Candidate nr: **2044 |2049| 2090 | 2073 | 2093 |**

SHIPPING MANAGEMENT SYSTEM APPLICATION

PG3302 Software Design

A project Submitted to Kristiania University College in Information Technology in the Department of Economics, Innovation, and Technology

Initialize the application.

The C# Application is compressed as a (ZIP) file and contains an executable file which you can use to initiate the Application.

2023

Kristiania University College

Abstract

The aim of this project is to develop an online **Shipping Management System** console-based application. This application is developed using C# and uses the .NET framework to create a dependable and effective tool for the customers. The application feature helps customers to place single or several shipping orders at once. Customers can choose a wide range of delivery modes/methods *(e.g., Air-Delivery (Airplane), Land-Delivery (Truck) ,Sea-Delivery (Ship))*. This application will simplify the ordering process by providing a series of text-based commands the user can interact with throughout the application. The solution provides an interface and changes menu to present all available options by providing both admin and user Dashboard. Allows customers to choose preferable shipping modes & methods they like and edit, update delivery details, and submit their order. When the order has been placed and payment has gone through, a confirmation will be displayed to the customers Desktop. Order details will be updated across both the database and the user & admin panel/ Dashboard.

This documentation will provide a comprehensive overview of the **Shipping Management System** Application project. As well as our group's collaborative process to achieve a final product.

Keywords: Goods, Online, Management, Use Case Diagram, UML Diagram ,ER Model, Database (DBMS), Diagram.

Table of Contents

Figure

Introduction

This section provides a high-level overview of the application. It then defines the objective and purpose of the project.

Project Overview

Our main goal with this project was to provide the Shipping Management business with a cutting-edge online Shipping Management System solution. No doubt, high technological developments have improved streamline and efficiency as more and more sectors use them. But there are always improvements to be made. We have in this project developed a desktop-based application that will enable a smooth and user-friendly Shipping Management System.

Project Aim and Objectives

Our application mainly focuses on Shipping Management by country thereby within regions. Customers can register and login and then place their shipping order. Delivery data extracted from the user will be handled by the application to process the delivery, except from the actual delivery mechanism. Assume the following: Our Application / Shipping Management System does not own any transport companies. It coordinates with the third party to plan and execute the actual delivery. The application will also allow Administrators to manage (Update/Edit/Delete) orders view order status.

Features

- Verification & Authentication: Makes the application safe in terms of online attacks.
- Delivery order: Allows registered customers to place different types of delivery orders.
- One - many Schemes: Customers can place more than one order at a time.
- User-friendly **:** The console application is simple and easy to use.
- Environmentally sustainable: Digital versions of all documents regarding delivery eliminate needless paper waste.

Functionality

The application has two main distinct segments in terms of roles, each with a unique purpose and functionality. These segments consist of the administrative side and customer-facing components, which are accessed through the console Application.

The forthcoming sections will provide a clear picture of their operational significance within the broader framework of the Shipping Management System.

Design and Implementation

Our application is implemented in C# and is console-based. Customer interface is implemented using text-based commands and the data is stored in SQLite database. Application provides Admin panel/Desktop and Customer Panel/Desktop that can only be accessed by the registered profile.

Use Case Diagram

Use case or behavioral UML(Unified Modeling Language) as a fundamental component gives a good overview of actors that are in our application. Encapsulated below is our proposed Use Case Diagram representing the admin side / Actor and customer /Actor. The purpose of these diagrams is to illustrate how these users interact with the application system. Showcasing as a tool helped us to identify the functional requirements and see user interactions clearly before applying the software architecture and kept our project on track.

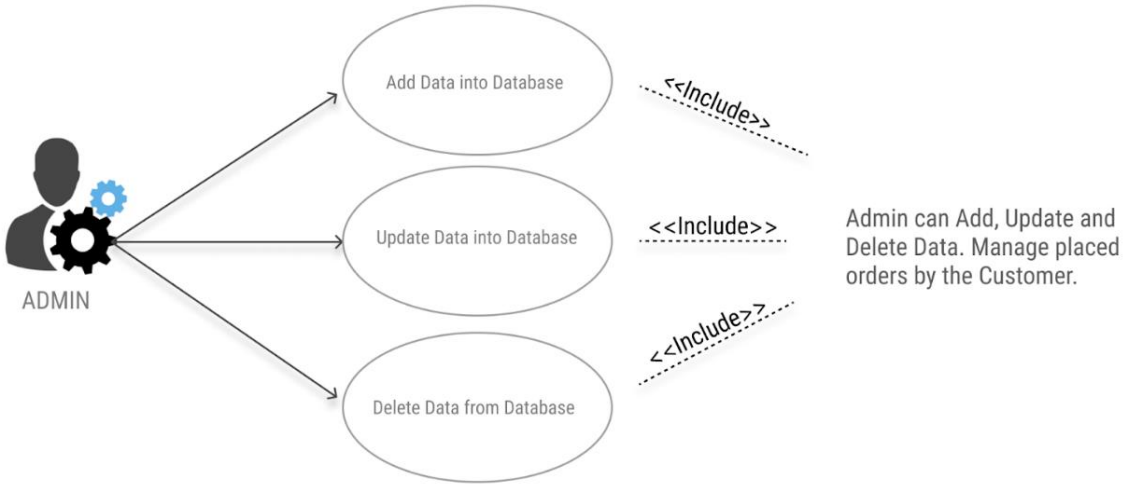


Figure I Use Case Diagram - User Side

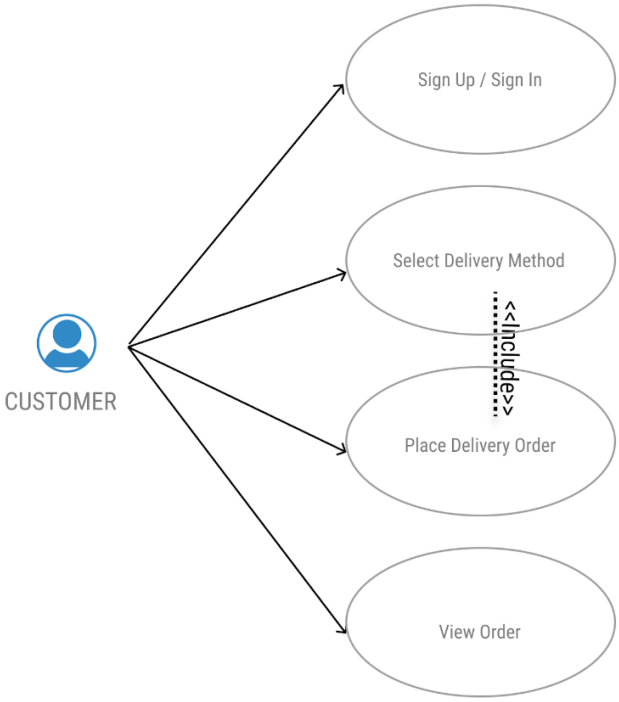


Figure II Use Case Diagram - Admin Side

Administration Management Platform

The diagram below illustrates the core functionality of the administrative side.

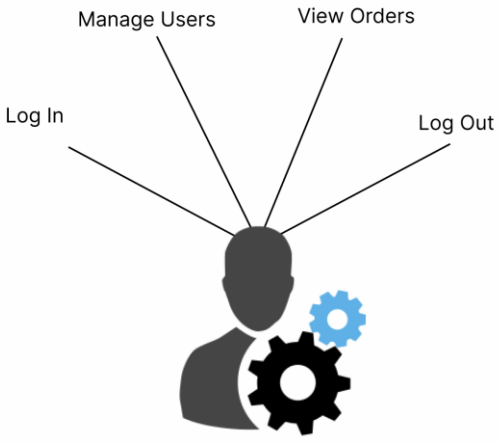


Figure III Functionality of Administrator

Detail of functionality is as follows:

Sign In
Properties:

- Username
- Password

When connected to the management platform, a login input log with Username and Password will be displayed. Administrator will insert a default Username and Password, provided by us. After a successful login, the Administrator will be able to change/update current Password from within the desktop of the Administrator.

If access approved, the Administrator would see the following menu:

- Manage Users
- View Orders
- Sign out

Menu Details:
Manage users:
Properties:

- List of all registered users
- Delete user
- Create a new Admin. Account
- Account Setting
- Return

When Administrator clicks on “Manage users”, a list of properties will be displayed:

- List of all registered users
    - This will display registered users including their unique ID and Username. Password shall be displayed.
- Delete user
    - This functionality is there to terminate users
- Create a new admin Account
    - This gives the Administrator full access to add a new Administrator.
- Account Setting
    - This allows the Administrators to modify and update their own password
- Return:
    - When clicking on return the Administrator will be redirected to the main menu

View Orders:
Allows Administrators to view order History.

Sign-out:
The administrator will be redirected to the Log-In menu.

Potential :

Future development for Administrator panel.

During the design process our group had elaborated several functionalities and features for the Administrator panel/desktop. That involved Delivery Modes and Delivery Methods.

Delivery Modes

Properties:

- View Delivery Modes
- Add Delivery Modes
- Edit Delivery Modes
- Enable / Disable Modes

When Administrator clicks on Delivery Modes, functionality detail of the properties are as follows:

- View Delivery Modes
    - When the Administrator clicks on “View Delivery Modes”, a list of modes will display. The list consists of the following three:
        - Air (Fastest mode of transportation, therefore higher price)
        - Sea
        - Land
- Add Delivery Modes
    - Administrators will be able to add new features/modes of transportation for delivery services. This functionality is there for future development.
- Edit Delivery Modes
    - This gives the Administrator access to make changes regarding delivery Modes. This functionality is also reserved for future development.
- Delete Delivery Modes
    - This functionality allows the Administrator to delete(Enable/Disable) Delivery Modes.

Delivery Methods

Properties:

- View Delivery Methods
- Handle Delivery Methods

When the Administrator clicks on “Delivery Methods”, a list of delivery methods will appear.

- View Delivery Methods
    - Se all delivery methods
- Handle Delivery Methods
    - Enable and Disable

Foreground Customer Management Platform

The following illustration represent the functionality of a user side in our application.

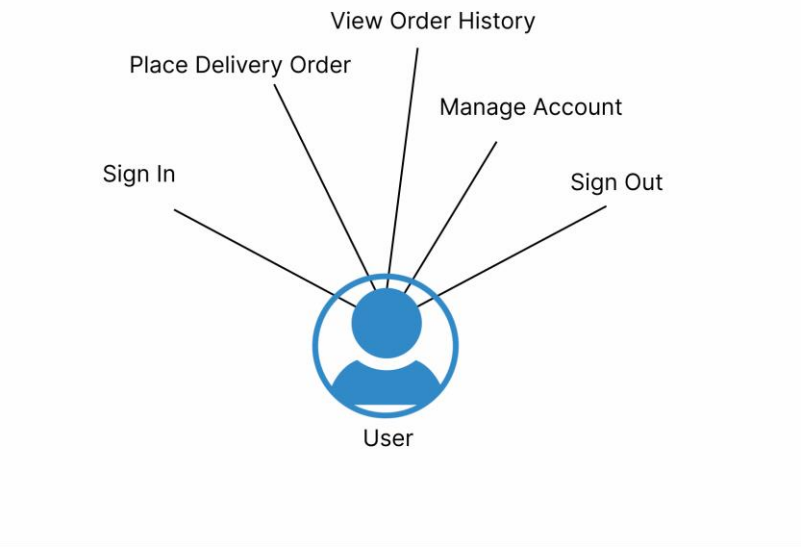


Figure IV Functionality of User

Detail of functionality is as follows:

Sign In

Properties:

- Username
- Password

Login input field with Username and Password will display. Customer will insert Username and Password. Authentication & Verification will take place and thereby access will be either approved or denied. This functionality is only available for registered users. If customers are not registered, they can easily create an account.

Customer Dashboard

Properties:

- Place Delivery Order
- View Order History
- Manage Account
- Sign out

When Customers are Authenticated, they will be able to see the lists mentioned above.

Place Delivery Order
When customers click on 'Place Delivery Order', they are taken to a page where they will be asked to provide all necessary information about the Receiver and Package information. They will also be able to choose delivery Methods available for their delivery. After providing the necessary information, they will be able to place their order and thereby checkout with payment. If successful, they will receive confirmation and there after they will be able to view their order. Whenever ready they can always navigate them self to the main Dashboard.

View Order History
When navigating to this page users can easily get access to a comprehensive list of their previous/past orders. This will help them to keep track and review transactions history.

Manage Account
When clicking on 'Manage Account', customers will be navigated to an Account Setting page, where they will be able to change their Email ,Home Address, postal Code and password. Future development of this page may contain features such as; Personal Information Update, Phone Number Management, Account Security Settings, Notification Preferences, Privacy Settings, Language and Regional Settings etc.

Sign out
Will clicking on Sign out, customer will be able to sign out.

Database Model

SQL Injection

SQL injection is a serious treat and serious security vulnerability. To secure our application from such treats and vulnerabilities, our group has taken the following steps:

- Avoid SQL strings
- Avoid Dynamic SQL
- Validation of user input

Our group has spent enough time understanding Data-Context and Crud-Operations to be able to implement the necessary features to create a safe connection and operation. We have used parameterized queries instead of SQL strings.

In addition, we have implemented input Sanitization, to eliminate treats and make sure that inputs are safe to be processed.

Our Shipping Management Application relies on a database to manage customers and their order and administrators. To coordinate these various components and provide effective functionality, our group has opted to employ SQLite database which is a Serverless relational database management system (RDBMS).

Reason for choosing SQLite as a database engine for our project[1]:

- Fast
- Self-contained
- High reliability

Fast:

SQLite is fast for writing and reading operations. This makes accessing and manipulating data very fast and easy.

Self-Contained:

SQLite is "stand-alone" or "self-contained" in the sense that it has very few dependencies. It runs on any operating system, even stripped-down bare bones embedded operating systems.[2]

High reliability:

As a storage solution SQLite is highly reliable. It does not give problems.[3]

We believe that using SQLite in our application will help us to get easy access to data and the fact that it`s faster than a file system will save us a lot of time.

Customer Table

The main purpose of this table is to store necessary details about our customers and consists of the following fields:

- CustomerId (LONG) (PK)(FK)
- Email (STRING)
- Name (STRING)
- Address (STRING)
- PostalCode (STRING)

[1] *SQLite, what is SQLite, (2023) , Available at : SQLite Home Page , (Last - Accessed 29/11/2023)*

[2] *SQLite, what is SQLite, (2023) , Available at : SQLite Home Page , (Last - Accessed 29/11/2023)*

[3] *SQLite, what is SQLite, (2023) , Available at : SQLite Home Page , (Last - Accessed 29/11/2023)*

Order Table

The purpose of the Order Table is to store details about our customers order and consists of the following fields:

- OrderId (LONG) (PK)
- CustomerId (FK)
- ShippingAddress (STRING)
- OrderStatus (STRING)
- Price (INT)

User Entity Table

The purpose of the User Entity table is to store data of users Username and Password for Login. This table consists of the following fields:

- Id (LONG) (PK)
- UserName (STRING)
- Password (STRING)
- Role (STRING)
- Discriminator (STRING)

Entity-Relationship Diagram

The purpose of the following Entity-Relationship Diagrams is to get an overview of our schemas in the database and to show relationships between our components (UserEntity, Admin, User, Customer, Order).

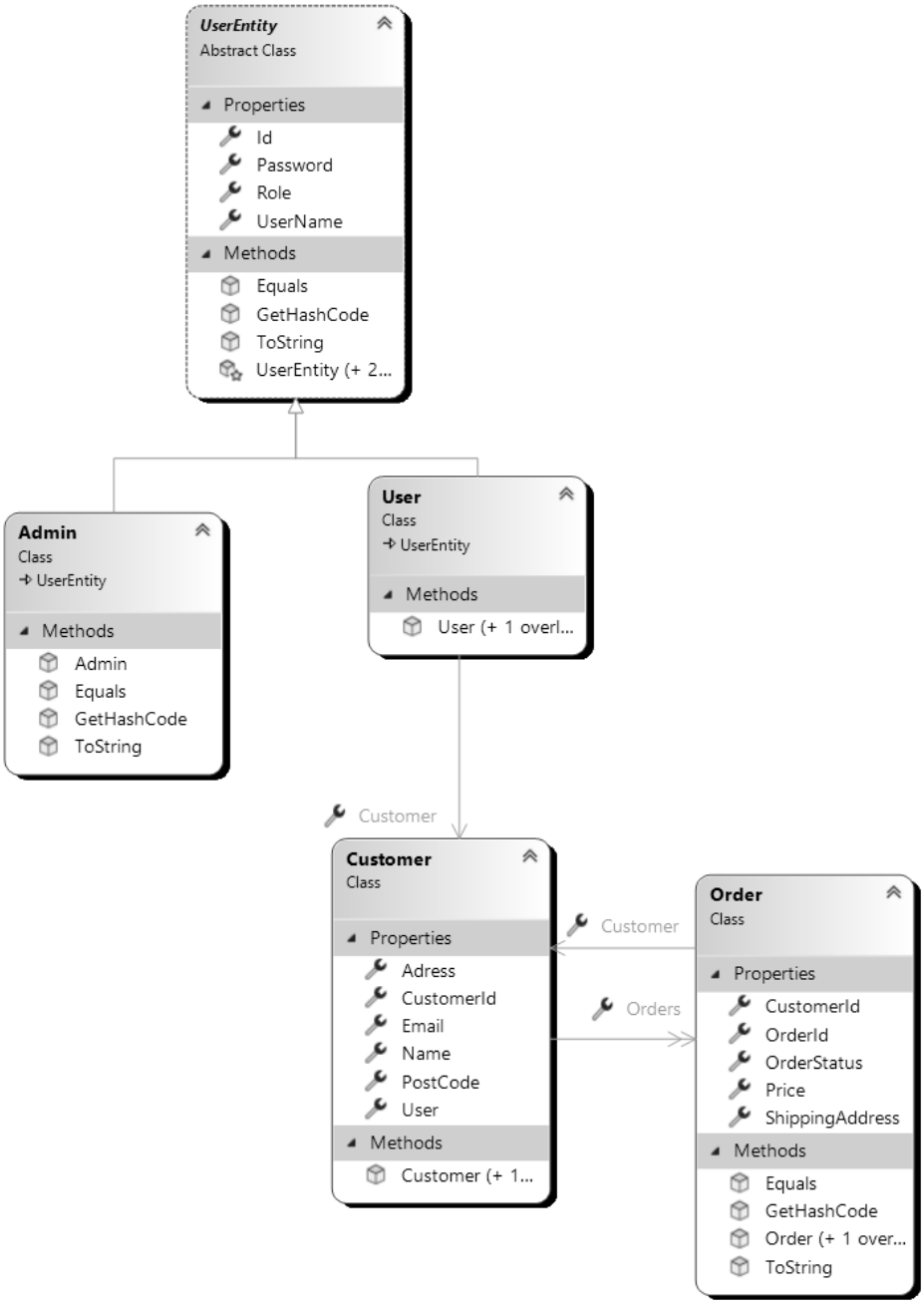


Figure V Entity Relationship Diagram

Charts and Diagrams

Entity-Relationship Model (ER)

The purpose of ER (Entity-Relationship Model) is to represent the conceptual structure of our application through a graphical approach. This graphical modelling approach aims to define the elements and their entities to help us build a good database system.

To enhance a better understanding of our ER (Entity-Relationship Model), we have chosen to separate the elements and their entities into sections. Rectangles in the diagram are entities while edges and diamonds indicate relationships. Entities and relationship attributes are described as circles.

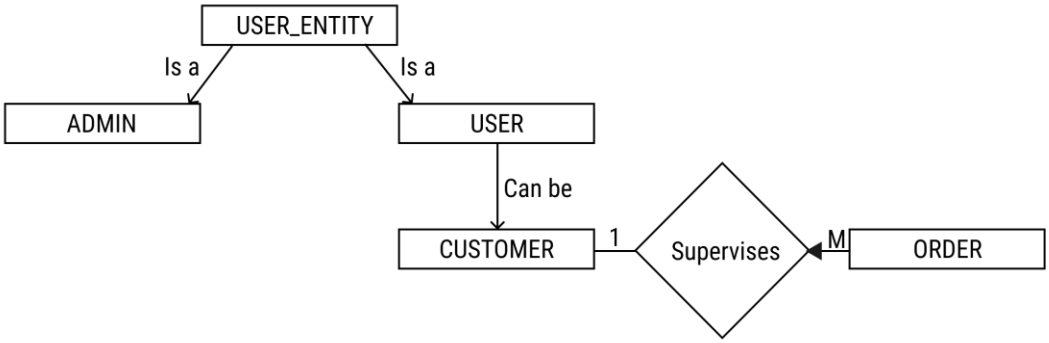


Figure VI Entity-Relationship Model of our Application

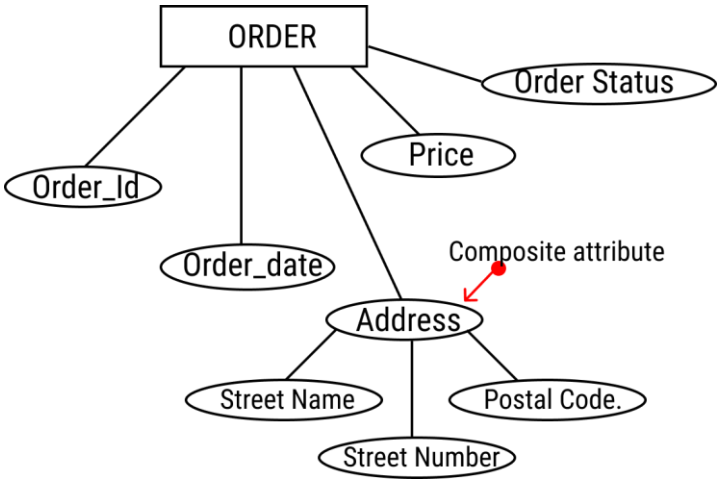


Figure VII Entity-Relationship Model - Order

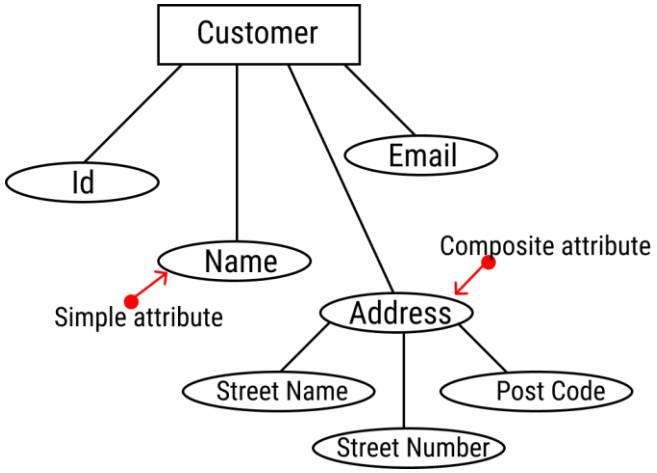


Figure VIII Entity-Relationship Model - Customer

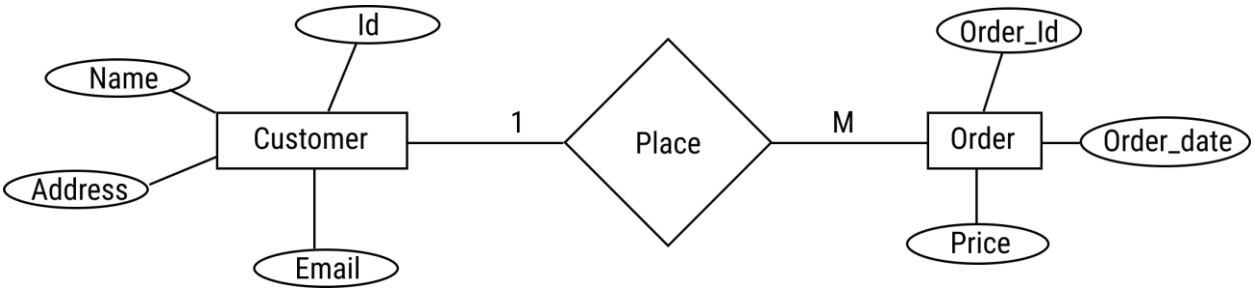
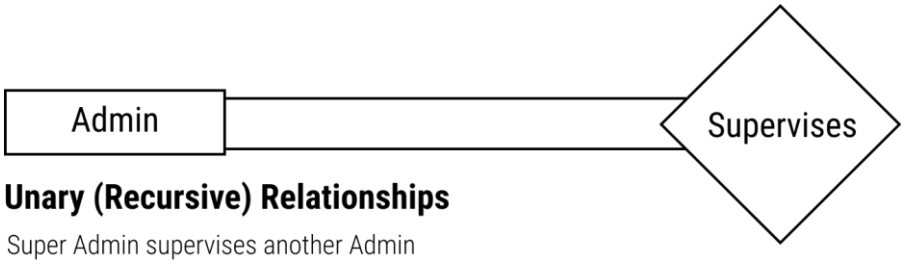


Figure IX Customer - Order Relationship



Unary (Recursive) Relationships

Super Admin supervises another Admin

Figure X Administrator Unary Relationship

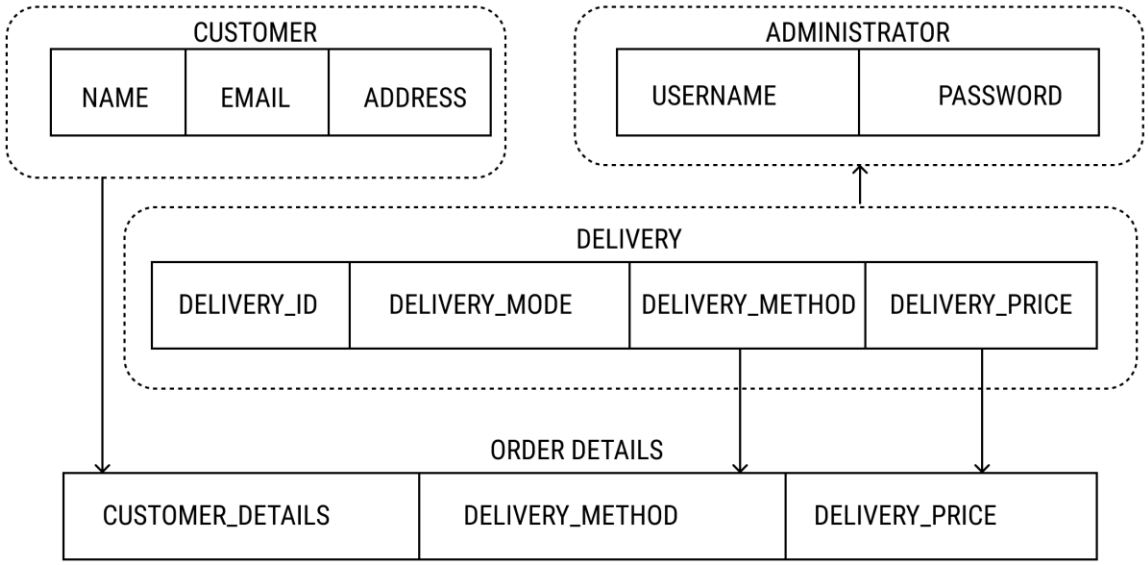


Figure XI Relationship Entity

Flowchart

The purpose of the Flowchart is to show the logical flow of our application including operations and procedures. A graphical representation of our Algorithm or Data will play a major role in helping us understand our application system better and nevertheless makes it easier for us to visualize our code before implementation. The flowchart presents a step-by-step solution to problems that occur during process run.

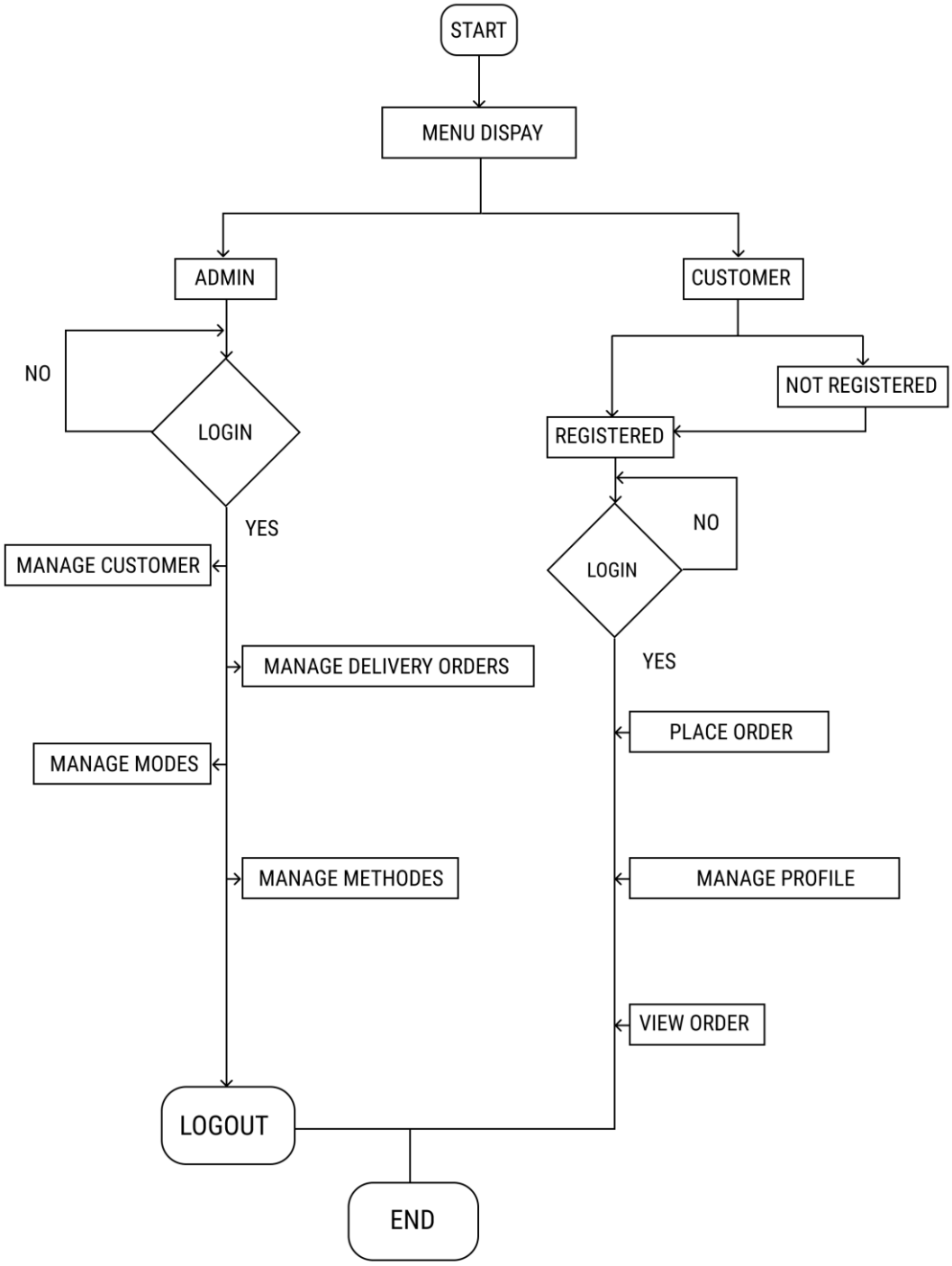


Figure XII Flowchart - Shipping Management Application

The following diagram illustrates the working mechanism of the customer side.

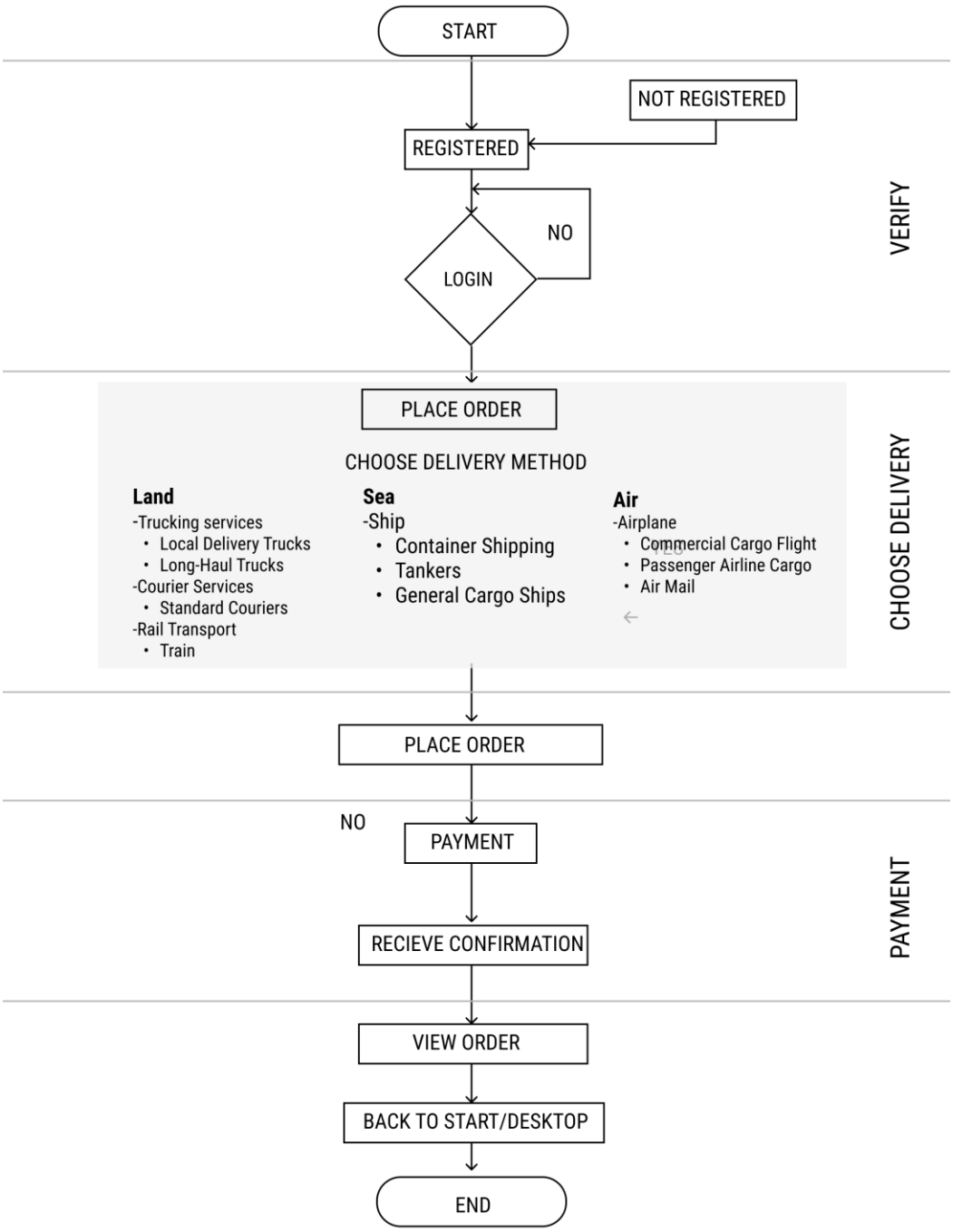


Figure XIII Working Mechanism of Customer

Sequence Diagram

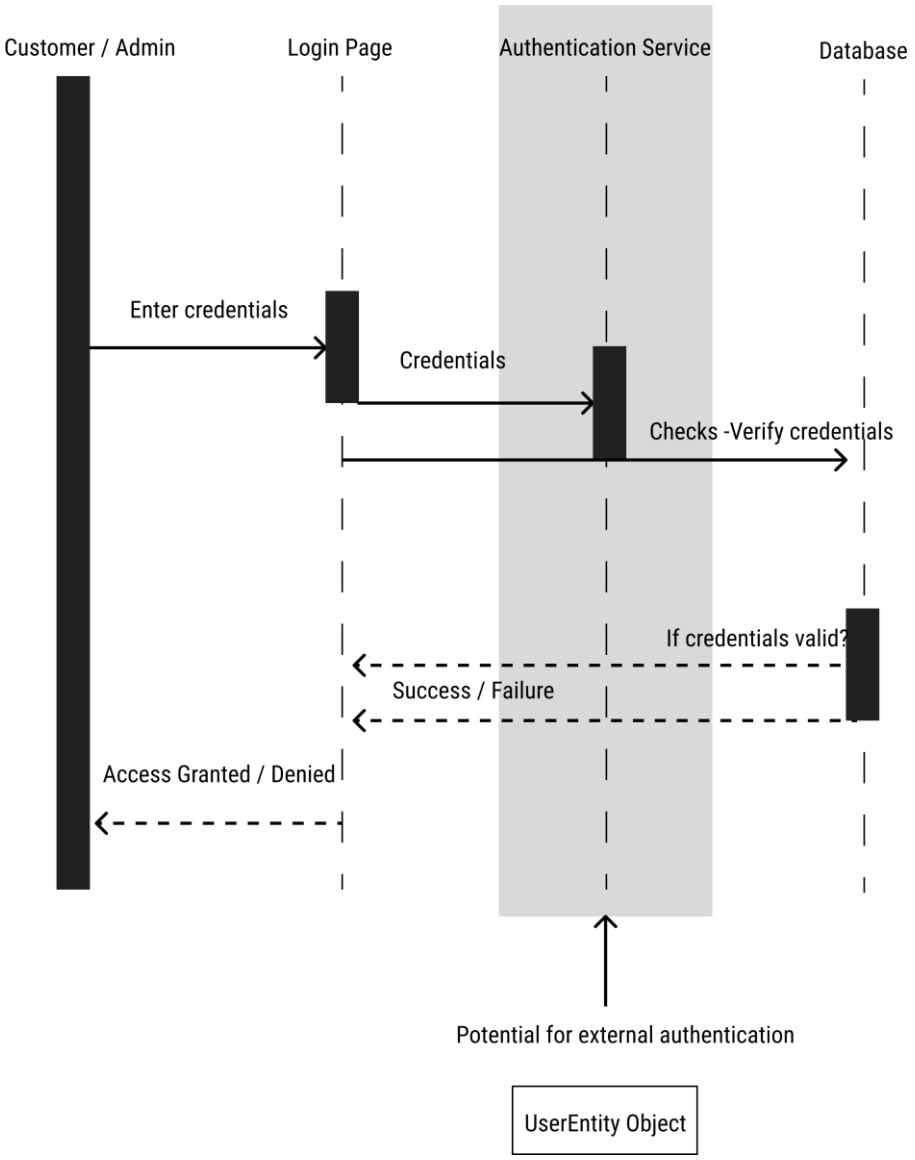


Figure XIV Sequence Diagram - Login

These schemas played a major role in designing our database schema and nevertheless help us understand how our data will be stored in our database.

UML Class Diagram

Design: The diagram specifies the structure of how our application software system will be written and function, without completing & implementation of our program. This phase is described as a transformation from “what” our application system must do, to “how” our application system will do it. [4]

The class diagram will give us a comprehensive overview of ;

- Classes to be implemented into our system.
- Fields and Methods in each class.
- Relation and interaction of classes

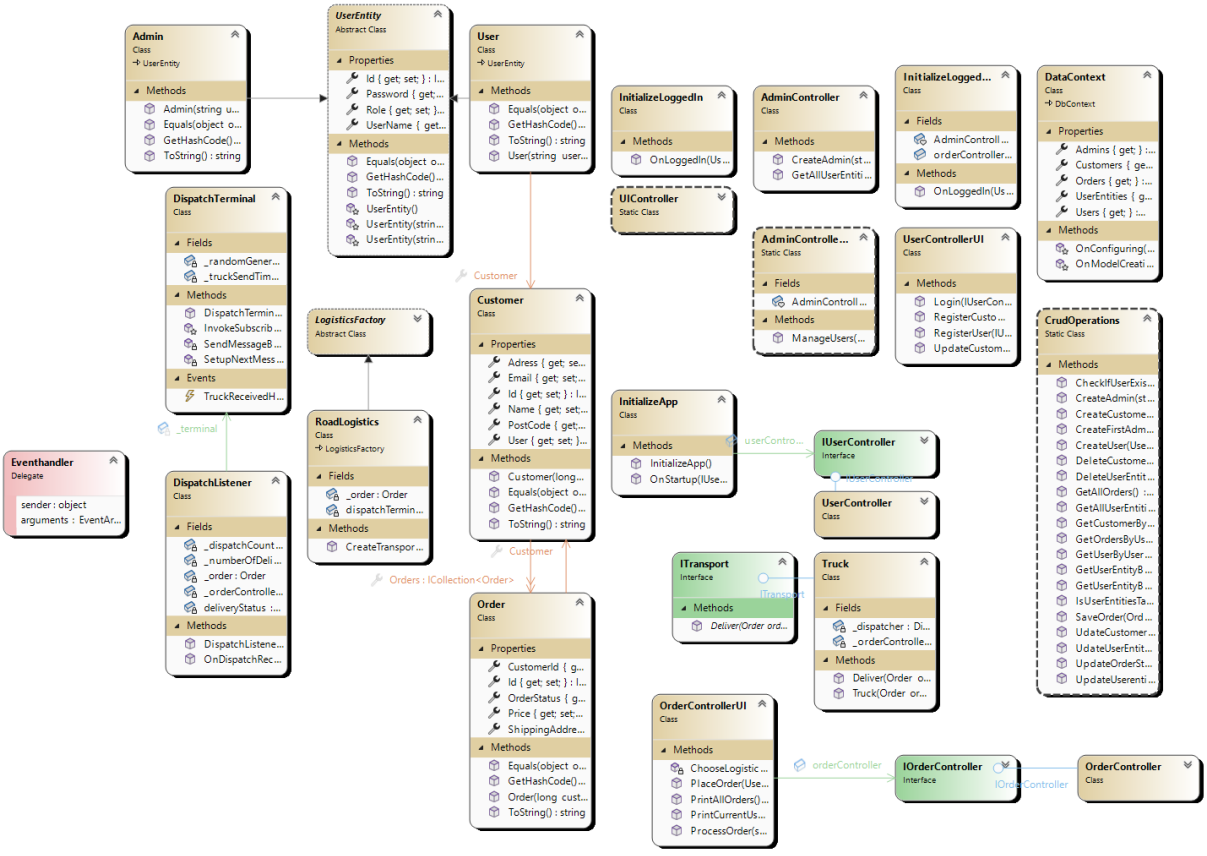


Figure XV Class Diagram

[4] *Torlak, (2016) UML Class Diagrams. Available at : L07.key (washington.edu) (Accessed 25/11/2023)*

Coding Convention

This section is crucial for maintaining code readability and consistency. Nevertheless, makes collaboration within our group easy by offering comprehensive guidelines. Our group has chosen to follow industry practices and guidelines so that our code is easy to understand, to maintain and nevertheless to extend. [5]

Tools and analyzers

To enforce the common code standards in our project we have chosen to use code analysis in Visual studio. By enabling this tool we will allow our group to work effectively and implement the rules. This tool will provide warnings, diagnostics and rule violations that can help us to enforce suitable standards for our project and thereby ensure that our code is readable, maintainable, and consistent.

Language Convention

Language conventions provide the best possible practices in .NET development and are therefore not only instructive but also help us obey the rules of contemporary programming. In our project we have exerted our utmost effort in applying the following bullet points.

- Use language keywords for data types rather than runtime types. IE “int” rather than “System.Int32”
- Try to avoid implicit types when possible.
- Try to utilize specific exception types.
- Interact with the database through LINQ statement programmatically rather than SQL statements.
- Keep layers separate both within the code and folder structure.
- Indentation size is 4 and type is spaces.
- New line format is “\r\n”(?)

See list of Language guidelines provided by Microsoft here.[6]

[5] *Microsoft, Common C# code conventions, 2023. Retrieved November, 11, 2023* .NET documentation C# Coding Conventions - C# | Microsoft Learn

[6] Microsoft, Language guidelines, 2023. *Retrieved November, 11, 2023* .NET documentation C# Coding Conventions - C# | Microsoft Learn

Naming Conventions

C# documentation and the .NET Framework design guidelines provided by Microsoft has been used as a guideline in this project with the aim to write a readable, maintainable, and consistent code. The following conventions are to be implemented and are best practices for naming in our C# console-based application.

- Directories: To adhere to the C# namespace and class naming rules, we have applied Pascal Case for folder names
- NO SINGLE CHARACTER NAMES OR ACRONYMS/ABBREVIATIONS IN NAMES UNLESS APPROVED BY ENTIRE GROUP except in lambdas
- Interfaces starts with I and uses Pascal Case
- Classes and methods use Pascal Case
- Class names are singular, not plural, and reflect a single instance of the object they return
- Method names follow a Verb Noun structure that describes what the method does.
- Method parameters, local variables and private fields use camelCase
- Private instance fields use camelCase and have an underscore prefix
- Microsoft suggests prefixing static fields with s_, I have no preference for this so we can decide whether to use it together.
- Reverse domain name notation for namespaces
- No double underscores in identifiers
- Names should be in English.
- Test Classes: Name of the class being tested followed by “Tests”

Syllabus Topics

Techniques and Tools for Collaboration

This section of the document describes the Techniques and Tools our group have used in portraying the technical information that must be documented during the development of the project.

Tools for Communication and Management

Choosing the right tools for communication and management is very important and is critical for effective project development and collaboration. Our group has utilized different packages/tools to streamline the project process.

- Discord:
  For real-time communication, Discord was our primary choice.

- Version Control with GitHub:
  We have opted for GitHub Version Control to manage and track our code and have absolute control over our codebase.

- Trello and GitHub Project board:
  Both Trello and GitHub offer flexibility and are Dev-friendly. But since GitHub offers better Security features, we have selected the GitHub Project Management board as primary. Since Trello offers Third-Party Integrations, we have kept it within the radar.

- Google Workspace:
  Our group has chosen to use Google Workspace, particularly Google Document for collaborative document editing. It facilitates the way feedback is exchanged within the group and makes a real time document update and editing effective.

- Figma:
  Figma is a collaborative Interface Design Tool, which we have chosen to use in this project to produce our own Illustrations, Diagrams and Charts.

Design Patterns & Principles

These sections of the documentation will briefly explain what types of design principles and design patterns our group has chosen in this project and provide a comprehensive answer to the pivotal question:" Why were these methods chosen?".

Selection of Design Patterns

Pertaining to the essence of Software Development, awareness of any obstacles and finding a way to handle them is a crucial aspect of programming.

Our group has come across challenges such as how to handle complexities in object creation, especially since our Delivery application system has several types of delivery "Modes" and "Methods" each with specific elements. This issue has been resolved by using abstract Factory Design Pattern.

This allowed our group to implement an easy way to initialize objects of different types while keeping creation separate from function. Adding several methods will allow the code base to be extended without modifying the existing code and allows us to utilize to keep our code ***"DRY"***. This in many ways will encourage scalability and Flexibility as the application evolves and several processing methods might need to be added in the future. For this reason, the abstract Factory Design pattern is the best candidate for our project.

Implementation

As an example, we have added the following code snippet. More details have been applied into our codes with a comprehensive comment.

```
using Shipping_Management_Application.Factories.Transport;

namespace Shipping_Management_Application.Factories.Logistics
{
    public abstract class LogisticsFactory{
        public abstract ITransport CreateTransport();

        public int DeliveryCost(string address){

            if (string.IsNullOrEmpty(address)){
                throw new ArgumentException("You need an address to calculate the
delivery price");
            }

            string numbersInString = FetchNumberFromAddress(address);

            if (string.IsNullOrEmpty(numbersInString)){
                throw new InvalidOperationException("There are no numbers in your
address");
            }

            int convertAddressNumber = int.Parse(numbersInString);
            int price = 100;
            return convertAddressNumber * price;
        }

        public string FetchNumberFromAddress(string address){
            string number = "";

            foreach (char c in address){
                if (char.IsDigit(c)){
                    number += c;
                }
                else if (number != ""){
                    break;
                }
            }
            return number;
        }

    }
}
```

```
namespace Shipping_Management_Application.Factories.Logistics
{
    public class RoadLogistics : LogisticsFactory
    {
        public override ITransport CreateTransport(){
           return new Truck();
        }
    }
}
```

Selection of Design principles

Our group has chosen to apply the SOLID principle into our project. Implementation details are represented as follow:

SOLID Principles [7]

Single Responsibility Principle (SRP)

We adhere to the Single Responsibility Principle to ensure that each class in our application focuses on one singular functionality and one thing only. This approach creates a clean and maintainable codebase where each class or object has one explicit responsibility - Our strategy is to keep track of where the different functionalities are supposed to be, ensuring that every class remains within its designated domain or layer. By using the Single Responsibility Principle, it simplifies the understanding and managing of each class but also enhances the scalability of our application.

Open/Closed Principle (OCP)

We have designed our entities in the application to be extendable without having to alter the existing codebase. This approach is important to maintain a stable codebase - We initiate shared interfaces for entities that have the same implementation among the application to make sure other new functionalities can be added without hesitation.

Liskov Substitution Principle (LSP)

The Liskov Substitution Principle guarantees that the subclasses can be substituted for their super-classes without affecting the behavior of the application. 17 - We design our subclasses to ensure that they cannot be changeable with their super-classes.

[7] Sahbaz, Comprehensive Guide to Solid Principles in C# , Comprehensive Guide to SOLID Principles in C# | by Edin Šahbaz | Medium, Last accessed : 30/11/2023

Interface Segregation Principle (ISP)

The Interface Segregation Principle is applied to the application to make sure that our classes are not overloaded with unnecessary methods or fields. This principle plays a major role in creating efficient interfaces - Our goal is to create specific interfaces that allow classes to interact smoothly without being hindered by other functionalities that do not have anything to do with it.

Dependency Inversion Principle (DIP)

The Dependency Inversion Principle has the importance of how we think about the design of our application. In other words, this means that the big picture functionalities and the more detailed ones of the application should use general concepts and not specific ones. This makes the code more flexible for testing - In our code, we use interfaces and abstract classes to keep the main functionalities of our application separate from the smaller detailed parts.

DRY Principle

A place where we have applied this principle is through the Crud Operations class. By having this class, we allow others to reuse and maintain these components.

Open/Closed Principle (OCP)

Classes, modules, and functions in our project are open for extension but closed for modification.

Development process

This section of the documentation will briefly describe our development process. During the initial phase of this project, our group members got the chance to brainstorm and share potential application concepts for development. Each member of the group handed in their proposal and after a brief meeting and discussion , final proposals were chosen. All members voted to move forward with “Shipping Management Development System”. The chosen proposal has its origin from one of the Lecture in Software Design – PG3302. NB! The general code implementation from the lecture covers only a basic skeleton with simple Abstract Factory Methods for Transport. This code snippet was provided by the lecturer to demonstrate the Abstract Factory method.

Our group sees the potential for further development and integration of this application and therefore decided to take the proposal as a challenge. Nevertheless, the proposal appeared to be suitable for our academic examination:

Proposal suggestion from the group is presented as follows:

- Person-1: Order Management System for logistic
- Person-2: Clinic Management System
- Person-3: Auction Application
- **Person-4: Shipping Management System (Voted)**
- Person-5: (Didn`t have any suggestions, but was involved in the process)

Design Phase

To ensure productivity by pushing the Design Phase one step further our group chose to start working on a Class Diagram representing the proposal. Using tools mentioned above in the **Tools for Communication and Management** section, the group continued the collaboration process both online and in campus at Urtegate 9.

Online meetings are held 3 times a week. While it was necessary to conduct 1 physical meeting at Urtegate 9 to go through documentation and exchange of ideas to strengthen the development process of the Application.

As a result, the following Software Design Document (SDD) & Technical Design Documentation (TDD) has been produced with the aim of developing a fully functional application.

- Use Case Diagram
- Working Mechanism Diagram
- Sequence Diagrams
- Flow Chart
- Entity-Relationship Model
- Relationship Entity
- UML Class Diagram

The following sections of the documentation will give a comprehensive description of the SDD & TDD.

Layering

Modularity By dividing the application down into a three-tier[8] design architecture and into their own respective sections(presentation, business layer and data access layers), allows us to test each individual section without having to include the rest. LEGO blocks have the same concept. We can

[8] What is three-tier architecture, IBM , What is Three-Tier Architecture | IBM , Last accessed 30/11/2023

focus on creating, modifying, and troubleshooting each block at a time until everything meets the standards.

Scalability

The three-tier design architecture is split down to the user interface, business logic and the data storage. Having this setup gives the possibility for each part to be 18 scaled and tested individually based on the requirements. If there is a time where the application needs to be improved, ex. in the user interface layer where a lot of users come in at once, then we can simply add more resources to that layer without having to bother the rest.

Maintainability

Maintainability ensures that the software system is available to be modified and correct its faults, thus improving the application's performance. Having a high maintenance process such as updates and patches in a real life or local application, makes things go faster and more efficiently. Not only that, but it also minimizes downtime and makes sure that the software delivers at the right time.

Reusability

In terms of reusability, having code that can be used multiple times is genuinely a good practice. This can for example be classes and methods that have been tested which can be used in other places of the software. This gives the programmers less time to think about new possible errors that can be introduced and reduces development time.

Adaptability

All users have different needs, which makes adaptability come into play. Adaptability makes sure that the software adjusts to its users' requirements.

Robustness

Robustness enables the software to act based on unexpected inputs or outputs and handles those without having to shut down the software. This happens a lot in real life, where users either give unexpected inputs and outputs by mistake or intentionally, and the software handles those with profession 19 4) Design Pattern.

Testing

Testing of code was a crucial step to ensure our application`s Quality, Functionality and Reliability. For this project our group has chosen to focus on both manual and automated testing methods, which we think has given us both control and overview of our testing methods due to the methods flexibility and intuition.

The testing is conducted by applying the NUnit Testing Method. This allowed our group to check the business logic from each component in isolation and ensure that each codebase functions correctly and cohere to expected behaviors. Following that, we have implemented an Integration Test where we evaluate the interoperability of the application`s various components and pinpoint any problems that may result from their interaction.

Our group has also conducted a Manual Testing Method to assure the UI(User Interface) and UX (User experience) components. The focus on implementing these testing methods for UI and UX is to be able to see our application from customer view/perspective and reassure that both the UI and UX meet our standards. More details of the Testing Method , you will find in the “Shipping_Management_Application.UnitTest.cs” folder.

Taking everything into consideration our project`s /Application`s testing strategy aims to find and address defects at early stage, enhance the caliber of the code , and provide the Applications users with a quality, functional and stable Application.

Modern Features in C#

We've used default values for both properties and method arguments for setting commonly recurring values as well as allowing for nitty usage of the Null-coalescing assignment, see UIController. ReadAStringInput for example.

Nullable reference types are used largely when we want to handle possible null references elsewhere. Null-coalescing assignment is used primarily in the Crud-Operations class for exception throwing.

we chose to primarily use properties over fields, since they are easier to share in safe manner. We also used the setters for updating objects before passing them to the database, especially for update operations. Events in our applications are used in EventDispatch and DispatchListner.

we used the required keyword for must have members in classes, EG. username. We also used the init access modifier for username since it should never change. We used interpolated strings for

writing strings containing values to the console. Not sure if LINQ, lambdas and EFcore counts, but we've used these as well.

Refactoring

To improve our application's readability and efficiency we have modified our internal codebase. By refactoring our components in our application, we have achieved the following points:

- Enforce single responsibility principle
- Enforce open-closed principle
- Separation of layers, pre refactoring we had all three layers in one method, post refactoring we have one, and bindings to the other layers
- Reduce redundant in our code, namely concerning handling multiple inputs
- Made our code more modular, one of the methods we split out is also use in another place in the codebase

As an example: The following code snippet from the UserController class illustrates implemented refactoring.

Before refactoring

```
public void UpdateCustomer(UserEntity user)
{
    Customer customer = InitializeApp.userController.GetCustomer(user);
    Console.WriteLine("You can update your profile");
    Console.WriteLine("What do you want to update? (Email, Address, PostCode, Password)");
    string? res = Console.ReadLine();

    if (string.IsNullOrWhiteSpace(res))
    {
        Console.WriteLine("Invalid input, please try again");
        return;
    }
    try
    {
        //TODO: se if we can dry this off
        //if (res is "email" or "address" or "postcode" )
        //{
        //    using DataContext context = new DataContext();
        //    Console.WriteLine($"enter new {res}");
        //    string input = Console.ReadLine();
          //    context.Customers.First(c => c.Id == customer.Id).Valueof(res) =                    input;
        //}
        switch (res.ToLower())
        {
            //lots of repeated lines
            case "email":
                Console.WriteLine("Enter the new email:");//this was in the business logic layer, which
                    should not write to the console
                string? newEmail = Console.ReadLine();
                    customer.Email = newEmail;//direct access to database, which should only happen in
                    the data layer
                Console.WriteLine("Email updated successfully");
```

```
                break;
            case "address":
                Console.WriteLine("Enter the new address:");
                string? newAddress = Console.ReadLine();
                customer.Adress = newAddress;
                Console.WriteLine("Address updated successfully");
                break;
            case "postcode":
                Console.WriteLine("Enter the new postcode:");
                string? newPostCode = Console.ReadLine();
                customer.PostCode = newPostCode;
                Console.WriteLine("Postcode updated successfully");
                break;
             case "password"://this should be discrete so we can reuse it for other user entities
                Console.WriteLine("Enter the new postcode:");
                string? newPassword = Console.ReadLine();
                user.Password = newPassword;
                Console.WriteLine("Postcode updated successfully");
                break;
            default:
                Console.WriteLine("Invalid option");
                break;
        }

    }
    catch (DbUpdateException)
    {

        Console.WriteLine("something went wrong");
    }
}
```

After refactoring

```
        /// <summary>
        /// After refatoring
        /// The method has been seperated with multiple methods in their appropriate layers
        /// this is the UI part, it handles input from user and then sends to the buisness logic layer
        /// <see cref="UserController.UpdateCustomerWithValues"/>
        /// </summary>

        public static void UpdateCustomer(IUserController userController, UserEntity user)
        {
            Customer customer = InitializeApp.userController.GetCustomer(user);

            List<string> options = new List<string>()
            {

                "What do you want to update? (Email, Home Address, Post code, Password)"

            };
            List<string> validInput = new List<string>()
            {
                "Email",
                "Address",
                "PostCode",
                "Password"
            };
            string inputOption = UIController.MenuFacade(options, validInput);
            Console.WriteLine($"please enter new {inputOption}");
```

```
            string inputValueToChange = UIController.ReadAStringInput();
            if(inputOption is not "Password" && customer is null)
            {
                Console.WriteLine("You need to register as a customer to manage customer profile");
                return;
            }
            else if (inputOption == "Password")
            {
                userController.UpdateUserEntityPassword(user, inputValueToChange);
            }
            else
            {
                userController.UpdateCustomerWithValues(customer, inputOption, inputValueToChange);

            }




    /// <summary>
    /// after refactorin cont.
    /// the business logic layer has seperated update customer and update userentity
    /// this lets us reuse the update userentity method for admin and any future userentities
    /// <see cref="CrudOperations.UpdateUserentity"/>
    /// </summary>

    public void UpdateCustomerWithValues( Customer customer, string option, string value)
    {
        switch (option.ToLower())
        {

            case "email":
                customer.Email = value;
                break;
            case "address":
                customer.Adress = value;
                break;
            case "postcode":
                customer.PostCode = value;
                break;
        }
        UpdateCustomer(customer);
    }

    public void UpdateUserEntityPassword(UserEntity user, string value)
    {
        user.Password = value;
        CrudOperations.UpdateUserentity(user);
    }

}



/// <summary>
/// refactoring end
/// here the database update happens, in the data layer as it should be
/// </summary>
/// <param name="user"></param>
public static void DeleteUserEntity(UserEntity user)
{
    using DataContext context = new DataContext();
    context.UserEntities.Remove(user);
    context.SaveChanges();
}
public static void UpdateUserentity(UserEntity user)
{
    using DataContext context = new DataContext();
```

```
    context.UserEntities.Update(user);
}
```

By refactoring our code components, we managed to construct a Monolithic structure and got rid of unnecessary redundant in our code.

Collaboration

Our collaborative approach to build "Shipping Management Application" has been effective as we acknowledge diversity in knowledge, attitudes, skills and experience, whose integration makes it possible to offer rapid, flexible, and innovative responses to problems and challenges.[9]

However, we had our challenges during the collaboration period, which we believe are handled at early stage and our group were able to move forward without consuming mental and emotional energy.

Challenges Encountered & Lessons Learned

Some of the challenges we faced are :

- Subgroup meetings
- Working in a silo

Subgroup meetings

The failure to properly use established communication channels had led to members of the group not getting meetings memo. This issue was identified at an early stage and was resolved by holding a group meeting. As a solution; Group members came to an agreement to eliminate subgroup meetings and made a weekly schedule planner for alignment meetings.

Working in a silo

Refactoring the project file without the presence of all group members has led to a conflict. This action has been resolved by holding meetings and a full review of both project codes. After a rigorous code review, the group came to an agreement on choosing one project and enforced strict

[9] *Rico, Alcover de la Hera, Tabernero (2011) ,Work Team Effectiveness, a Review of Research From The Lase Decade (1999-2009), Available at: untitled (jibc.ca).*

GitHub guidelines to avoid future conflicts. The group came to acknowledge *"…production of knowledge depends to a large extent on the effectiveness of teams"*[10]

GitHub & git

Our group has utilized Git for tracking versions of our Source code in our Repository and GitHub for version control and sharing our Repository for collaboration. To avoid Merge conflict, code Inconsistency ,overwriting codes, bad review processed and overlap, our group have developed a clear guideline to keep every member of the group on track.

The guideline involves the following points:

- Repository is to be kept PRIVATE *(Should not be changed or shared with others outside the group)*
- NEVER PUSH TO THE FINAL BRANCH.
- Any update to our Source Code should be pushed through a new branch.
- Branches will be reviewed as a group before any Merging occurs.
- After Merge, the updated branch will be cloned into local storage.

By the end of the month, our group had created 13 branches and kept our Final Branch safe. This process had helped our group in resolving and restoring code related issues with no time.

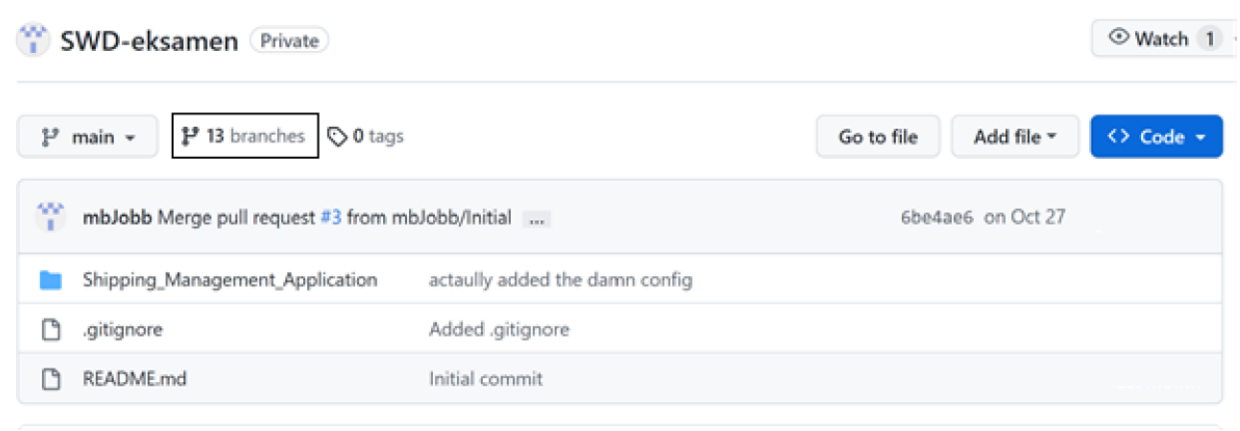


Figure XVI GitHub Repo- Screenshot

What didn`t work?

To manage our C# Application project, we had developed a project[11] task-board to plan and track our project in GitHub. After trying it`s functionality, the group decided to plan and keep track of our

[10] *Wuchty, Jones & Uzzi,(2007) , Retrieved November 25/11/2023.*

project using manual tools, such as Google docs and Discord. As mentioned earlier, our group held a regular meeting, and each meeting was opened by each member going through what has been accomplished based on the general task created after each meeting.

Conclusion

As we approach the completion of the PG3302 - Software Design examination project, it is vital to consider the various aspects of our work. This examination project has given all members of this group a great opportunity to apply theoretical concepts and learn more about the complexities of Software Design.

By using the strategies mentioned above in this documentation, our group managed to handle the challenges of collaboration throughout the project. The diversity in knowledge within our group has contributed a great number of viewpoints, suggestions and plans to the table, resulting in an all-encompassing learning experience.

Our aim and tireless effort in implementing a 3-layer architecture, had a locked target on developing a reliable and effective application system. Our group has made every effort to achieve just that and deliver a final product.

To conclude, every element of this project has been extremely educational and has taught us the importance of theory , practice, and collaboration in Software Engineering. We have gained a solid basis for future development activities and nevertheless learned how to be a good team player.

[11] *GitHub, About Project, Retrieved November, 27, 2023, About Projects - GitHub Docs*

[12] *External resource used for inspiration: (PDF) Online Food Ordering Management System (researchgate.net)*