MS windows mini-project

A general overview

This year's mini-project deals with project design in the industry. Specifically, a shipping and delivery management system for a single project.
Examples: a restaurant that accepts delivery orders and dispatches them, a bookstore that delivers books door-to-door.
We will develop an application that has:

(a) an administrator's interface where we could manage the messengers and the delivery orders
(b) a courier's interface, where delivery tasks can be viewed, handled, and updated.

The manager **opens orders** that are located within the range between the company's **physical address** and the **maximal distance** defined by the company. In addition, the admin can define, for all company orders, a **maximum delivery time window** that the company commits to meet, beyond which an order is considered late. The manager can also define **a risk time window**, within which any order that has not yet been delivered is considered at risk (of losing money).
A courier can select an order to handle from among the open orders. Each courier may handle **only one order at a time**. When a courier chooses to handle an order, it means the order is assigned to them and they will carry out the delivery to completion.
Couriers can report the completion of a delivery according to the delivery completion type. Examples: (a) the order was delivered and is now closed; (b) the customer refused to accept the order, so it was returned to the company and closed; (c) the customer could not be located, and the order was returned to the company and reopened awaiting a new delivery attempt.
A manager can view the full delivery history for monitoring and management purposes. Each courier can view only their own delivery history.
Our system runs a simulation. In order to provide a sense of time progression, we require a system clock that is separate from the computer's real clock. The system administrator will be able to initialize and adjust the system clock. In every operation, the system's current time will be the current value of the system clock, not the computer's real time, (e.g., DateTime.Now).

Our system will be built in stages during the semester under the hierarchical multi-tier architecture. This subject will be covered during the semester.
Bonus add-ons
The optional enhancements will be colored in orange

The details of these add-ons and their corresponding scores can be found in the following regularly updated document: **DotNet2025–Project–Addons**.
Teams may propose additional features beyond the original list. Those will be approved subject to the instructor's discretion. The instructor will evaluate the score of the proposed add-ons and, when applicable, update the add-ons document accordingly.
The add-ons are a significant component of the project grade; therefore, our recommendation is to implement as many of them as possible, and to begin accumulating additions already in the early stages.

Use of AI tools
Please see the syllabus note on this item.

Document corrections
The documents describing the mini-project (the general document and the stage documents) are detailed and content-rich. A significant effort was made so that they could be precise and helpful. Should any errors or contradictions be discovered, please email your instructor with your question, and they will forward it to the coordinators. Corrected sections will appear in **pink** highlight.

NOTE: Naming convention
Please note the difference between the class's mini-project (which is the course you are taking) and the project management system itself. We will hereafter refer to them as "the **mini-project**" and "the **project**" respectively. Also note that there is another item termed "project". This is a member of a "solution" in VS2022. We will refrain from using the VS2022 term "project" in this document.

Mini-Project stages' brief description
The application is developed in stages.

1. The project will be developed in eight stages. The maximum weighted score for all stages fulfilled according to the mandatory requirements is 100 points and will constitute 30% of the final project grade.
2. During the final class, a demo run and an oral defense will be carried out in front of the class, with a score of up to 100 points, constituting 60% of the final project grade.
3. Additionally, up to 15 points may be accrued through the implementation of the optional add-ons (enhancements).

As already stated above, the add-ons consist 15 points and thus are a considerable component of the project grade. Therefore, we recommend implementing as many of them as possible, and accumulating them early on. Including these add-ons, the total score of the three grading components may reach 105 points. However, the maximum final score for the mini-project will not exceed 100.

Please review the eight (8) listed stages of the project, numbered from 0 to 7. For each, there exists a separate document with deep-down instructions and explanations. These will be revealed on moodle as we progress through the semester.

Stage 0 – Orientation
Installations, getting to know the development environment, the local Git repository and the internet-based Git mirror (github.com).
Time to complete: 1 week (5 grading points)

Stage 1 - The Data Access Layer (DAL): Modeling the data

- The data contract (the data entities DO)
- Service Brokerage (DalApi): access interfaces to the data entities (an interface per entity)
- Storage structures
- Access methods to data (CRUD)
- Exceptions
- Console-based user interface for testing the Data Access Layer (the test suite)
- Initial data base

Time to complete: 1 week (10 grading points)

Stage 2 – Improving the Data Layer interface, adding LINQ

- A unified data brokerage by means of a generic interface
- Updates to interfaces so they utilize the generic **IEnumerable<>** objects
- Updates to data retrieval and processing using LINQ (no **foreach** loops)
- Exceptions

Time to complete: 1 week (10 grading points)

Stage 3 - XML data storage format

- Implementation of save-to XML files in the Data Access Layer.

Time to complete: 1 week (10 grading points)

Stage 4 – The multi-tier architecture and the Business Layer (BL)

- Building and integrating two layers of the multi-tier model.
- Styling (design) templates of *Singleton* and *Simple Factory Method* in the DAL and BL.
- Data contracts for the Business Layer (logical entities BO).
- Data brokerage for the Business Layer by means of a generic interface (BlApi).
- Console-based user interface for testing the Business Layer (the test suite)

Time to complete: two weeks (20 grading points)

Stage 5 - A basic graphical interface

- A simple display layer (GUI)

Time to complete: 1 week (10 grading points)

Stage 6 - A fully blown graphical interface

- Completion of the graphical interface
- Updates to the display layer via XAML bindings

Time to complete: 3 weeks (25 grading points)

Stage 7 - sub-processes and simulations

- Working with sub-processes
- An additional interface for simulating GUI actions in the display layer
- Interfacing the simulator into the GUI

Time to complete: two weeks (10 grading points)

Table of contents

General

General system requirements

The project's interface can be used by (a) the "administrator" and (b) a "courier".
The administrator could perform actions in four main categories

- Manage couriers
    - Add or Remove couriers
    - View the list of couriers (sort, categorize, and group)
    - View the list of tasks assigned to a courier
    - View the list of deliveries associated with a courier, visualize statistical properties of historical deliveries each courier has handled, or the current one.
- Manage the deliveries
    - View a delivery list (sort, categorize, and group)
    - Update an existing delivery (name, description, dispatch, …)
    - View the logs of each delivery task (delivery attempts, re-openings, …)
- Manage the system clock
    - Initialize and set the clock
    - Advance the time (minutes, hours, days) to allow the simulation expedited time keeping
- Manage various system settings

A courier could perform a limited set of actions. These may include

- View and update their personal details, including the maximal personal distance to which they are willing to travel.
- View the open delivery orders catalog, filtered to fit the personal travel distance limit.
- Choose a single open delivery order to handle at a time.
- View and update the delivery task details for the open order they are currently assigned to (including the update to status "delivered").
- Track and manage their own delivery history

Display screen specifications

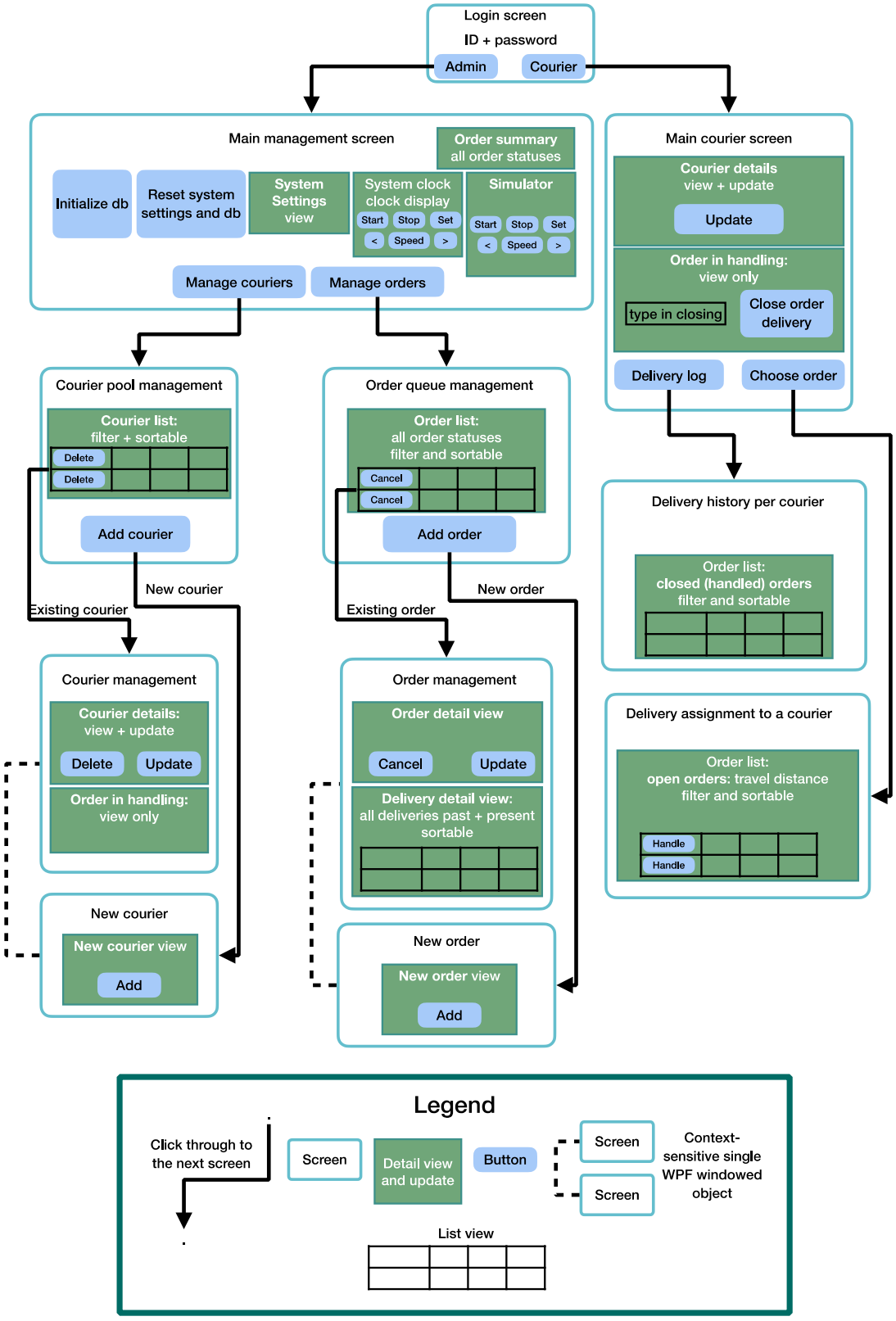


Figure 1: user interface screen diagram

At your team's discretion you can modify the scenario logic to create reasonable data and functional flow, as long as this could be fairly justified during the project presentation.

Screen description in full detail appears under the Screen functionality section. Here's a brief presentation of the screens visible in Figure 1:

- Entry/login screen – sign in for Admin or Courier
- Admin screens:
    - Main management screen – system overview and management
    - Courier pool management screen – courier pool inspection
    - New Courier screen
    - Courier management screen – for individual, existing, courier
    - Order queue management screen –inspection of orders of any status
    - New order screen
    - Order management screen – for individual, existing, order
- Courier screens:
    - Main courier screen – courier account management + order handling management
    - Delivery assignment screen – open orders list + choice of order to handle
    - Delivery history screen – order history of the courier

System logic

Delivery types

As long as a courier is not currently performing any deliveries (handling an order), the courier/admin will be able to select or change the "delivery type." When a courier is currently handling a delivery, the "delivery type" field is read-only.
The delivery type will be stored in the database under the courier's data entity (**DO.Courier**) and in the delivery data entity (**DO.Delivery**). It will further be displayed in the interface.

Delivery Types:

- Vehicle
- Motorcycle
- Bicycle
- On foot

Travel Distance Calculation

The delivery travel distance between the recipient's address and the dispatch center is an address-to-address routing with the use of geocoding. This is calculated when creating the Delivery data entity (**DO.Delivery**) [REF], and it is stored in the database. The travel distance is displayed on the **Delivery assignment screen**.

The calculation depends on the delivery type: (a) based on the assumption that cars and motorcycles travel along the same driving route, the distance of driving by car or motorcycle will be calculated the same. (b) based on the assumption that bicycles and pedestrians follow the same route, cycling and walking distances will be calculated the same.
A street address is represented by geodetic coordinates (Earth longitude and latitude). The details of calculating travel distances between addresses is in: Appendix 1 – Calculating Address Coordinates and Distance Between Addresses.

Delivery Completion Types

The following are the delivery completion types recorded in the **DO.Delivery** data entity:

- **Delivered** – The order was delivered (regardless of timing) and closed.
- **Customer Refused** – The courier arrived, but the customer refused to accept the order. The package is returned to the dispatch center and the order is closed.
- **Canceled** – The order was canceled after creation; two cases:
    - Before Delivery: An order awaiting assignment to a courier is on the list. It closes with a "dummy" delivery (start = end time), a type "Canceled," and a courier ID = 0.
    - During Delivery: The order was active; the admin requests the courier to return to dispatch. The order closes with an updated end time.
- **Recipient Not Found** – The courier arrives at the designated address, but the recipient is absent or unreachable. The package returns to dispatch, the delivery closes, and the order reopens.
- **Failed** – A route calculation error occurs when creating the delivery. The delivery closes; the order remains open.

Order Lifecycle Timing

Order lifecycle timestamps are ether stored as attributes in data entities (**DO**), or computed in the business layer and stored in logical entities (**BO**).

- **Order Opening Time** – Time the admin creates the order; stored in **DO.Order** [REF], and set using the system clock.
- **Delivery Start Time** – When courier collects an order; stored in **DO.Delivery** [REF], set by system clock at creation.
- **Expected Delivery Time** – An estimated delivery date and time, based on delivery type, distance, and average speed. Computed in business layer.
- **Delivery Ending Time** – Date and Time when the delivery finished, triggering order closure or reopening.
- **Maximum Delivery Time** – Latest allowable delivery date and time based on company's maximal delivery distance and order opening time. This is independent of the delivery type. Computed in business layer.

Order Status (OrderStatus)

Expresses the order status based on the latest delivery. Calculated in the business layer using data from DO.Order [REF] and DO.Delivery [REF]. After calculation it is stored as an attribute in certain business entities.

- **Open** – not assigned to any courier, and not yet closed.
- **In Progress** – currently handled by a courier.
- **Delivered** – order closed, customer received it, last delivery ended as "Delivered."
- **Refused** – order closed, last delivery ended as "Customer Refused."
- **Canceled** – order closed, last delivery ended as "Canceled."

Schedule Status (ScheduleStatus)

Represents the order's timing compliance based on the latest delivery. Calculated in the business layer from DO.Order and DO.Delivery. Variables that affect the schedule status are risk and max delivery time thresholds. These are set by the admin.
Timing status is estimated and then stored as an attribute in the relevant business entities:

- **On Time** – an open or active order with enough time to deliver within company's maximum delivery time window; or closed and finished before that maximum time.
- **At Risk** – an open or active order with less than the risk time window, with time left before maximum delivery time.
- **Late** – an open or active order exceeding the maximum delivery time, or closed after the maximum delivery time.

Courier wage

Add-on
The courier's pay method should be proposed by your team based on approaches that exist as market-standard. It is recommended to get AI assistance and ask it to align with Israeli employment models. You are welcome to extend this model to use pay slips and expense accounts. Attributes and methods should be added as needed in the system configuration and the relevant entities. Wages should display for both courier and admin daily, monthly, and annually.

Use-case scenarios

The following example scenario illustrates the system's operational flow and main functionalities.

The admin defines global configuration, including:

- Company address
- Maximum delivery distance
- Average speed per delivery type

- Maximum delivery time window
- Risk time window

At any given time, the admin may add a courier to the system:
→ The courier's details are stored in the database (ID, delivery type, work start date, and personal max travel distance).

Whenever a customer calls to place an order, a new order is created:
The admin creates an order
→ An order record is created with a unique running ID, recipient address, and order creation timestamp.
→ The Status is initially set to *Open* and remains so until closed (e.g., delivered, canceled, etc.).
→ Expected and maximum delivery times are estimated but not stored.
→ The admin can view all orders in any status.
The courier
→ The courier only sees open orders within their defined maximum personal travel distance.
→ The courier can select an open order for handling.

When a courier selects an order for delivery:
→ A delivery record is created (courier ID, order ID, start time = now, end time and completion type = null).
→ Order status is set to *In progress*; no other courier can select it now, the current courier cannot choose other deliveries.
→ The order in handling appears on the courier's Main screen.

A courier reports successful delivery:
→ Ending time is updated from the system clock; completion type is set to *Delivered*.
→ Order status is set to *Delivered* (checked against the maximum delivery time).
→ The courier sees it in the delivery history screen; the admin sees it in the order queue management screen.

A courier reports that the customer could not be located:
→ Ending time is updated from the system clock; completion type is set to *Customer not found*.
→ The order returns to dispatch, it is set to *Open* again, and is available for reassignment to couriers.
→ The order is now visible in the courier's history and the admin's order queue management view.

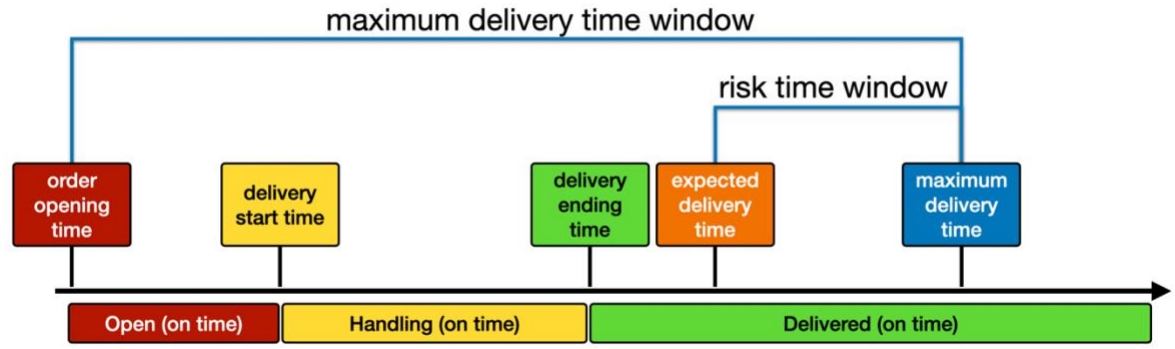
When a customer cancels an order already in progress:
→ The admin contacts courier asking to return to dispatch.
→ Order status is set to *Canceled*.
→ Ending time is set to the current system clock; completion type to *Canceled*.
→ The order is visible in the courier's delivery history screen, and in the admin's order queue management view.
When a customer cancels an order that's open but does not have a delivery progress status:
→ Order status is set to *Canceled*.

→ A “dummy” delivery is created (start and end times are identical, completion type moves to *Canceled*, courier ID is set to 0).
→ The admin can view the canceled order in the order queue management screen.

Order lifecycle charts

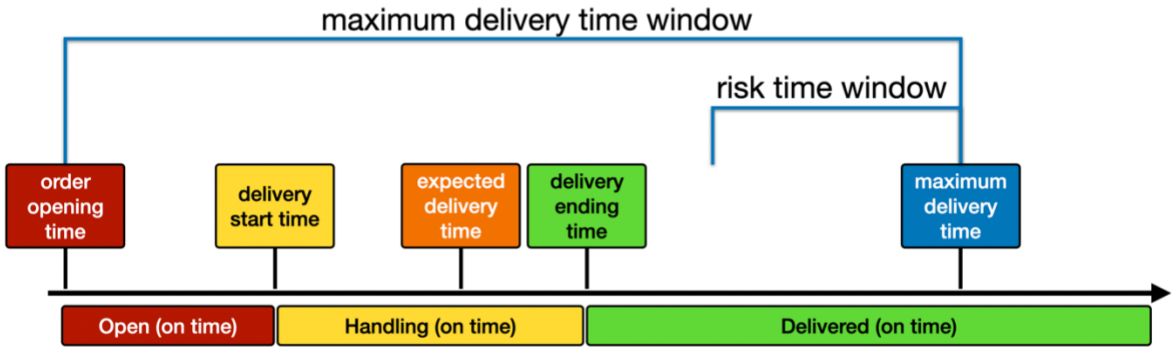
Case 1

Delivery ends and order closes before expected delivery time, before risk window, and before max delivery time (On-time).



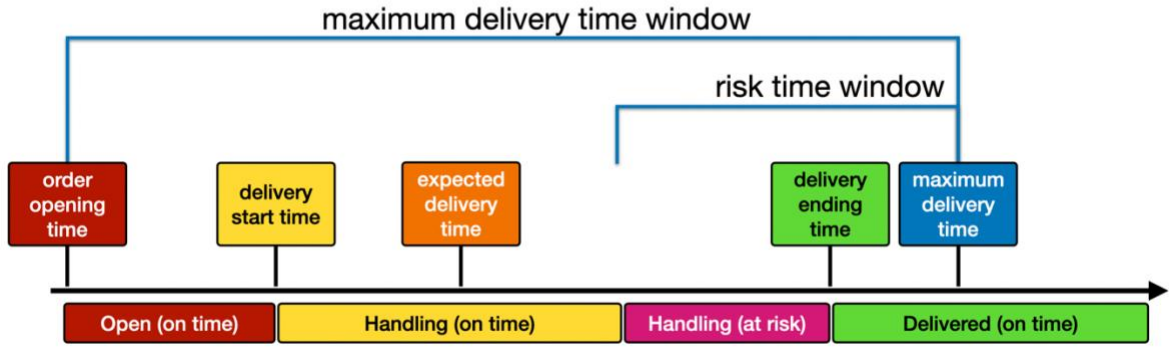
Case 2

Delivery ends and order closes after expected but before risk and max delivery time (On-time).



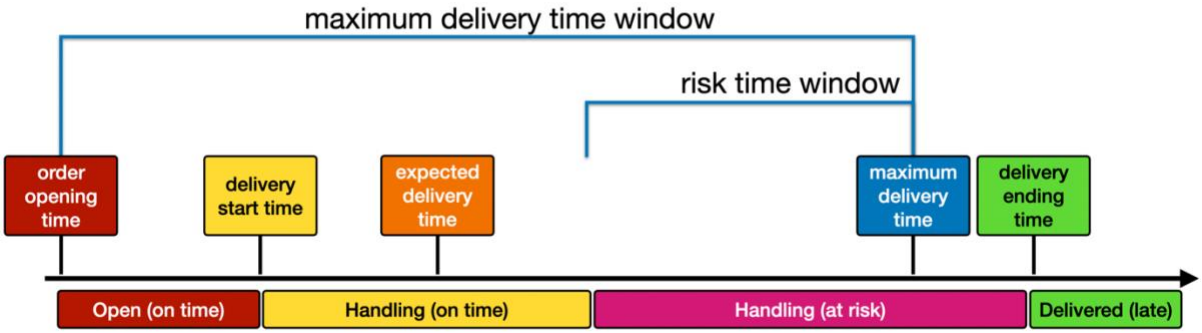
Case 3

Delivery ends and order closes after expected time and within risk window but before max delivery time (At risk, delivered on-time).



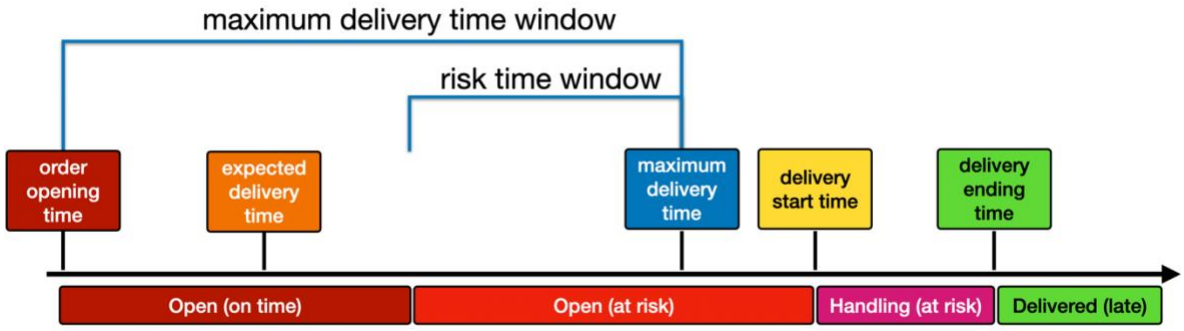
Case 4

Delivery ends and order closes after both expected and max time, following risk window (Late).



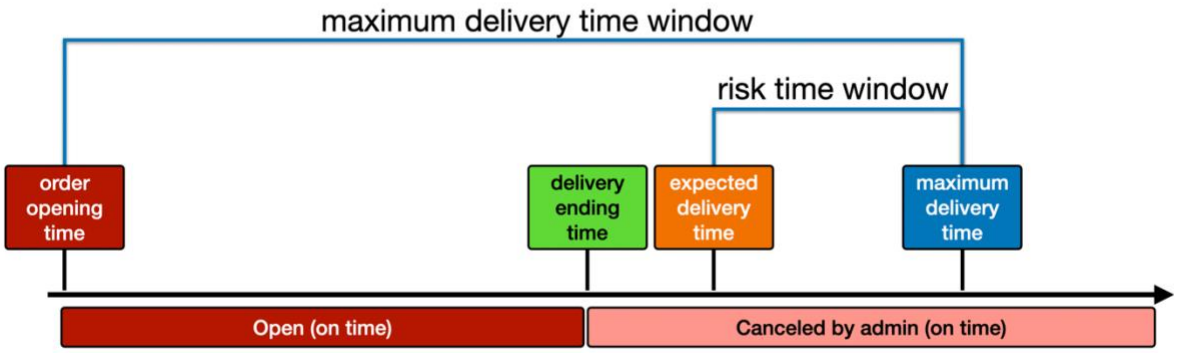
Case 5

Order enters risk window before handling begins, assigned after max time, and closes after expected and max time (Late). Delivery ends.



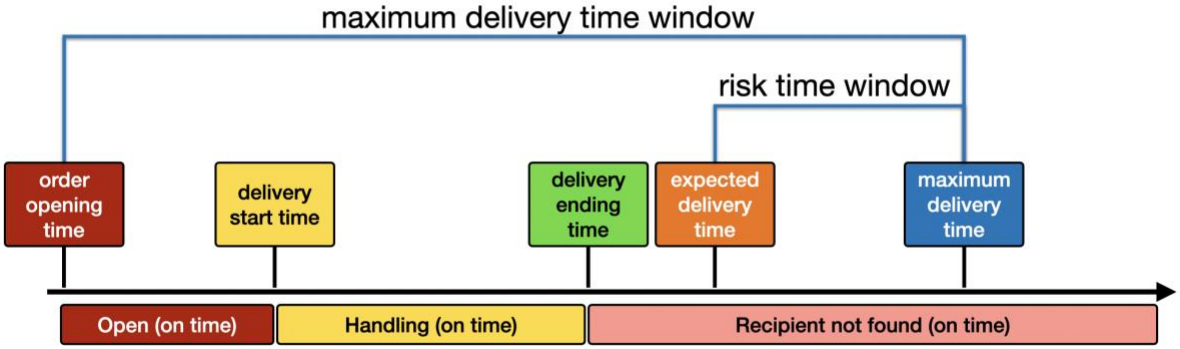
Case 6

Delivery ends and order closes before expected, risk, and max time (On-time).
Reason for closing: customer called to cancel; canceled by admin before handling.
Order returns to dispatch and closes permanently.



Case 7

Delivery ends and order closes before expected delivery time, before risk window, and before max delivery time (On-time).



The Data Access Layer

The purpose of the DAL is to enable efficient storing and retrieving data in flat format, allowing a disconnect from any logic or computational actions.

Data Layer Entities

We define the following entities: a configuration **(Config)** entity, and three data entities (**DO.Courier**, **DO.Order**, **DO.Delivery**)
Also, we may define interfaces for each data entity to represent the data brokerage system. These contain CRUD (Create, Read, Update, Delete) methods.
In the first few stages the database will be a memory store, using **lists** of data objects as the storage model.
Functionality: (A) The configuration entity and the data entities should be reset-able by restoring a default set of values to the **Config** entity, and clearing the **data entity lists** from data objects. (B) The data model should be populated initially by a data-init method such that we could run the program and load fake information into all

fields. This method should create valid and reasonably accepted lists of data objects. (C) A test suite (**DalTest**) should enable manual testing of the DAL. This test module should include initialization of the data object lists and a CLI (text-based) interface.

The configuration **(Config)** entity

A static entity. No objects should be instantiated from this entity. Its properties can be regarded as “System variables”.

Attribute Name	Type	Nullable	Updatable	Additional Explanation
Auto-incrementing identifier **`NextOrderId`**	int	No	No	Represents the identifier number for the next new order. Increments automatically (increments by 1 each time the number is retrieved, when adding a new order).
Delivery identifier **`NextDeliveryId`**	int	No	No	Represents the identifier for a new instance of the delivery entity that links a courier to an order. The next identifier for a delivery entity is taken from this field. Increments by 1 automatically when adding a new delivery entity.
System clock **`Clock`**	DateTime	No	By admin or simulator	Since our system simulates courier management, a system clock is maintained separately from the computer’s real clock. The administrator can initialize and update the system clock’s time. In every operation, the system’s time will be the current value of the system clock.
Administrator’s identifier **`AdminID`**	int	No	By admin	Represents the identifier for the administrator. A numeric value with a valid parity digit.
`AdminPassword`	string	No	By admin	**Add-on 1:** At initialization a default password will be assigned, after which the admin can update it. **Add-on 2:** Verify that the password is strong. **Add-on 3:** Encrypt the password. This attribute depends on (1) and (2).
Dispatch full address **`CompanyFullAddress`**	string	Yes	By admin	A complete and valid address, in proper format, of the company’s site. All couriers depart from here to perform deliveries and return here at the end of each delivery. **Examples of addresses:** הנשיאים 7, פתח תקוה, ישראל; המרכז האקדמי לב, ירושלים, ישראל. The business layer will check whether the address truly exists in reality and will calculate its geodetic coordinates. This information will be stored in the configuration entity. The address will be initialized to null. Should the address be invalid at update time, its previous value in the data layer will not change. As long as the address value is null (i.e., no valid address), orders cannot be opened. **Stage 1:** The address will be hardcoded. **Stage 4:** Validation will be done synchronously via code. **Stage 7:** Geocoding will be done asynchronously.

Attribute Name	Type	Nullable	Updatable	Additional Explanation
`Latitude`	double	Yes	Automatically by logical layer upon company-address update	Latitude – a real number indicating decimal degrees north (positive) or south (negative) from the Earth's equator. As long as the address is invalid, both latitude and longitude will be null. These attributes are intended for computing distances between addresses and are not meant to be displayed in the user interface. The value will be updated by the business layer whenever the company address changes. Please browse the www for this computation. **Stage 1:** Value will be hardcoded. **Stage 4:** Coordinate computation will be done synchronously. **Stage 7:** Computation will be asynchronous. See Appendix 1 – Calculating address coordinates and distance between addresses.
`Longitude`	double	Yes	Automatically by logical layer upon company-address update	Longitude – a real number indicating decimal degrees east (positive) or west (negative) from the prime meridian. As long as the address is invalid, both latitude and longitude remain null. These attributes are used for calculating distances between addresses and are not intended for display. The business layer updates them whenever the company address changes. See Appendix 1 for details.
Maximal delivery radius **`MaxGeneralDeliveryDistanceKm`** (aerial distance)	double	Yes	By admin	The maximum aerial distance (in km) between the company's address and the recipient's one. Only orders located within this range will be accepted. If the value is null, the company has no distance limits for orders. A simple mathematical function exists online for calculating the aerial distance between two addresses given their coordinates.
Average driving speed **`AvgCarSpeedKmh`**	double	No	By admin	Average driving speed by car [km/h]. Used for estimating actual delivery times and distances from the company to the recipient.
Average motorbike speed **`AvgMotorcycleSpeedKmh`**	double	No	By admin	Average motorbike-ride speed [km/h]. Used for computing actual delivery times and distances from the company to the recipient.
Average bicycle speed **`AvgBicycleSpeedKmh`**	double	No	By admin	Average riding speed in km/h by bicycle. Used for computing actual delivery times and distances from the company to the recipient.
Average walking speed **`AvgWalkingSpeedKmh`**	double	No	By admin	Average walking speed in km/h on foot. Used for computing actual delivery times and distances from the company to the order.
Maximum delivery time window **`MaxDeliveryTimeSpan`**	TimeSpan	No	By admin	The company's commitment to the delivery time of all orders to customers. A time range that helps determine whether an order is at risk and which orders were delivered on time or not. The time units

Attribute Name	Type	Nullable	Updatable	Additional Explanation
				(hours/days) will be set according to the company type.
Risk time window **`RiskRange`**	TimeSpan	No	By admin	A time range after which an order is considered at risk—approaching the required maximum delivery time and still not delivered. Time units (hours/days) will be set according to the company type.
Inactivity time window **`InactivityRange`**	TimeSpan	No	By admin	A time frame during which a courier has not been active; beyond it, the courier will automatically be defined as "inactive." During this period the courier has not performed any deliveries. The time units (hours/days) will be set according to the company type.

The data entities **DO**

DO.Courier

Attribute Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
Courier identifier **`Id`**	int	No	No	The unique ID of a courier. A numeric value with a parity digit.
`FullName` (First and Last)	string	No	By admin or courier	The full name of the courier.
Telephone number **`MobilePhone`**	string	No	By admin or courier	Represents a valid mobile phone number of the courier. A string of exactly 10 digits starting with 0.
`Email`	string	No	By admin or courier	Represents a valid email address in proper format. See online resources for how to check this later (in the Business Layer).
`Password`	string	No	By admin or courier	**Add-on 1:** Initially, a default password is assigned by the admin, later the courier may update it. **Add-on 2:** Verify that the password is strong. **Add-on 3:** Encrypt the password. This attribute does not exist for those who do not implement this add-on.
`Active`	boolean	No	By admin only	Indicates whether the courier is active or inactive. For a resigned couriers, the delivery history is retained, but the courier cannot handle new orders.
Maximal personal travel distance **`PersonalMaxDeliveryDistance`**	double	Yes	By admin or courier	The courier's personal maximum radius of operation [km] from dispatch, up to which the courier is willing to deliver. Must be less than or equal to the company's general maximum delivery radius. The Business Layer will allow the

Attribute Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
				courier to choose an order to handle from all orders within this radius. If the value is null, the courier has no distance limits for orders. A simple mathematical function exists online to compute the aerial distance between two coordinates.
`DeliveryType`	ENUM	No	By admin or courier	Delivery type: **Car / Motorcycle / Bicycle / On foot.** See details of delivery types and their impact on delivery distance and speed estimations here: *Delivery Types*[REF]. Can be updated only as long as the courier is not currently handling an order.
Work start date and time `EmploymentStartTime`	DateTime	No	No	The date and time at which the courier started working for the company. Effectively it's the time of the current courier's entity creation. It will be set according to the current system clock, stored in the configuration entity.

DO.Order

Property Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
Order Number auto-incremental `Id`	int	No	No	Represents the unique identifier of the order entity. When an order is created, the system retrieves the next running order number from the Configuration entity.
`OrderType`	ENUM	No	By admin	Different types of orders depending on the company type.
`VerbalDescription`	string	Yes	By admin	A short textual description of the order's contents.
`FullOrderAddress`	string	No	By admin	The complete and valid address of the order location in a proper format.Examples:• HaNesi'im 7, Petah Tikva, Israel• Jerusalem College of Technology, Jerusalem, Israel. The Business Layer will verify that the address exists and calculate its Lat/Lon. These values will also be stored in the order entity. In the Data Layer, only valid addresses (including valid lat/lon) can be stored. Stage 1 – address are manually entered. Stage 4 – synchronous validation. Stage 7 – asynchronous calculation. See Appendix 1 – Calculation of Address

Property Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
				Coordinates and Distance Between Addresses.
`Latitude`	double	No	Automatically updated in the BL following any update to the customer's address	Latitude – measures how far north or south a point is from the equator. Longitude – measures how far east or west a point is from the prime meridian. Used for distance calculations only; not displayed in the UI. The value updates every time the address changes. Stage 1 – manually entered. Stage 4 – synchronous calculation. Stage 7 – asynchronous calculation. See Appendix 1 – Calculation of Address Coordinates and Distance Between Addresses.
`Longitude`	double	No	Automatically updated in the BL following any update to the customer's address	Longitude of the order address (check the explanation for Latitude).
`CustomerFullName`	string	No	By admin	Full name of the person placing the order.
`CustomerMobile`	string	No	By admin	Customer's mobile phone number (for contact if needed). Must be a valid Israeli number: exactly 10 digits, starting with 0.
`Volume`, `Weight`, `Fragility`, etc.	Mandatory, but at your discretion	Mandatory, but at your discretion	By admin	Additional optional fields describing the order, depending on company needs. Add as desired.
`OrderOpenTime`	DateTime	No	No	Date and time when the order was created. Determined by the system clock (in the Configuration entity).See *Order Lifecycle Times*.

DO.Delivery

Attribute Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
Delivery auto-incremental identifier **`Id`**	int	No	No	Represents the unique identifier of the delivery entity. When a courier chooses to handle an order, a linking entity is created. The system accesses the configuration entity to obtain the next running delivery ID for it.
Order identifier **`OrderId`**	int	No	No	Represents the unique identifier of the order the courier chose to handle.
Courier's identifier **`CourierId`**	int	No	No	Represents the ID number of the courier who chose to handle the order. In the case of a "fake delivery", a delivery entity is created as a result of order cancellation (by admin) before assigning to any courier. In this case, the ID number will be **0**.
`DeliveryType`	ENUM	No	No	The type of delivery at the time the delivery was opened. (The courier's delivery type may

Attribute Name	Type	Nullable	Updatable	Additional Explanation The value is validated only in the Business Layer. Here, it should be assumed that the value is valid.
				later change in the Courier data entity, but it will not affect deliveries already performed by that courier.)
Delivery start time **`DeliveryStartTime`**	DateTime	No	No	The date and time when the current order was collected by the courier from the company. This is also the moment when the current delivery entity was created. Determined according to the current time of the system clock, stored in the configuration entity. In the case of a “virtual delivery”, a delivery entity is created due to an order “cancellation” (by admin) — therefore, the **delivery end time** will be identical to the **delivery start time**. See details here: *Order Lifecycle Timing*.
Travel distance **`ActualDistance`**	double	Yes	No	The travel route [km] between the company and recipient’s addresses. Computed according to the type of delivery chosen to handle the order. See details on how to compute distance here: *Actual Order Distance Calculation*[REF]. The distance is calculated in the **Business Layer** and stored in this property. As long as the distance calculation has not been completed, the value remains null. If the calculation fails (e.g., due to a network error), the delivery is closed with an ending type of “Failed”. In the early project stages, the calculation is performed synchronously. In stage 7, it must be performed asynchronously.
`DeliveryEndType`	ENUM	Yes	No	Describes the completion manner of the current order by the current courier. See details of end types here: *Delivery End Types*[REF]. If delivery is still in progress, the property value is null. In the case of a “fake delivery” created due to an order cancellation (by admin), delivery end type is “Cancelled”.
Delivery ending time **`DeliveryEndTime`**	DateTime	Yes	No	The date and time when the delivery was completed. Setting the *DeliveryEndType* triggers update for *DeliveryEndTime*, based on the current system clock (a property in the configuration entity). As long as the delivery has not yet ended, both properties remain null. The delivery entity continues to exist for the purpose of delivery history tracking. If the delivery is still in progress, this value is null. In the case of a “fake delivery” created as a result of order cancellation (by admin), the *DeliveryEndTime* will be identical to the *DeliveryStartTime.* See details here: *Order Lifecycle Timing*[REF].

Service Contracts (Interfaces) of the Data Layer – DalApi

- A CRUD interface must be defined for every entity and implemented according to the project's Stage 1 and Stage 2 documents.
- Each entity must have an interface containing all CRUD methods, and an implementation of that interface.

Important:

- To implement these methods properly, strictly follow the exact procedure described in the appendix "Extension on CRUD Access Methods" in the documentation of Stage 1.
- An interface for the configuration entity (**IConfig**) must also be defined and implemented.

Implementation rules:

- Assume all inputs to CRUD methods are valid; no logic or validation should occur here.
- Only ID validation is allowed.
    - **Create**
        - If the entity comes with a predefined ID (e.g., national ID), ensure it's unique; otherwise, throw an exception.
        - If the entity does not have a predefined ID (e.g., auto-incremental), the Data Layer generates it in the Create method.
    - **Read**
        - Verify that an entity with the given ID exists; otherwise, throw an exception.
    - **Update**
        - Assume all fields are valid; verify that an entity with the given ID exists; otherwise, throw an exception.
    - **Delete**
        - The decision whether deletion is permitted is made in the Business Layer. Therefore, assume the request is valid, and only verify that the entity exists; Throw an exception otherwise.
- **Filtering by a filter** passed from a higher-level layer is implemented in Stage 2 (advanced).
- All other logic and validations belong to the Business Layer (BL) in later stages.

Additional Notes:

- The **Courier** entity uses a **national ID** as its unique key (not an auto-incremental value). When adding a courier, its identifier does not originate in the DAL. Therefore, the DAL should ensure that its ID is unique.
- The **Order** and **Delivery** entities do use **auto-increment IDs**, generated by the Data Layer's `Create` method via the Configuration entity's next-running-number mechanism.

- **Date-type properties** (e.g., order opening time) that are assigned from the system clock, should be set in the Business Layer (or **DalTest**). Therefore, these properties are already set with values before CRUD execution.

Data Initialization

- The database must be initialized with consistent and interrelated values forming a functional system ready for testing.
- Initialization occurs within the **DalTest** project, inside the `Initialize.Do` method and the supporting helper methods (as defined in Stage 1).

Configuration Initialization

- One admin – their ID number is stored in the **Config** entity.
- Company address and coordinates are hardcoded into the initialization of **Config**.
  See *Appendix 1 – Calculation of Address Coordinates and Distance Between Addresses*.
- All other configuration variables are also initialized manually with appropriate values.

Entity List Initialization

Data must be valid, legal, and logically related, maintaining referential integrity.

All objects are created by **calling the DAL layer's Create method** for the corresponding entity. Configuration must be initialized first, as other entities depend on it. Helper methods may be written to avoid code repetition.

Couriers

- At least **20 couriers**.
- Most are active, some inactive (former employees with delivery history).
- Include all delivery types (car / motorcycle / bicycle / on foot).
- Generate a fixed collection with IDs, names, emails, and delivery types.
- Randomize delivery type with uniform distribution.
- Choose at random personal maximum delivery radii within the company's global limit.
- Randomize start-of-employment times earlier than the system clock.
- Use reasonable logic in creating all other properties.

Orders

- At least **50 orders**, including:
  - 20 open
  - 10 currently in progress
  - 20 closed (all types)
- Order IDs are auto-generated in the DAL (so pass 0 during creation).
- Create a realistic mix of order types and descriptions.

- Randomize the opening times, all should be earlier than the system clock.

Delivery addresses:

- Please use real addresses within the company's maximum delivery range.
- Place orders that are geographically near to the company location.
- Manually retrieve latitudes, longitudes, and distances when preparing the address list.
- Maintain a list/array of addresses with coordinates and distances (air, walking, and driving).
- When creating orders assign each order's address, at random, from this list.
- Implement a helper method to calculate the aerial distance between an order's address and the company address using coordinates.

Deliveries (Courier–Order Assignments)

- Include deliveries with all possible completion types (including reopened ones).
- Initialize deliveries after initializing couriers and orders.
- When initializing the deliveries we retrieve courier lists and order lists from the DAL and assign deliveries in a loop
    - Delivery IDs are auto-generated (set to 0 when creating).
    - Randomly pick order and courier ID's (ensuring distance ≤ courier's maximum radius).
    - Randomize delivery start times appropriately based on order start time.
    - Ensure a courier is not handling overlapping deliveries.
    - If a delivery is complete:
        - Randomize the ending time reasonably.
        - Choose at random the completion type (allocate probabilities to the types when randomizing).
    - If a delivery is active or has a type that indicates a closed order, remove that order from the order queue to prevent re-selection.

Resetting the Data Layer

Reset Configuration Data

As part of the **IConfig** interface, define and implement a **`Reset()`** method returning configuration data to their initial values.

- Auto-incremental counters – reinitialize to any starting value (not necessarily 1).
- System clock – may reset to any reasonable time.
- All other configuration data – reset to starting values convenient to you.

Reset Entity Lists

- Clear all entity lists (empty them completely).
- For each entity, reset using its CRUD **`DeleteAll`** method.

Translation Score: 10

Test Program – DalTest

- The **DalTest** program should allow the following operations continuously until "Exit" is chosen:
  - Exit
  - Input/output for CRUD, and DeleteAll for each entity
    - Menu per entity listing all CRUD actions
    - Read other appropriate user input, call the corresponding interface method
    - Catch and display exceptions
  - Database initialization
  - View all database content at once
  - Configuration options menu
  - Reset the entire DAL (configuration + lists)
- See the detailed testing requirements in the Stage 1 project document.

Reminders:

When running the tests, input must be valid, legal, and logically consistent.

Input format validation for non-string data is handled automatically by **`Parse`** or **`TryParse`**.

If parsing fails, the user must be prompted to re-enter the value.

Data storage in XML files

According to the Stage 3 guide, a separate XML file must be kept for each entity in **DO**. In this project this means that there will be three XML files that reflect the **DalList** object lists in **`DataSource`**.
Courier store – couriers.xml
Order store – orders.xml
Delivery store – deliveries.xml
Configuration store – **`data-config.xml`**:

The configuration file should store all the configuration options for the project, including the next available identifier for each auto-incrementing Id field, and the system clock (start end and state).

Appendices

Calculating Address Coordinates and Travel Distance Between Addresses

Preface

In this project, we will handle addresses (company and order addresses) and compute distances between them:

- Distance on the Earth's surface – the "Aerial" distance
- Distance along routes
    - Driving distance
    - Walking distance

Examples of addresses:

- HaNesi'im 7, Petah Tikva, Israel
- Jerusalem College of Technology, Jerusalem, Israel

A valid geocoded address is represented by two geographic coordinates:

- **Latitude**: degrees on the Earth's sphere. Range: –90° to +90°. Positive = North of the equator, Negative = South of the equator
- **Longitude**: degrees on the Earth's sphere in East–West axis. Range: –180° to +180°. Positive = East of Greenwich, Negative = West of Greenwich

To compute any distance type between two addresses, both points must have valid coordinates (in our project - on a lat/lon scale).
If an address is valid (exists), its coordinates can be calculated.

Required Calculations for the Project

- Given a valid address → find its coordinates (Latitude, Longitude)
- Given two valid addresses (two coordinate pairs) → retrieve the distance (aerial / driving / walking)

Notes:

- Geocoding (finding the coordinates), and estimating driving, or walking distance requires an **API call** to external **geocoding services**.
- **Aerial distance** uses a simple mathematical formula based on a pair of coordinate (available online).

Distance estimations through the project stages

Stage 1 (Initialization): Calculations performed manually using external websites. Results entered explicitly into the system.

Stage 4 (Business Layer): Calculations done in code via a synchronous method that calls a geocoding web API. Execution waits until the network response arrives (no async/await).

Stage 7 (Simulator): Code updated to use asynchronous network requests to geocoding services. See details in the documentation for Stage 7.

Manual Calculation

1. Manual Coordinate Lookup

Using Google Maps (or similar):

1. Open Google Maps.
2. Enter the desired address and search.
3. A red pin appears on the map.
4. Right-click the pin → menu shows latitude/longitude.
5. Click the coordinate values to copy them.

2. Manual Routing Distance (Driving/Walking)

1. Open Google Maps.
2. Choose “Directions” and enter two addresses.
3. Select travel mode: driving or walking.
4. The route appears on the map.
5. Hover over the route — a bubble shows total distance.

Geocoding via Code

To calculate through code, use external **Geocoding APIs** that support:

- **Forward Geocoding:**
  Convert a text address to geographic coordinates (Lat/Lon)
- **Reverse Geocoding:**
  Convert coordinates to a textual address
- **Routing:**
  Compute a route between two points (driving, walking, etc.)

For this project, we may need **Forward Geocoding** and **Routing**.

Example Geocoding Providers

- **OpenStreetMap (OSM):**
  Free of charge, no API key required, unlimited daily requests.
- **Google Maps:**
  Paid subscription beyond the free usage limit. Requires a credit card to enroll. In practice, usage here will remain free.
- **LocationIQ:**
  Free up to 5,000 requests/day. Google account supported.

- **geocode.maps.co**
  Free, requires an account.

Calling a Geocoding API (Example)

To retrieve the coordinates for a given address:

1. In C#, send an **HTTPClient** request to one of the geocoding APIs.
2. The query string includes the API URL and the desired address.
3. The server searches its database for the address.
4. The response is returned in **JSON** or **XML** format, with the coordinates (Latitude, Longitude).
5. Parse and extract the coordinate values in the response.

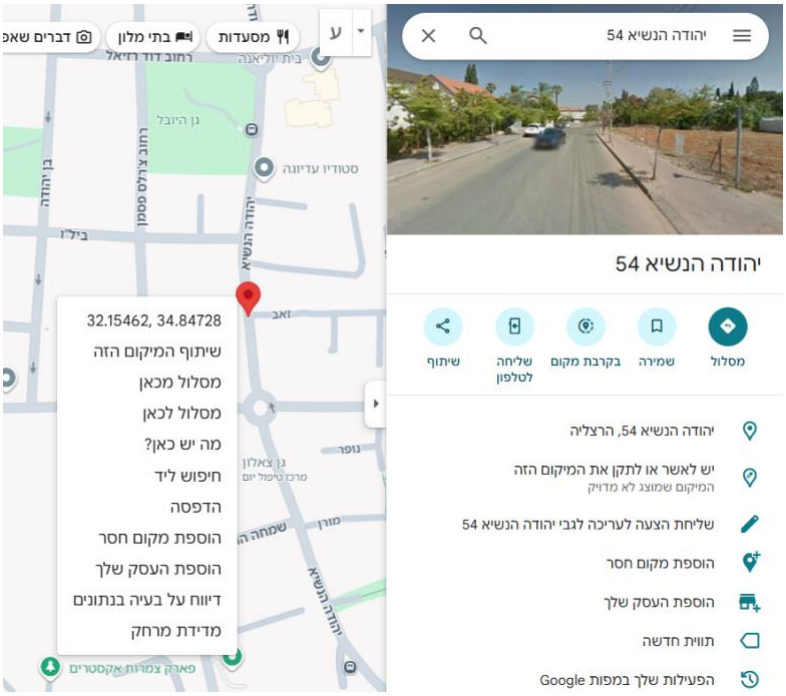


Figure: manual search for coordinates