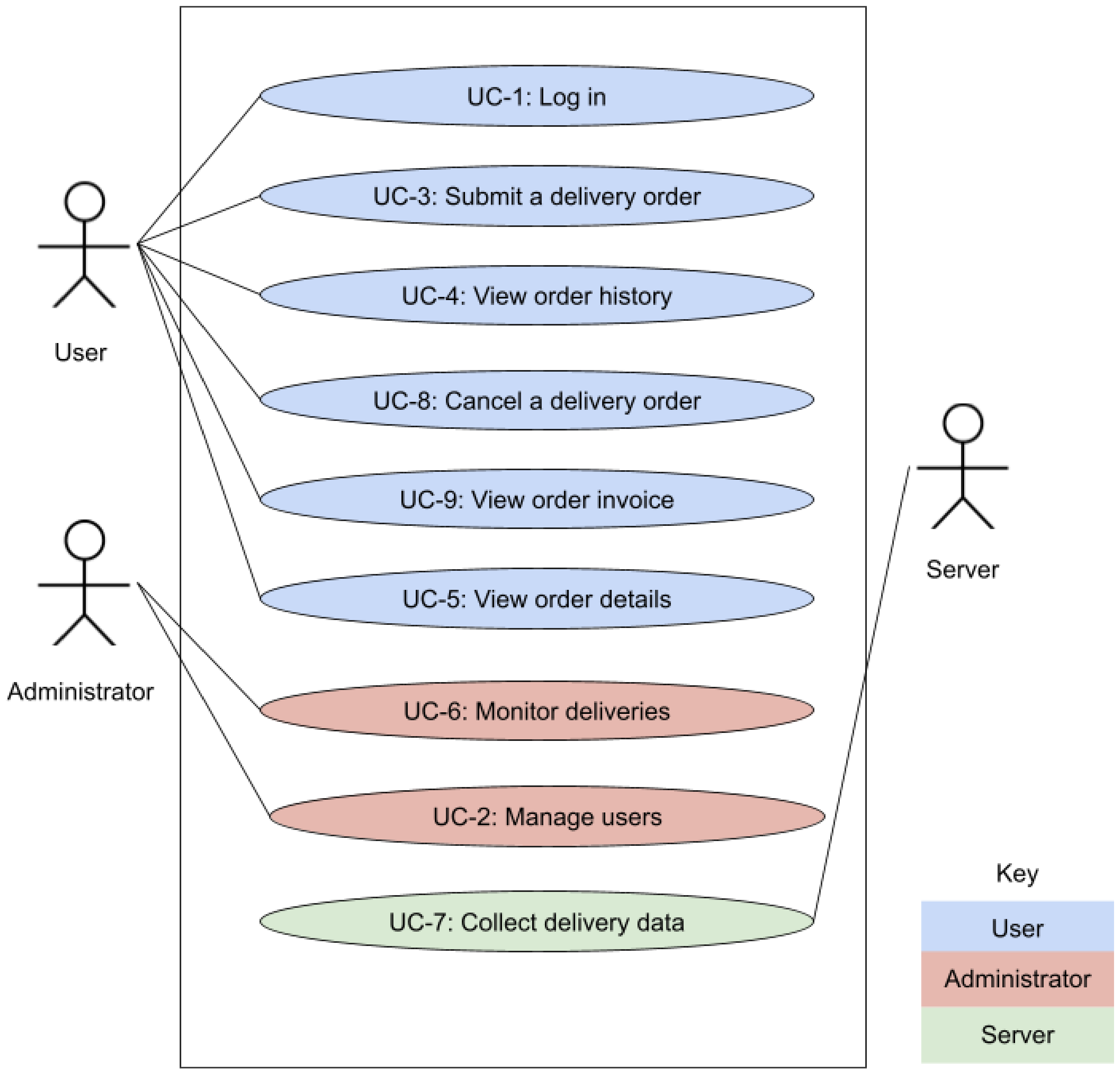


Figure 1: Use case model diagram for the WeDrone Delivery system application

1.1.2 Use Cases

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 an order summary page.
UC-2: Manage users	An administrator adds or removes a user or modifies permissions for a user (e.g. convert a user into an administrator or vice versa).
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/dropoff 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-5: View order details	An user views the details for a specific order (cost, tracking, status, etc.)
UC-6: Monitor deliveries	An administrator can view the deliveries in progress.
UC-7: Collect delivery data	The system will collect and store delivery order data into a database.
UC-8: Cancel a delivery order	A user can cancel their delivery order.
UC-9: View order invoice	A user can enter their order ID or click through from the order history page to see the invoice for that order.

1.1.3 Quality Attribute Scenarios

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 in the last 24 hours (both for administrator and user views).	UC-4,5,6,9
QA-4	Security	All logins (user and administrator) are recorded with a timestamp and IP address.	UC-1,2
QA-5	Scalability	The system is able to respond to an increase attendance to the website	ALL
QA-6	Usability	The application will have an intuitive and easy to use UI.	UC-3,4,5,6,8, 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, 8, 9

1.1.4 Constraints

ID	Constraint
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 in volume.
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.
CON-6	An existing relational database server must be used.

1.1.5 Architectural Concerns

ID	Concern
CRN-1	Establishing an overall initial system structure for the web application.
CRN-2	Leverage our team’s understanding of the MVC framework for web application development, including Microsoft SQL Server, ASP.NET Core and HTML/CSS.
CRN-3	Allocate work to members of the development team and ensure deadlines are met.
CRN-4	Ensure the application is reliable and conforms to all business requirements.
CRN-5	Ensure the database performs all operations and queries as required by the application and provides the correct and proper data.

1.2 ADD Iteration 1: Establishing an Overall System Architecture

1.2.1 Review Inputs

Category	Details
Design Purpose	The WeDrone delivery application is a greenfield system in a mature domain. The purpose is to produce a detailed design supporting the development of a proof-of-concept application demonstrating a possible architecture to model this domain.
Primary Functional Requirements	From the Use cases presented, the primary ones were determined to be: UC-3: It directly supports the core of the business UC-5: It directly supports the core of the business UC-8: It directly supports the core of the business
Quality Attribute Scenarios	The scenarios as described in the progress report are prioritized as follows:

Scenario ID	Importance to customer	Difficulty of Implementation According to the Architect
QA-1	High	Medium
QA-2	High	Medium
QA-3	High	High
QA-4	High	Low
QA-5	Medium	Medium
QA-6	High	High
QA-7	Medium	Medium

	From this list, QA-1, QA-2, QA-6, QA-7 are selected as drivers.
Constraints	All the constraints are included as drivers
Architectural Concerns	All the architectural concerns are included as drivers

1.2.2 Establish Iteration Goals by Selecting Drivers

The goal of the first iteration is to establish an overall initial system structure, as outlined in CRN-1. The architect must consider the following for the general structure of the system:

- QA-1: Availability
- QA-2: Performance
- QA-6: Usability
- QA-7: Interoperability
- CON-1: The system must be accessed using a web browser
- CON-4: Users must provide valid addresses for pickup and drop off (validation done by the external map API)
- CON-5: There must be an established internet connection with reliable bandwidth during selection and confirmation
- CON-8: A relational database server must be used
- CRN-2: Leverage understanding of web application development using Microsoft SQL and ASP.NET core framework

1.2.3 Choose One or More Elements of the System to Refine

In our situation, the entire system except for the Drone Controller needs to be refined. This is done by the decomposition. The scope of our application does not handle how the drones themselves operate.

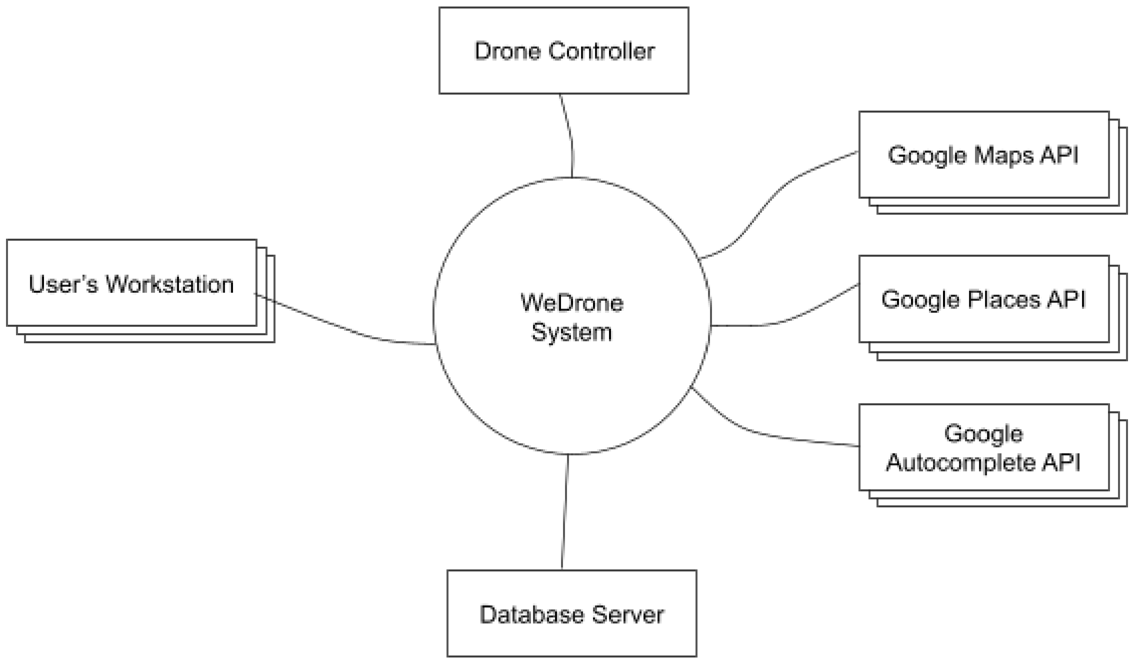


Figure 2: Context diagram for the WeDrone delivery system

1.2.4 Choose One or More Design Concepts That Satisfy the Selected Driver

Design Decisions and Location	Rationale
Logically structure the application to use the **Web Applications** reference architecture. Both the client and server part of the system encompass the web applications architecture in our project scope.	The bulk of the application resides on the server and is composed of a layered framework: presentation, business and data layers. The ASP.NET core framework is naturally adapted to the web applications reference architecture.
	Discarded Alternatives:

Alternative	Reason for Discarding
Rich Client Application	The Rich Client Application reference architecture supports the development of applications that are installed on the user's PC. This architecture was discarded because it does not support applications which run on the user's web browser.
Rich Internet Application	The Rich Internet Application reference architecture is oriented towards the development of applications with a rich user interface that run on a web browser. This architecture was discarded because our application is more oriented towards the functionality of the service rather than the richness of the user interface.
Mobile Application	The Mobile Application reference architecture is discarded due to the fact that it is developed around handheld devices. Our application is a web based application and does not

	incorporate the use of a separate device.
Physically structure the application using the **three-tier deployment pattern**	Since the system is accessed from a web browser (CON-1) and an existing database must also be used (CON-8), a three-tier deployment is the best choice.
Build the user interface of the web application using HTML and CSS components.	Since the application is web based, using UI frameworks that are compatible across various browsers and that are user-friendly are paramount to the user experience (QA-6: Usability).
Deploy the application using the Microsoft Azure publishing process	ASP.NET Core application will be published using Azure, as this is preferred for the Microsoft application development stack.

1.2.5 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Design Decision and Location	Rationale
Create a module dedicated to interacting with the external API to get the geocode information needed to calculate distances between the nodes in our application.	Abstracting the logic related to interacting with the external API allows for the ability to change API services or use a different method for gathering geocode information, if need be, later in the WeDrone application lifecycle.
Create a module dedicated to interacting with the database.	Abstracting the logic related to using the database enables the ability to change our database if needed (e.g. switching from Microsoft SQL to MySQL server etc.)

1.2.6 Sketch Views and Record Design Decisions

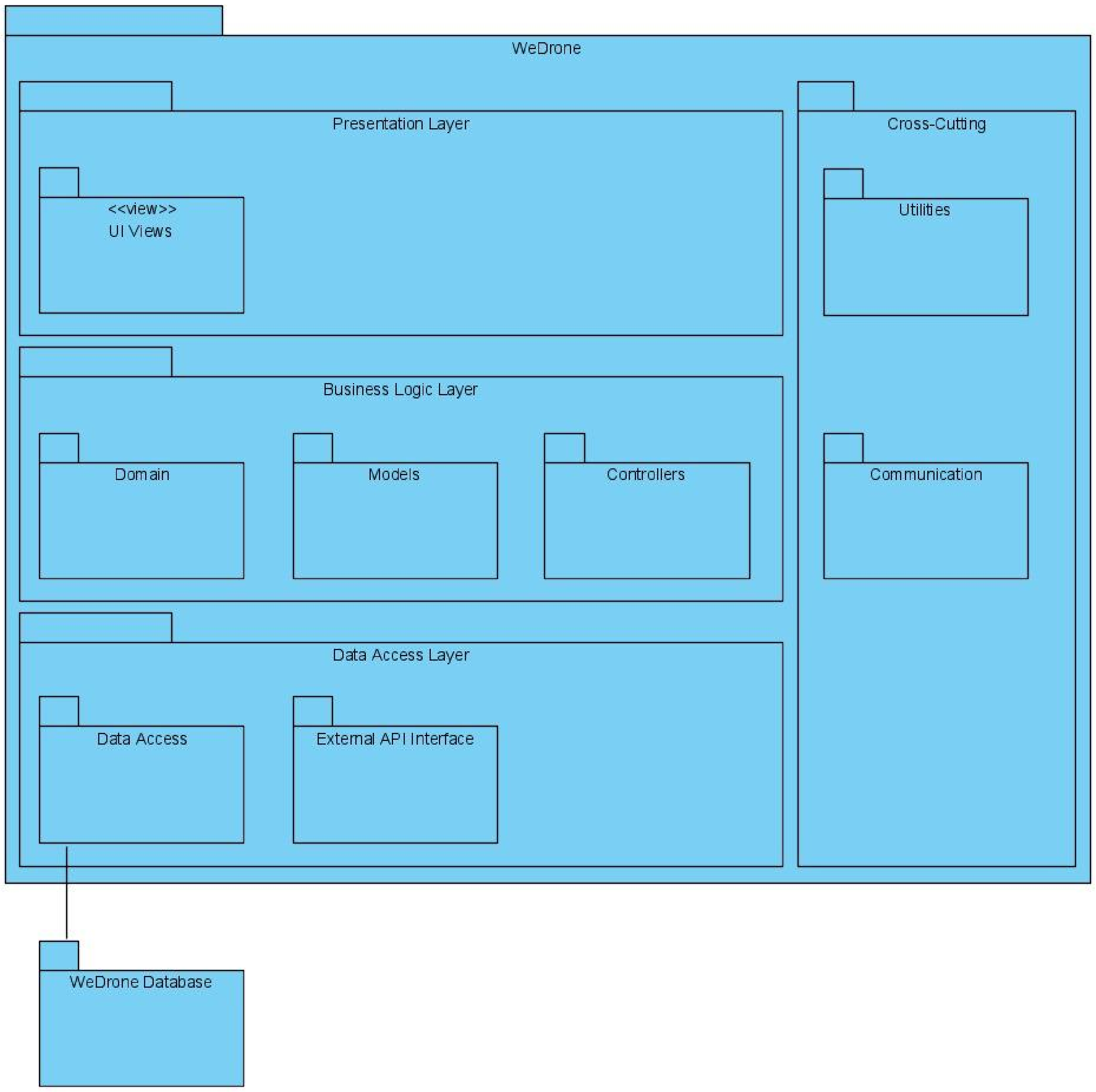


Figure 3: Package diagram of modules for our reference architecture: Web Applications

The responsibilities of the modules displayed above in Figure 3 are summarized in the table below:

Element	Responsibility
Presentation Layer	Contains the components needed to interact with the user (get input and display information)
UI Views	Contains components necessary to create a user interface in the browser
Business Logic Layer	Contains the components that are required for the main functionality of the application
Domain	Contains all the domain classes
Models	Contains all the model classes
Controllers	Contains all the controller classes
Data Access	Contains components necessary to store and retrieve information from the database
External API Interface	Contains components required to interact with external API services
WeDrone Database	The WeDrone Database is the Microsoft SQL Server database, residing on a separate server
Cross-Cutting Layer	Contains components that are addressed by the overall system and impact several layers and functionality of the system
Utilities	The utilities classes offer convenience functions that may be used throughout the application.
Communication	Contains components necessary to handle changes in data and external systems (service agents)

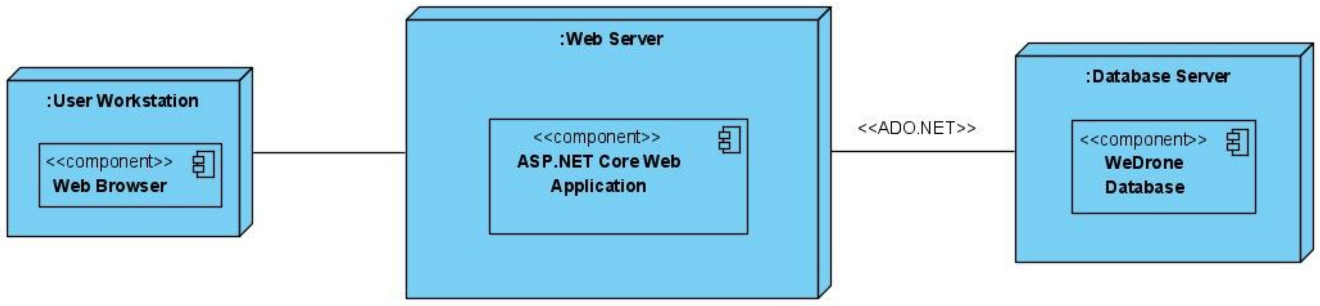


Figure 4: Initial deployment diagram for the WeDrone delivery system application

The responsibilities of the elements displayed above in Figure 4 are summarized in the table below:

Element	Responsibility
User Workstation	The user's web browser (Client) that handles the client-side logic of the web application and connects to the server.
Web Server	ASP.NET Core Web Application (Server) that handles requests and the core of the application on the server (business logic)
Database Server	Microsoft SQL server that handles the WeDrone database

Relationship between elements in the deployment model are in the following table:

Relationship	Description
Relationship between client workstation and application	The client communicates via https with the web server which serves the WeDrone application.
Relationship between web server and database server	The web application communicates with the database via the ADO.NET protocol.

1.2.7 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
	UC-3		Selected reference architecture (*Web Applications*) establishes the modules that will support this functionality.
	UC-5		Selected reference architecture (*Web Applications*) establishes the modules that will support this functionality.
	UC-8		Selected reference architecture (*Web Applications*) establishes the modules that will support this functionality.
	QA-1		Some relevant decisions made: application is web based and will be hosted by a reliable hosting service to ensure 24/7 uptime. However, cloud implementation for hosting MS SQL Server is in review. (possibly Azure?)
QA-2			No relevant decisions made: web hosting and loading time performance yet to be considered in implementation.
	QA-6		Use of common UI web components and design features will be implemented for simple UI. Details on CSS Framework and page layouts still in progress.
	QA-7		Some relevant decisions made with how APIs and other components will be incorporated into our application.
		CON-1	*Web Applications* architecture chosen, which is specific for applications that are accessed through a web browser. The ASP.NET core application we are building will be accessible through any modern web browser.
	CON-4		Use of the external API to get real world validated data for user input addresses and conversion into geocode data for accurate distance calculations.
CON-5			No relevant decisions made as of yet.
		CON-6	Use of Microsoft SQL Server has been chosen and implemented as our RDBMS. Database schema and

			data entity model is complete.
	CRN-2		Leverage understanding of web application development using Microsoft SQL and ASP.NET core framework.

2.1 ADD Iteration 2: Identifying Structures to Support Primary Functionality

The goal of this iteration is to address the general architectural concern of identifying structures to support primary functionality.

2.1.1 Establish Iteration Goal by Selecting Drivers

The main use cases that are selected as design drivers are as follows:

- UC-3: Submit a delivery order
- UC-4: View order history
- UC-8: Cancel a delivery order

2.1.2 Choose One or More Elements of the System to Refine

All components necessary to create a delivery order and to view the order details need to be refined. This will include the necessary application logic (business layer) and database views (data layer) components to make queries and represent the data to the user.

2.1.3 Choose One or More Design Concepts That Satisfy the Selected Drivers

Design Decisions and Location	Rationale and Assumptions
Create a **Domain model** for the application	A domain model is necessary to have before starting the functional decomposition. It helps identify the major entities in the domain and their relationships. There aren't any alternatives to the domain model, as using a different model would lead to an ad hoc architecture which is difficult to understand. Having the domain model is also good with identifying relationships with major components. For our system, it is important for the relationship between the presentation and business layer to be identified as this will allow the user's account system to have a smooth transition when looking at previous orders and cancelling an order.
Identify **Domain Objects** that map to functional requirements	Every functional element within the application needs to be included in a self contained building block or a domain object.
Decompose **Domain Objects** into general and specialized **Components**	The modules associated with the layers in which they are located are the finer-grained elements that support the functionality of the domain object.
Use ASP.NET Core with Entity Framework (ORM)	ASP.NET is a popular web-development framework for building web apps on the .NET platform. Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. Other alternatives were not considered because we had initially decided to implement the Microsoft stack for our application development, including Microsoft SQL for our database. The development team is already familiar and capable of using these frameworks in conjunction with each other (CRN-2)

2.1.4 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Design Decision and Location	Rationale
Create only an initial domain model.	An initial domain model must be made to accelerate the design phase entities such as the order history, confirmation of delivery, and cancelation modules that participate in the primary use cases which need to be identified and modelled.
Map the system use cases to domain objects	Analyzing the system's use cases can help us identify the domain objects.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface	Decomposing the domain objects ensures the modules that support all of the functionalities are identified. This is only used for the primary use cases to allow another team member to identify the rest of the modules, helping divide work among the team.
Connect components associated with modules using ASP.NET Core	The current frameworks utilize inversion control to allow different aspects to be supported alongside the modules being able to be unit-tested.
Associate frameworks with a module in the data layer using Entity Framework ORM	ORM mapping is encapsulated in the modules that are contained in the data layer. The Entity Framework in ASP.NET core is associated with these modules.

2.1.5 Sketch Views and Record Design Decisions

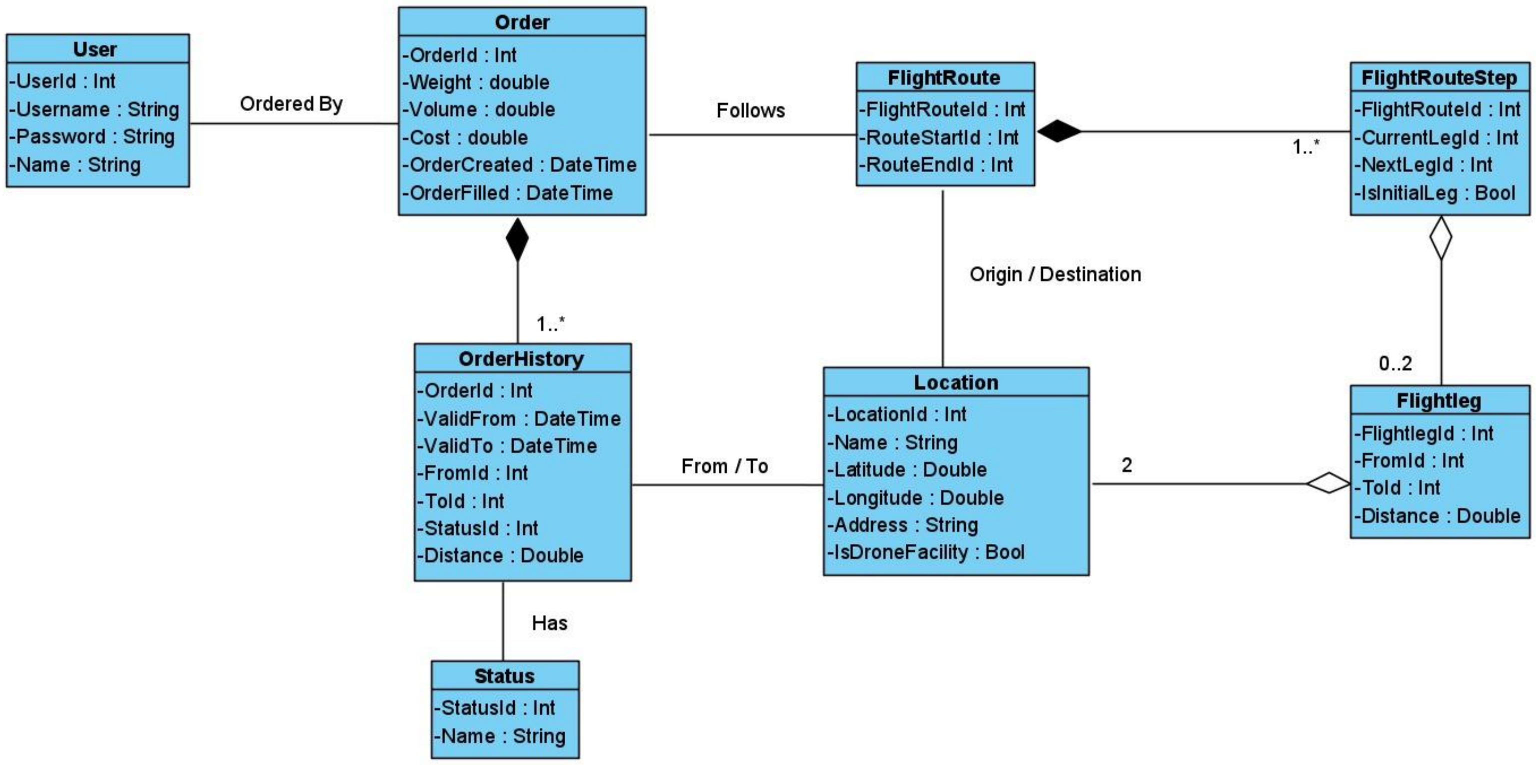


Figure 5: Initial domain model for the WeDrone delivery system application

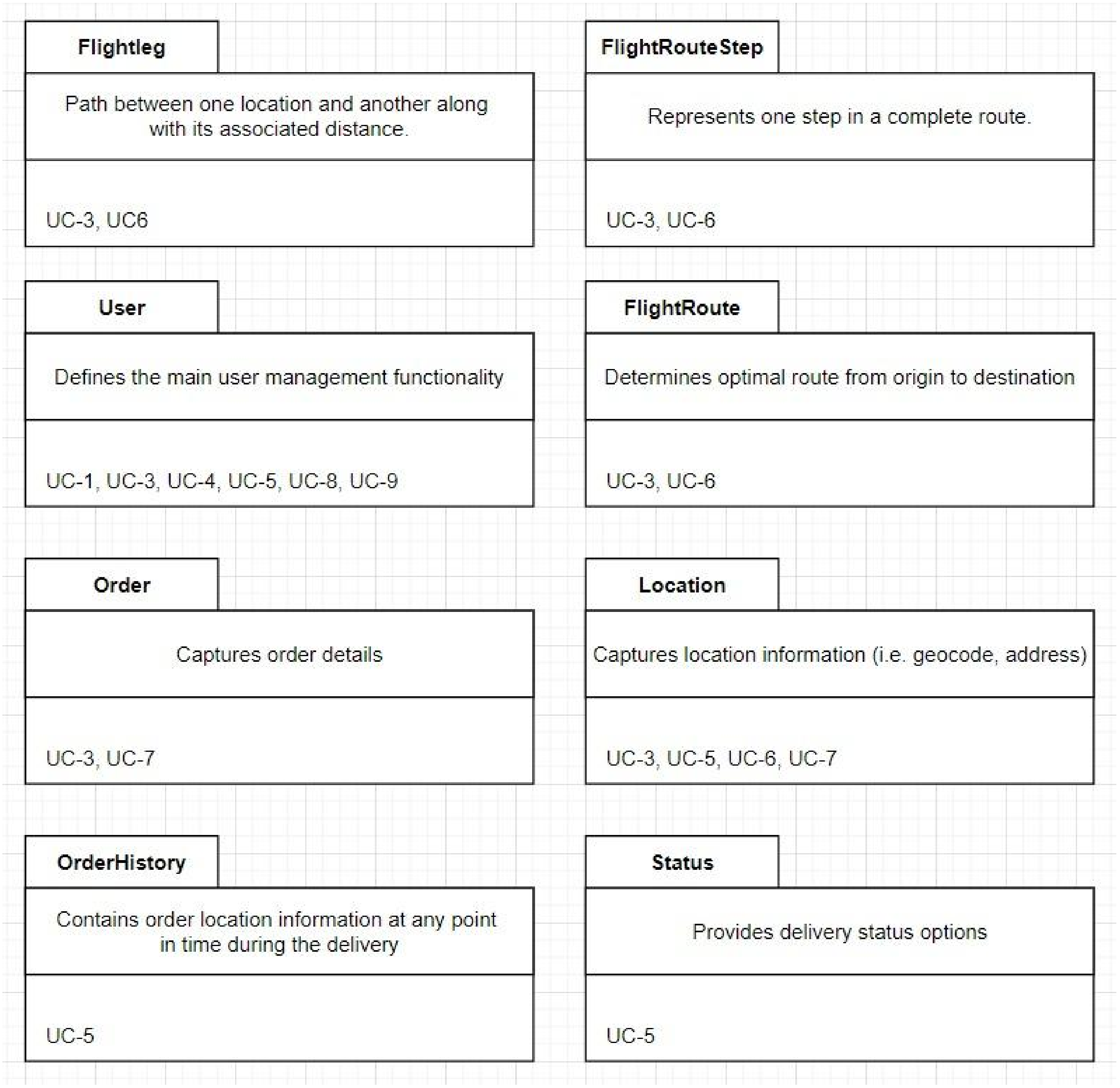

Flightleg
Path between one location and another along with its associated distance.
UC-3, UC6
FlightRouteStep
Represents one step in a complete route.
UC-3, UC-6
User
Defines the main user management functionality
UC-1, UC-3, UC-4, UC-5, UC-8, UC-9
FlightRoute
Determines optimal route from origin to destination
UC-3, UC-6
Order
Captures order details
UC-3, UC-7
Location
Captures location information (i.e. geocode, address)
UC-3, UC-5, UC-6, UC-7
OrderHistory
Contains order location information at any point in time during the delivery
UC-5
Status
Provides delivery status options
UC-5

Figure 6: Domain object model and associated use cases

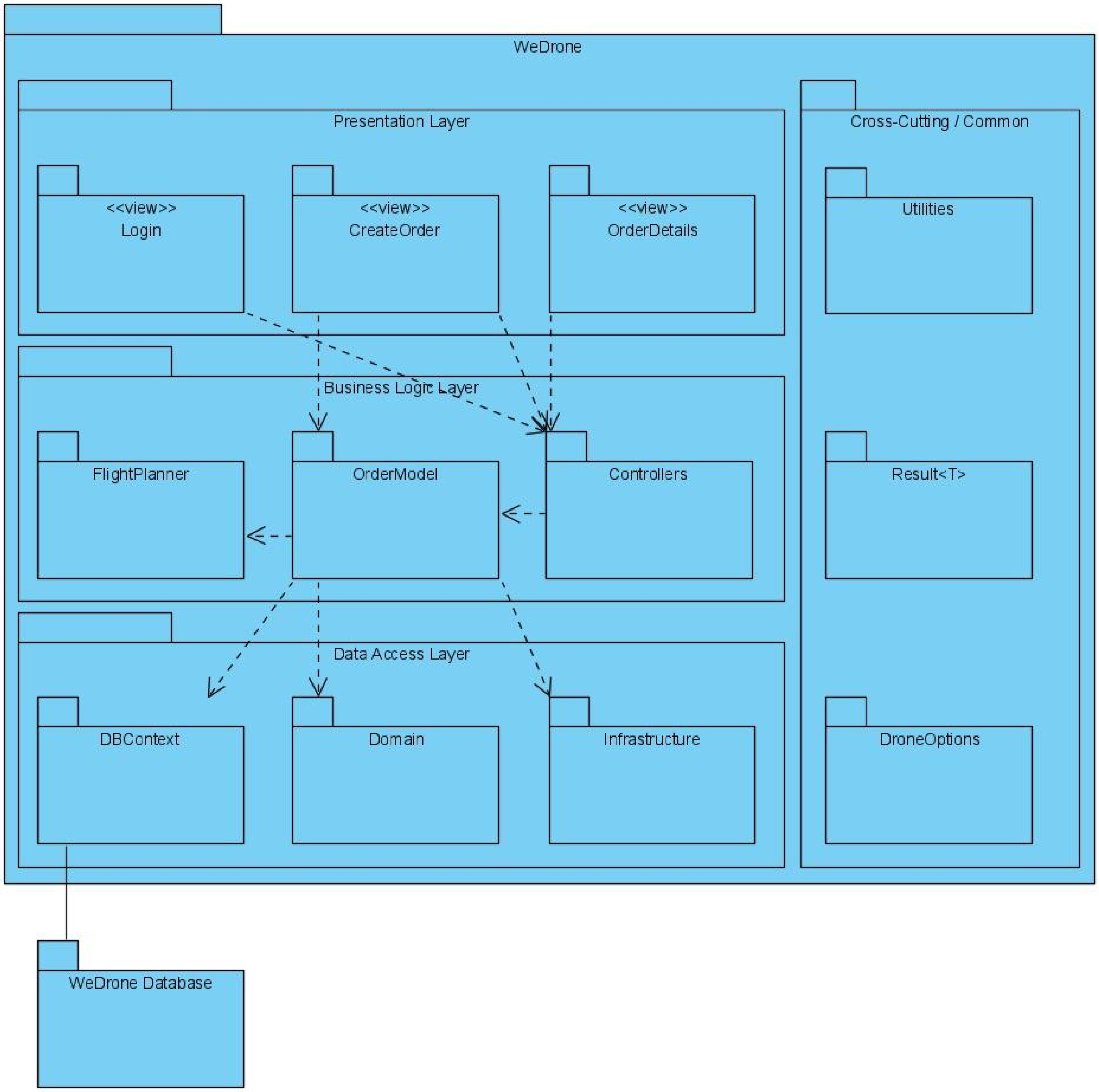


Figure 7: Package diagram of modules for our primary use cases

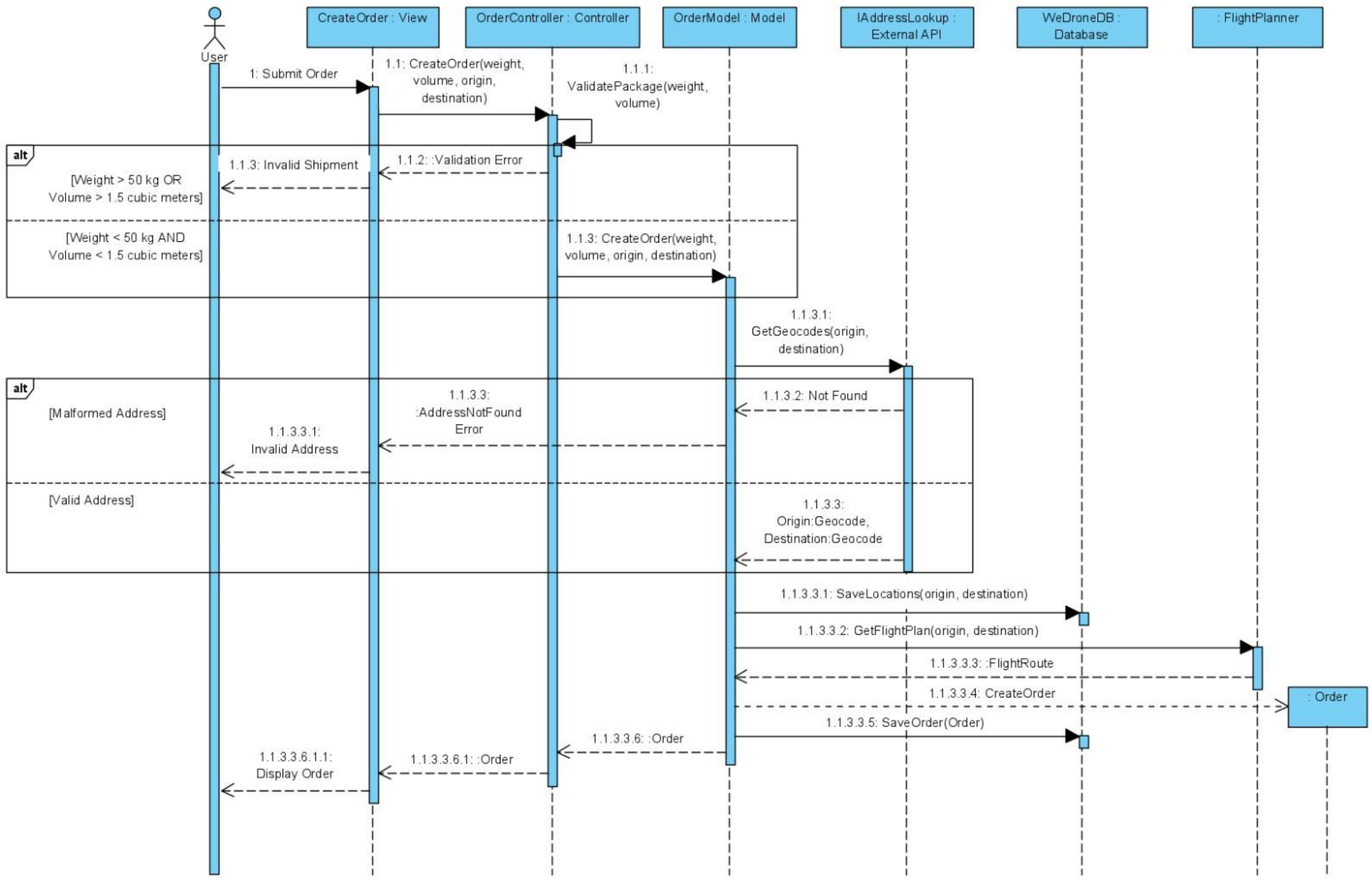
The responsibilities of the modules displayed above in Figure 7 are summarized in the table below:

Element	Responsibility
Login View	Login page of the WeDrone application that takes a username and password to authenticate the user. This supports the use case UC-1.
CreateOrder	The create order page allows the user to provide package details along with the origin and destination addresses for the proposed delivery. This supports UC-3.
OrderDetails	The OrderDetails view allows the user to track the status of their delivery order. This includes the drone’s current flight leg as well as the current order status. This supports UC-5.
FlightPlanner	The FlightPlanner package is responsible for determining the flight path for the drone to deliver the package from the origin to the destination. The FlightPlanner will prepare a FlightRoute that will be associated with the order and followed step by step until the delivery is completed.
OrderModel	The OrderModel is responsible for creating the order and saving it to the database. Once an order is created, the user can be notified of the order details.
Controllers	The Controllers package contains the OrderController and the AccountController. The AccountController is responsible for authentication, and the OrderController is responsible for validating input from the CreateOrder view and communicating with the OrderModel.
DBContext	The DBContext class is required by the object relational mapper (ORM) Entity Framework which serves as the interface to communicate with the WeDrone database.
Domain	The domain package contains all of the domain classes of the WeDrone application. This includes classes that represent the order, location, flight legs, etc.
Infrastructure	The infrastructure package contains classes that communicate with the external APIs and provides the *IAddressLookup* interface to provide the application with the capability to query addresses.
WeDrone Database	The WeDrone Database is the Microsoft SQL Server database, residing on a separate server.
Utilities	The utilities classes offer convenience functions that may be used throughout the application.

Result<T>	The Result class acts as a wrapper for any class generically represented as *T*. It will wrap any response with a success or failure notification along with an associated failure message if applicable. This provides a common response type that can be used to determine if the success or failure path should be followed.
DroneOptions	The DroneOptions package represents all of the configuration options for the drone delivery, representing specifics such as drone speed, cost per kilometer, acceptable package dimensions, etc. Using this package, all of the configuration details are stored in one place so only a single change to the configuration is required in the application to reflect it throughout the application.

UC-3: Create Delivery Order

Figure 7 below is the sequence diagram for the main driver use case, UC-3: Create Delivery Order. This interaction begins with the user entering the dimensions and weight of their package and the pickup and drop off addresses (which are validated by the external API during input in real time). Once complete, the user then submits the order. If there is an error with the addresses (malformed or changed after the fact by the user from the returned address using the API) or the package dimensions/weight is not acceptable, an error is displayed and the order page is reset. Assuming the user input is valid, the application will then proceed with saving order data to the database and querying for the optimal flightpath to fulfil the delivery (FlightPlanner).



User
CreateOrder : View
OrderController : Controller
OrderModel : Model
IAddressLookup : External API
WeDroneDB : Database
: FlightPlanner
1: Submit Order
1.1: CreateOrder(weight, volume, origin, destination)
1.1.1: ValidatePackage(weight, volume)
alt
1.1.3: Invalid Shipment
1.1.2: :Validation Error
[Weight > 50 kg OR Volume > 1.5 cubic meters]
[Weight < 50 kg AND Volume < 1.5 cubic meters]
1.1.3: CreateOrder(weight, volume, origin, destination)
1.1.3.1: GetGeocodes(origin, destination)
alt
[Malformed Address]
1.1.3.3: :AddressNotFound Error
1.1.3.2: Not Found
1.1.3.3.1: Invalid Address
[Valid Address]
1.1.3.3: Origin:Geocode, Destination:Geocode
1.1.3.3.1: SaveLocations(origin, destination)
1.1.3.3.2: GetFlightPlan(origin, destination)
1.1.3.3.3: :FlightRoute
1.1.3.3.4: CreateOrder
: Order
1.1.3.3.5: SaveOrder(Order)
1.1.3.3.6: :Order
1.1.3.3.6.1: :Order
1.1.3.3.6.1.1: Display Order

Figure 7: Sequence Diagram for UC-3: Create Delivery Order

UC-5: View Order Details

Figure 8 below is the sequence diagram for UC-5: View Order Details. A user is able to view a table of the orders and retrieve details (cost, length, origin and destination address, etc.) for a specific order based on its OrderID. The URL will have /OrderID in its request to retrieve the specifics from the database for the order.

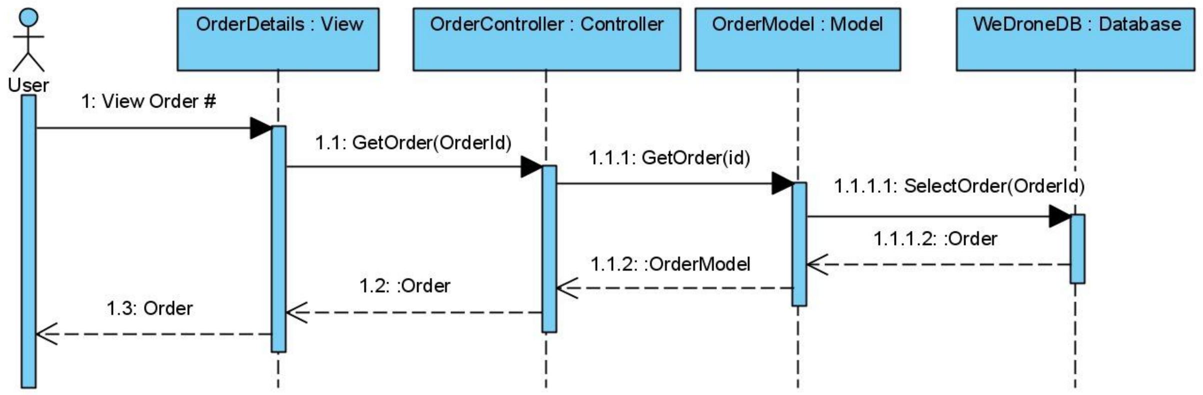


Figure 8: Sequence Diagram for UC-5: View Order Details

UC-8: Cancel a Delivery Order

Figure 9 below is the sequence diagram for UC-8: Cancel a Delivery Order. The OrderID is used to locate the order details in the database and then delete the corresponding record. A confirmation is then displayed for the user. If the OrderID is not found, an error message indicating the order is not found is displayed.

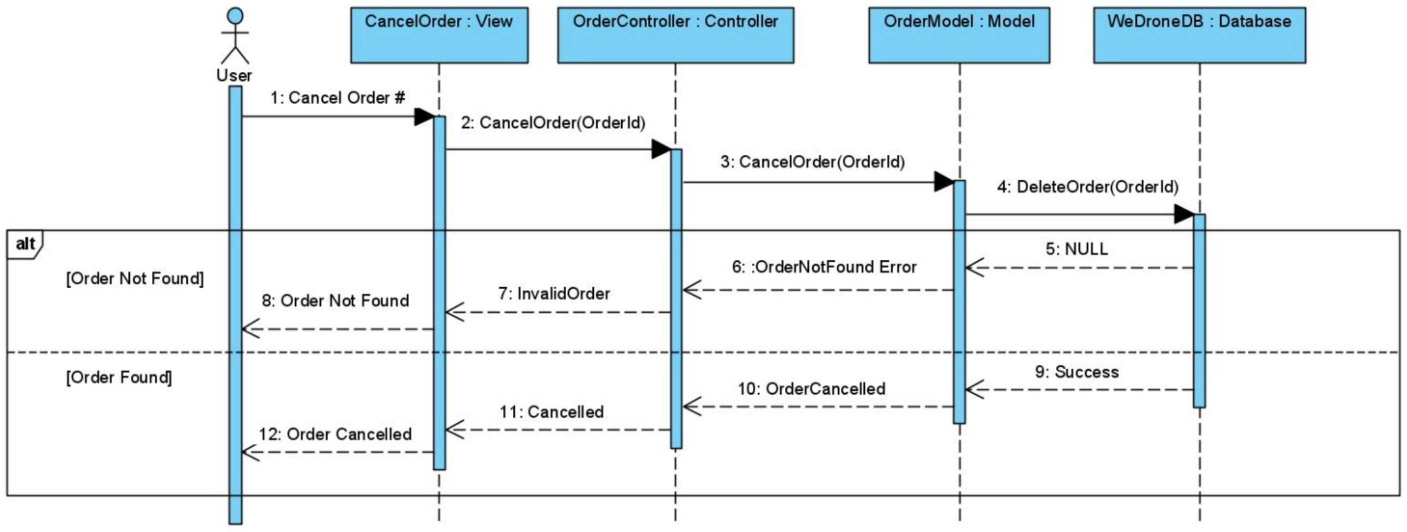


Figure 9: Sequence Diagram for UC-8: Cancel a Delivery Order

2.1.6 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
		UC-3	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-5	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-8	Modules across the layers and preliminary interfaces to support this use case have been identified.
	QA-1		Some relevant decisions made: application is web based and will be hosted by a reliable hosting service to ensure 24/7 uptime. However, cloud implementation for hosting MS SQL Server is in review. (Azure?)
	QA-2		Some relevant decisions made: web hosting and loading time performance should be sufficient using Azure hosting services. We are still investigating other hosting services that will run Microsoft SQL server and .NET framework.
	QA-6		Use of common UI web components and design features will be implemented for simple UI. Considering using .NET Blazor framework for UI along with other CSS components.
	QA-7		Some relevant decisions made with how APIs and other components will be incorporated into our application.
		CON-4	Use of the Google Maps/Places/Autocomplete APIs to get real world validated data for user input addresses and conversion into geocode data for accurate distance calculations.
CON-5			No relevant decisions made as of yet.
	CON-6		Order information for all users is to be stored in the database. Modules for data collecting and storage have been partially identified.
		CRN-2	Leverage understanding of web application development using Microsoft SQL and ASP.NET core framework.
	CRN-3		Allocation of work is still yet to be fully determined.

3.1 ADD Iteration 3: Addressing Quality Attribute Scenario Driver

3.1.1 Establish Iteration Goal by Selecting Drivers

The main purpose of this iteration is to analyze and provide reasoning supporting the architectural decisions for the QA-6 and QA-7 quality attributes.

- QA-6: The application will have an intuitive and easy to use UI.
- QA-7: 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.

3.1.2 Choose One or More Elements of the System to Refine

The application logic for finding the optimal flight path using the route information in the database and implementing the external API services are the components that need to be refined.

3.1.3 Choose One or More Design Concepts That Satisfy the Selected Drivers

The design concepts used in this iteration are the following:

Design Decisions and Location	Rationale and Assumptions
Introduce the **Google Maps/Places API** to retrieve geocode data and validated autocomplete delivery addresses.	By utilizing the Google Maps/Places API, the customer will be able to easily and accurately enter the address to where they want their order delivered. It also ensures that our maps system is always up to date for greater accuracy.
Introduce a **pathfinding algorithm** for sorting all flight paths and determining the most optimal path for the delivery.	The optimal flight path will be determined using a pathfinding algorithm. Implementing this would reduce our energy consumption and delivery time.

3.1.4 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions are summarized in the following table:

Design Decisions and Location	Rationale and Assumptions
Implement autocomplete when the user is entering a desired delivery location.	Using a field with autocomplete makes the process of entering and finding an address much simpler for users. It also decreases the chances of getting an address wrong due to typos.
Implement a pathfinding algorithm in the database.	By having a pathfinding algorithm in the database, we can determine the shortest and most efficient path for deliveries.

3.1.5 Sketch Views and Record Design Decisions

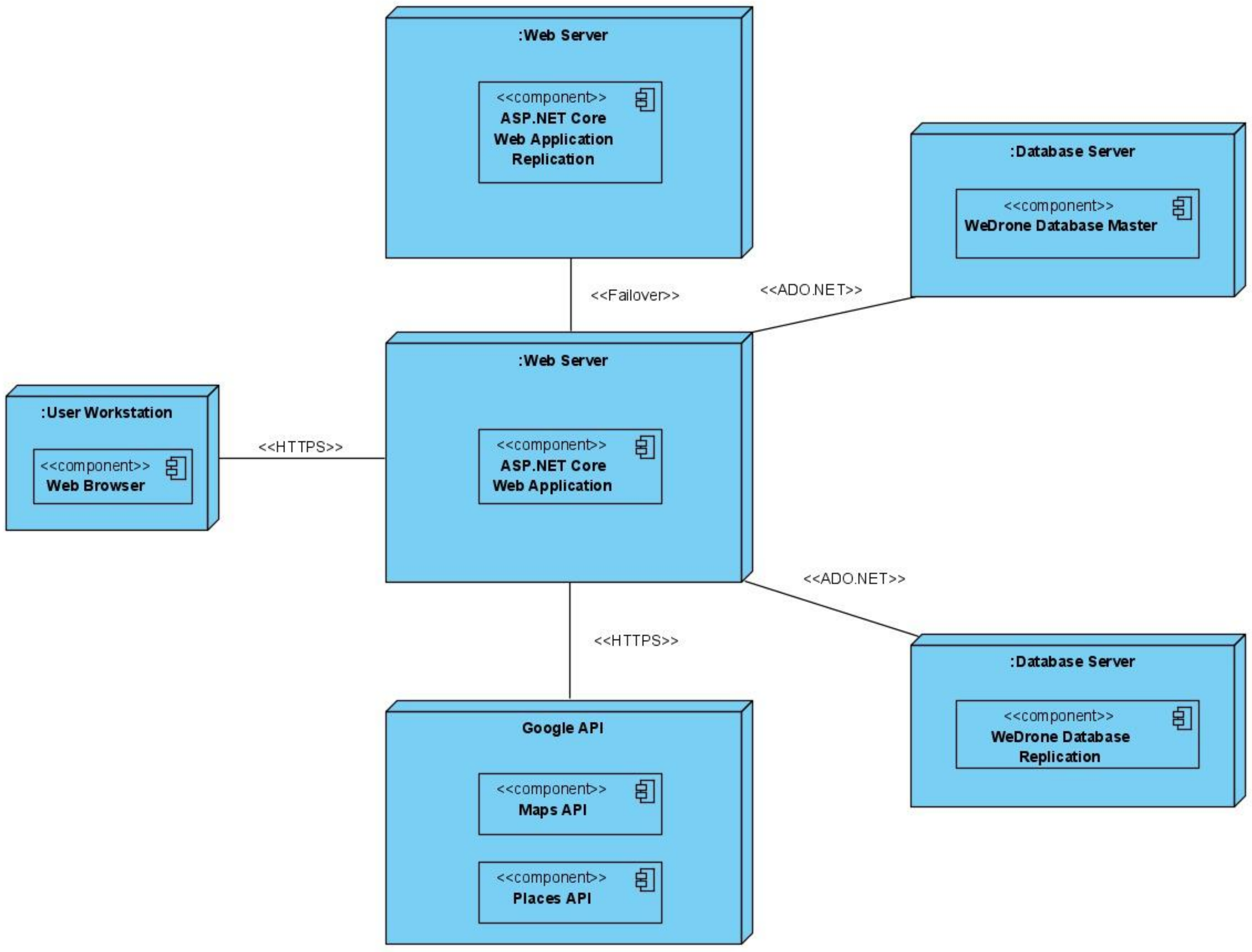


Figure 10: Refined deployment diagram

The responsibilities of the elements displayed above in Figure 10 are summarized in the table below:

Element	Responsibility
User Workstation	The user’s web browser (Client) that handles the client-side logic of the web application and connects to the server
Web Server	ASP.NET Core Web Application (Server) that handles requests and the core of the application on the server (business logic)
Web Server (Replication)	To meet 24/7 availability requirements and maintain integrity over the network a replication of the web server is required
Google API	Module that handles requests for geocode and address data with the Google Maps/Places APIs. Interfaces with the web server application to support primary use cases
Database Server (Master)	Separate server that hosts the Microsoft SQL database
Database Server (Replication)	To meet 24/7 availability requirements and maintain integrity over the network a replication of the database server is required

3.1.6 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
	QA-1		No relevant design decisions made.
	QA-2		No relevant design decisions made.
		QA-6	The introduction of the Google Maps API greatly enhances the usability of the project, reducing the time taken to locate addresses and ensuring geocode data is up to date.
		QA-7	Implementing the pathfinding algorithm in the database, it makes it much easier to determine the most efficient flight path.
CON-5			No relevant design decisions made. (no consideration for an “offline” mode at this time)
	CON-7		No relevant design decisions made.
	CRN-3		No relevant design decisions made.