## Logos - London Metropolitan UniversityCourse Submission Cover Sheet

Module: CS6004ES Application Development

Assignment no: 001

Weighting: 30%

Deadline: TBC

**Module Leader: Mr.** **Rattapol Kasemrat** **Student ID: 681347308200002**

Please note that there are specific regulations concerning **the use of AI and Academic Misconduct**. Below are extracts from these regulations. By signing, you acknowledge that you have read and understood these extracts.

(signature:) A black line in a white background

AI-generated content may be incorrect. Date: 19/7/2025

This header sheet should be attached to the work you submit.

Academic Integrity means being honest in your academic work and your studies and making sure that you acknowledge the work of others and giving credit where you have used other people's ideas as part of presenting your arguments. Your assessment submissions must therefore always be entirely your own work, based on your own learning and appropriately referenced including how you have used Generative AI. The University regards the use of Generative AI applications by students to deceive to gain unfair advantage as **academic misconduct**. This usage includes:

* **Plagiarism**, where AI tools are used to generate output and ideas that are presented or submitted as if they were the student's own work, without proper citation or references.
* Where a complete assignment is created using Generative AI and represented as a student's own work, this will be regarded as contract cheating in the same way as commissioning an 'Essay Mill' or other third party to complete your work. Further information can be found on : [Guidance on the use of Artificial Intelligence.](https://student.londonmet.ac.uk/your-studies/student-administration/guidance-on-the-use-of-artificial-intelligence/)

**Academic misconduct:** The University takes academic misconduct very seriously and seeks at all times to rigorously protect its academic standards. Plagiarism, collusion and other forms of cheating constitute academic misconduct, for which there is an explicit range of graduated penalties depending on the particular type of academic misconduct. The penalties that can be applied if academic misconduct is substantiated range from a reprimand to expulsion in very serious cases and for repeated instances of misconduct. You are also responsible for ensuring that all work submitted is your own and that it is appropriately referenced. The University does not tolerate cheating of any kind. You are strongly advised to familiarise yourself with the Academic Misconduct Policy and Procedure ([Academic Misconduct](https://student.londonmet.ac.uk/your-studies/student-administration/rules-and-regulations/academic-misconduct/)), which list a range of categories of academic misconduct and associated penalties, covering instances of academic misconduct (plagiarism, collusion, exam cheating).

Contents

[Course Submission Cover Sheet 1](#_Toc203741825)

[Introduction 5](#_Toc203741826)

[Database Design 5](#_Toc203741827)

[Microsoft Server Management Studio 6](#_Toc203741828)

[Software Development 7](#_Toc203741829)

[Software Architecture 8](#_Toc203741830)

[Detailed Instructions to Run the Program 8](#_Toc203741831)

[Custom-Developed Classes 10](#_Toc203741832)

[**Model Classes** 10](#_Toc203741833)

[**Assistant Model Class** 10](#_Toc203741834)

[**Driver Model Class** 12](#_Toc203741835)

[**EditUserRoleView Model Class** 13](#_Toc203741836)

[**Feedback Model Class** 15](#_Toc203741837)

[**Orderline Model Class** 16](#_Toc203741838)

[**TransportUnit.cs** 17](#_Toc203741839)

[Controllers 18](#_Toc203741840)

[**Driver Controller** 18](#_Toc203741841)

[**UserController** 20](#_Toc203741842)

[**FeedBacksController** 20](#_Toc203741843)

[Customer View 21](#_Toc203741844)

[Admin View 25](#_Toc203741845)

[Reflection 36](#_Toc203741846)

[References 36](#_Toc203741847)

**Deliverables**

Your submission should include the software project and a reflective essay as described below.

1. Your software artefact in the form of a Visual Studio 2015 project, which should include the program’s source code, compiled classes, the executable file and data file (if any).
2. A reflective essay (1000 or more words), which concisely documents:
   1. Detailed instructions to run the program
   2. The architecture of your software in terms of software classes, clearly indicating which classes to be of your own work and which classes from other sources (e.g., From textbooks, online sources such as MSDN etc.).
   3. Detailed description of the classes’ properties and methods
   4. Your reflection of own experience of using c# and visual studio for the development task, which feature you like and why, what issues you experienced and your solution to overcome it.

**Marking Scheme for the CS6004ESIndividual Coursework**

This individual coursework counts for 30% of the module mark. The following are guidelines for marking. Mark each item listed below on a scale 0 to 5 where the marks correspond. Then multiply the mark by the weighting indicated, total and divide by 2 to get the total mark.

|  |  |
| --- | --- |
| Mark | Characterised by |
| 0 | No work or work totally irrelevant |
| 1 | Work started on right lines but no result |
| 2 | Some result, with major lack and/or errors |
| 3 | Acceptable result but incomplete, or some good result with minor errors |
| 4 | Good result but can be further polished |
| 5 | Excellent result |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Item** | **Weight** | **Marks** | **Weight X Marks** |
|  | **IMPLEMENTATION** |  |  |  |
| 1 | The application user interface | 2 | 5 | 10 |
| 2 | Task1: Customer details and Management | 2 | 5 | 10 |
| 3 | Task2: Only Admin Can Manage Admin Details. | 2 | 5 | 10 |
| 4 | Task3: Admin Login and Customer Login. | 3 | 5 | 15 |
| 5 | Task4: Customer Transport operation detail Management. | 2 | 5 | 10 |
| 6 | Task5: Only Admin can Accept/Decline and update Customer Transport operation. | 3 | 5 | 15 |
| 7 | Task6: Customer Dashboard. | 3 | 5 | 15 |
| 8 | Task7: Admin Dashboard. | 2 | 5 | 10 |
| 9 | Task8: Only Admin can manage Product details. | 2 | 5 | 10 |
| 10 | Task9: Only Admin can generate various reports | 2 | 5 | 10 |
| 11 | Task10: Admin Dashboard | 2 | 5 | 10 |
|  | **DOCUMENTATION** |  |  |  |
| 1 | Detailed instructions to run the program | 1 | 10 | 10 |
| 2 | The software architecture | 3 | 5 | 15 |
| 3 | Detailed description of the classes’ properties and methods | 2 | 5 | 10 |
| 4 | Explanation about search algorithms used in the project | 2 | 5 | 10 |
| 5 | Reflection of own experience | 1 | 5 | 5 |
|  | PROGRAMMING STYLE |  |  |  |
| 1 | Clarity of code which shows the underlying algorithm | 1 | 5 | 5 |
| 2 | Sensible naming of programmer-defined variables, classes, properties and methods | 1 | 5 | 5 |
| 3 | Useful comments in code | 1 | 5 | 5 |
| 4 | Data validation and exception handling | 1 | 5 | 5 |
| 5 | Interface design and usability of the system | 1 | 5 | 5 |

# Introduction

This paper describes a new software system for e-Shift, a company that moves household goods. Right now, e-Shift handles everything by hand, which is becoming too difficult as they grow and have more customers. This new system will automate their daily tasks, making things smoother and faster. It will help them manage **customer details**, keep track of every **moving job**, and carefully follow each **load of goods**. The system will also help them know where their **lorries, drivers, and containers** are. Our goal is to build a system that helps e-Shift work better and make smart decisions, so they can keep their customers happy and grow their business.

# Database Design

The following is database table diagram. I used SQL Server Management Studio for database management. The AspNetRoles,AspNetUserRoles and AspNetUsers tables are built-in tables which are provided by ASP.NET Core Identity to assign roles for users. The other tables are as below:

* Orderline: Represents individual shipment orders, linking to a customer and containing details like addresses, date, weight, and status.
* Lorries: Contains details about the transport vehicles.
* Driver: Stores information about the drivers.
* Assistant: Stores information about the driver assistants.
* TransportUnit: Combines a Lorry, Driver, and Assistant to form a complete transport team for a job.
* Invoice: Records billing details, linked to Orderline and TransportUnit, including total amount and payment status.
* FeedBack: Stores customer feedback, linked to an Invoice.

**A computer screen shot of a computer

AI-generated content may be incorrect.**

Figure 1: Database Structure

# Microsoft Server Management Studio

Microsoft server management studio is used for database management. To connect server, first thing is to configure default connection. After that connect in the studio.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: appsetting.json

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Connect to Server

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Tables

# Software Development

I used git for software changes during development. Git is a version control system used to track changes in source code during software development. It allows multiple developers to collaborate, manage code history, revert to previous versions, and work on different features or fixes simultaneously through branches.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Github

# Software Architecture

I used Model-View-Controller (MVC) software architecture. A software design pattern called Model-View-Controller (MVC) divides an application into three primary components. The data and business rules are stored in the model. The View shows the data and the user interface. Clicks and form submissions are handled by the Controller, which also updates the Model or View as necessary. Because each component has a specific function, this structure facilitates code management, testing, and updating.

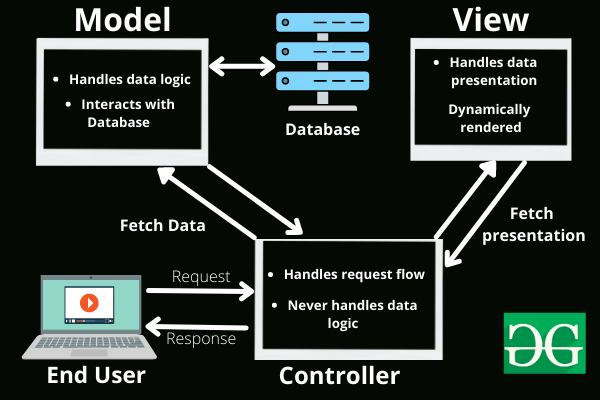


Figure 6: Model-View-Controller (MVC) architecture

# Detailed Instructions to Run the Program

To ensure the e-Shift application functions correctly, Visual Studio 2022 and .NET Framework 9.0 are required to be installed on the system. Installation of these prerequisites should be completed first.

Following the setup, these steps outline the process for running the application:

Step 1 - Open the Project:

The project files are located in a specific directory. Within this directory, locate the Visual Studio Solution file which is LogisticsWebApp.sln. Double-clicking this file will open the project within Visual Studio 2022.

Step 2 - Build the Project:

Once the project is loaded in Visual Studio, navigate to the "Build" menu.Select "Build Solution." This action compiles the source code and prepares the necessary executable files for execution.

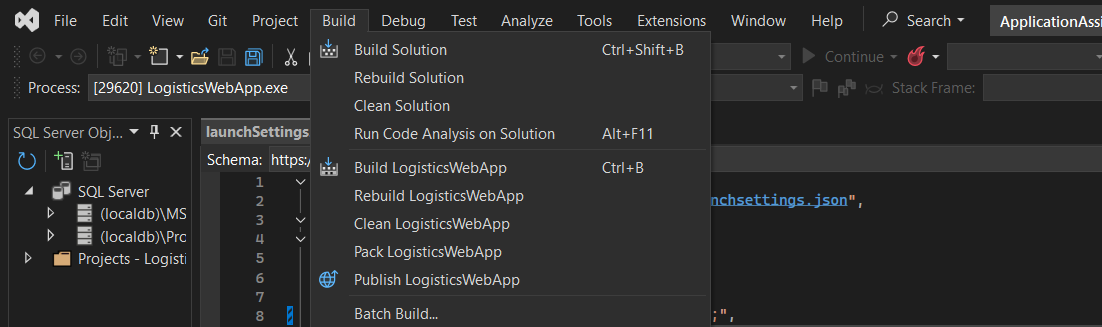


Figure 7: Build Solution

Step 3 - Run the Program:

After a successful build, the application can be launched directly from Visual Studio. Click the green "Start" button on the Visual Studio toolbar, or press F5 on the keyboard.

The application's main window will then be displayed.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: Run Application

Step 4 - Application Access:

No manual database setup is required, as the application manages its data persistence internally. For administrative functionalities and testing, the default administrator credentials are provided:

Admin Email: moses@gmail.com

Admin Password: Moses123!

New customer accounts necessitate registration through the application's provided interface.

**A screenshot of a login form

AI-generated content may be incorrect.**

Figure 9: New Customer Register

# Custom-Developed Classes

## **Model Classes**

### **Assistant Model Class**

The Assistantt Model class has properties for AssistantID, Name, Phone, and Age.

* AssistantID is a unique identifier.
* Name is required and can be up to 100 characters long.
* Phone is required, can be up to 20 characters long, and must be a valid phone number format.
* Age must be between 18 and 99.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 10: Assistant.cs

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11: Creating New Assistant

A user with manager role and edit, delete and see details about assistant.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12: Manage Assistants Dashboard

### **Driver Model Class**

The following are properties for Driver Model Class.

* DriverID is a unique identifier.
* Name is required and can be up to 100 characters long.
* Phone is required, can be up to 20 characters long, and must be a valid phone number format.
* Age must be between 18 and 99.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 13: Driver Model Class

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: Creating New Driver

In manage driver dashboard the manager can not only delete, edit but also find the driver by name or phone number.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 15: Driver Management Dashboard

### **EditUserRoleView Model Class**

The EditUserRolesViewModel class is used to edit user roles. It has these properties:

* UserId: to identify the user.
* UserName: displayed as "Username".
* Email: must be a valid email address and will be displayed as "Email".
* PhoneNumber: must be a valid phone number format and will be displayed as "Phone Number".
* AllRoles: A list of RoleSelection objects, used to show and select roles.

The RoleSelection class is used within EditUserRolesViewModel to represent a single role option. It has these properties:

* RoleName: The name of the role.
* IsSelected: A boolean indicating if this role is currently selected.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 16: **EditUserRoleViewModel.cs**

Only manager role can create new user and assign role.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17: Register New User Dashboard

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 18: Edit User Dashboard

### **Feedback Model Class**

This model class represents feedback information and includes these properties:

* FeedBackID: A unique identifier for the feedback, acting as its primary key.
* InvoiceID: A required identifier that links this feedback to an Invoice. It's a "foreign key."
* Invoice: This is a "navigation property" that allows direct access to the related Invoice object. It can be null.
* Rating: A required integer that must be between 1 and 5, representing a star rating.
* Message: An optional string for the feedback message, with a maximum length of 500 characters. The DataType.MultilineText suggests it should be displayed as a multi-line text area in a user interface.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 19: Feedback Model Class

### **Orderline Model Class**

The following is Orderline Model class. The class includes

* **OrderLineID**: A unique identifier for each order line.
* **CustomerID**: A foreign key linking this order line to a specific AspNetUser (likely a user registered in the system using ASP.NET Core Identity).
* **Customer**: A navigation property (of type IdentityUser) that allows the application to access details of the associated customer directly.
* **InitialAddress**: The starting location for the shipment.
* **DestinationAddress**: The delivery location for the shipment.
* **Date**: The date associated with the order line (e.g., order date, shipment date).
* **WeightKg**: The weight of the items in this order line, specified in kilograms, with a validation rule ensuring it's between 0.1 and 10000 kg.
* **Status**: The current status of the order line, defaulting to "Pending".

The attributes like [Required], [ForeignKey], [StringLength], [DataType], [DisplayFormat], and [Range] are data annotations used by ASP.NET Core for validation, database mapping, and UI display purposes.

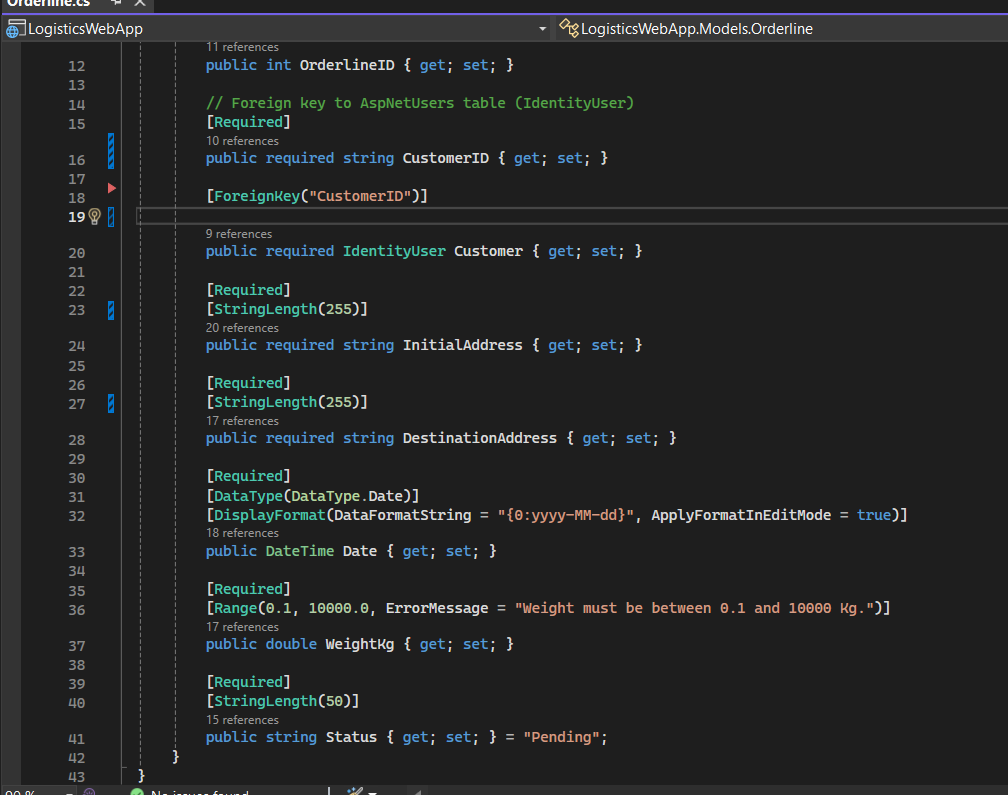


Figure 20: Orderline Model Class

### **TransportUnit.cs**

TransportUnitID: A unique identifier for each transport unit.

LorryID & Lorry: Foreign key and navigation property to link to a Lorry object (the vehicle). A transport unit *must* have a lorry.

DriverID & Driver: Foreign key and navigation property to link to a Driver object (the person driving). A transport unit *must* have a driver.

AssistantID & Assistant: Foreign key and navigation property to link to an Assistant object (the helper). A transport unit *must* have an assistant.

Container: A string property to describe or identify the container used for goods, with a maximum length of 100 characters.

Status: A string property indicating the current state of the transport unit (e.g., "Available", "In Transit", "Maintenance"), with a maximum length of 50 characters.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 21: TransportUnit.cs

# Controllers

There are eleven controllers in this project. Most of them have same basic structure. The following are some unique controllers.

## **Driver Controller**

This DriversController.cs file is a C# controller in an ASP.NET Core web app. It manages driver records in a logistics system. Here's a simple explanation of what it does:

**Access Control**

* Only users with **Manager and Admin role** can use this controller ([Authorize(Roles = "Manager,Admin")]).

**Functions**

1. **Index**
   * Shows a list of drivers.
   * You can search by name or phone.
2. **Details**
   * Shows detailed info of a driver by ID.
3. **Create (GET + POST)**
   * GET: Shows the form to add a new driver.
   * POST: Saves new driver to the database.
4. **Edit (GET + POST)**
   * GET: Shows the form to edit a driver.
   * POST: Updates driver info in the database.
5. **Delete (GET + POST)**
   * GET: Shows a confirmation page to delete a driver.
   * POST: Deletes the driver from the database.

**Helper**

* DriverExists: Checks if a driver exists by ID.

**Search Algorithm**

Linear search algorithm is used for searching. The Index action method in the DriversController which shows a list of drivers and includes a search feature. First, it saves the search keyword (searchString) into ViewData so the search box can remember the user's input. Then, it gets all drivers from the database. If the user typed something in the search box, it converts that keyword to lowercase for case-insensitive searching. It filters the drivers whose name or phone number (also converted to lowercase) contains the keyword. Finally, it sends the filtered (or full) driver list to the view.

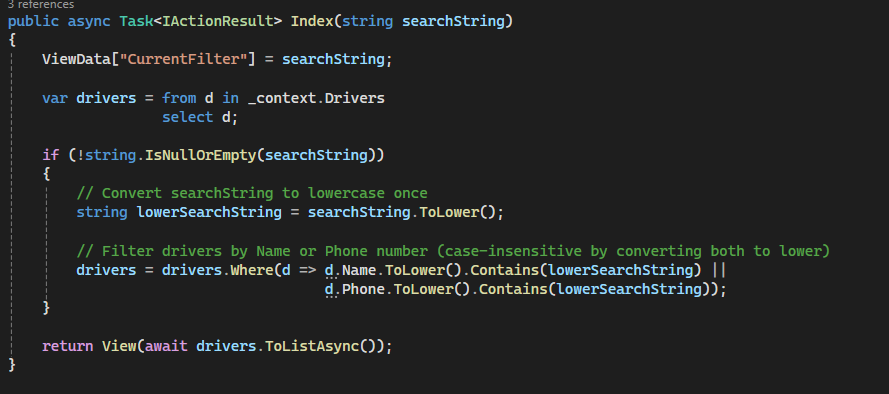


Figure 22: Driver search method

## **UserController**

This UsersController manages user accounts in an ASP.NET web app. Only users with the "Admin" or "Manager" role can access it. It lets you view a list of users, search them by name, email, or phone, and see their roles. You can also view details of one user, update their phone number, and change their roles (add or remove). It also supports deleting a user. The controller uses UserManager and RoleManager to manage user accounts and their roles.

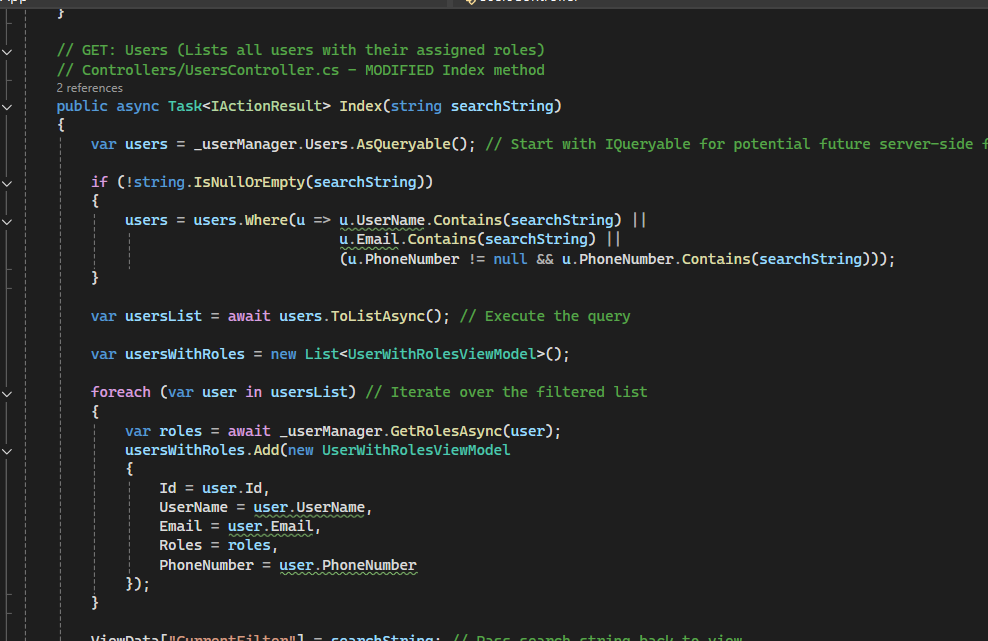


Figure 23: User Search Method

## **FeedBacksController**

The FeedBacksController in this ASP.NET Core application handles all the basic operations for managing feedback entries. It allows users to view a list of all feedbacks, see detailed information for a specific feedback, create new feedback entries, modify existing ones, and delete feedbacks. The controller uses Entity Framework Core to interact with the ApplicationDbContext database, ensuring that feedback data is persistent and can be accessed and managed through the web interface.

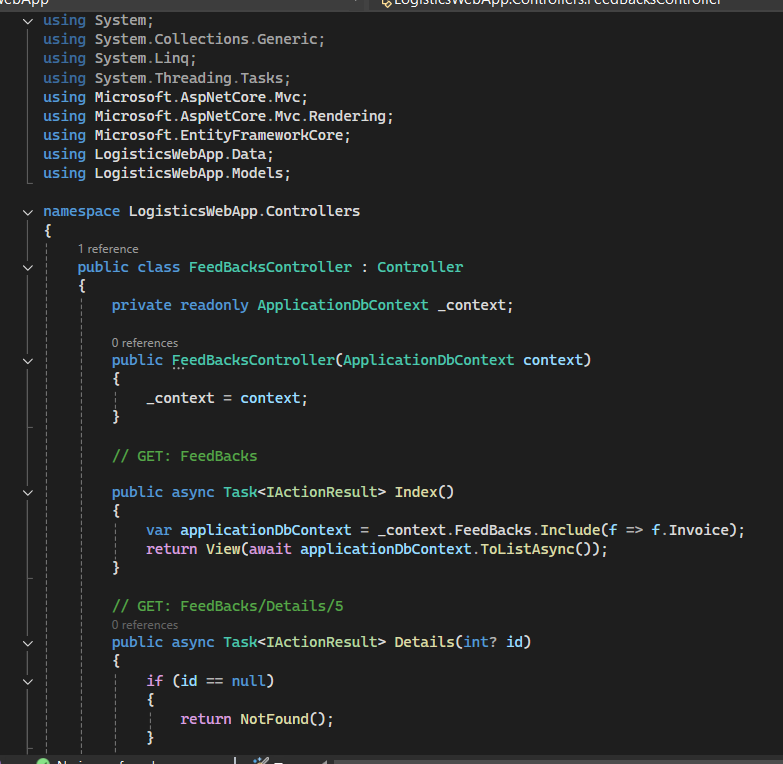


Figure 24: FeedBacksController

# Customer View

For new customer, he can create a new account. After creating a new account, an admin has to set confirmation email value false to true. After logging in as a customer,it will show the customer dashboard. Here the customer email and password is [zaw@gmail.com](mailto:zaw@gmail.com) and Zaw123!.

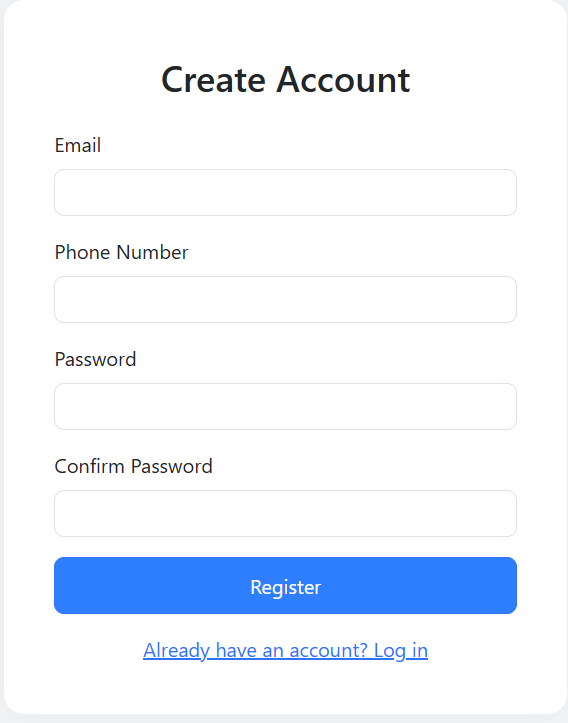


Figure: Create a new account

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Email Confirmation

A screenshot of a computer

AI-generated content may be incorrect.  
 Figure 25: Login as a customer

The first thing that website show is create order page. If the customer wants to create a order, he must fill the initial address, destination address, expected delivery data and item total weight. If the customer do not fill the required data, it will show error massages and cannot process to the creation order.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 26: Create Order Form

A screenshot of a computer

AI-generated content may be incorrect.

Figure 27: Form Validations

After a customer successfully created order, it will add to the orderline table. The customer can check all his order records in my orders page. But he can only view, he cannot edit nor delete. The new order will show in manager and admin view. They will create invoice for the order.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 28: Customer Order

In invoice list, customer can check order confirmation.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 29: Invoice List page

The customer can also give feedback based on his experience and can leave massage. A customer can also give stars form 1 to 5.The customer can see all the other customers’ feedbacks.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 30: Feedback Dashboard

# Admin View

In admin role, there are a lot of functions which are cannot be performed by customers. As an admin, he can also create an order and he can check all the orders in the system. An admin can also edit and delete current orders.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 31: Admin View

A screenshot of a computer

AI-generated content may be incorrect.

Figure 32: Orders List

A screenshot of a computer

AI-generated content may be incorrect.

Figure 33: Order Edit Page

A screenshot of a delete

AI-generated content may be incorrect.

Figure 34: Order Delete Page

An admin can control lorries information. Create, delete and edit. In create page, an admin need to put information such as license plate, model, capacity and status.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 35: Lorries List Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 36: Register New Lorry Form

A screenshot of a computer

AI-generated content may be incorrect.

Figure 37: Edit Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 38: Delete Page

**Manage Users Dashboard**

An admin can manage user accounts. Currently there are three roles admin, manager and customer. An admin can change role of users and also creates new user accounts. Users can be found by email, username or phone number.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 39: Manage Users Dashboard

A screenshot of a login form

AI-generated content may be incorrect.

Figure 40: Create New Account

A screenshot of a computer

AI-generated content may be incorrect.

Figure 41: Eidt User Page

**Driver Management Dashboard**

An admin can manage drivers information like other pages.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 42: Driver Management Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 43: Create New Driver Form

A screenshot of a computer

AI-generated content may be incorrect.

Figure 44: Driver Delete Form

**Mange Assistant Dashboard**

Like other dashers, an admin can create, edit and delete assistant information.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 45: Manage Assistant Dashboard

A screenshot of a computer

AI-generated content may be incorrect.

Figure 46: Create New Assistant Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 47: Delete Assistant Page

**Mange Transport Unit Information**

In manage transport unit page, an admin can choose driver, vehicle, assistant and container to create a propriate a lorry for an invoice.

A screenshot of a website

AI-generated content may be incorrect.

Figure 48: Mange Lorry page

A screenshot of a login form

AI-generated content may be incorrect.

Figure 49: Create Transport Unit Form

**Invoices Management Dashboard**

After a order come, the admin has to create an invoice for that order such as which driver will deliver, which container will be used , which assistant will go and calculate price.

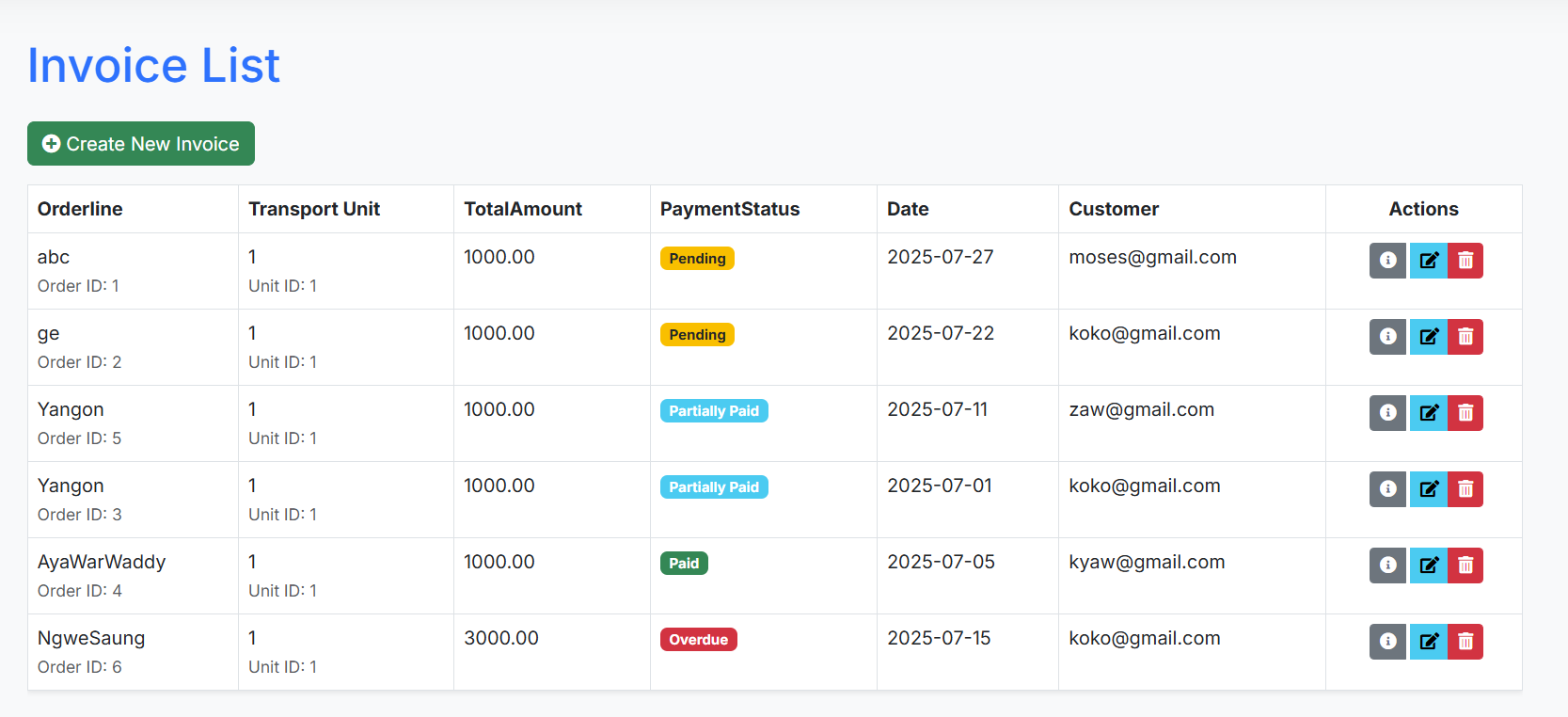
****

Figure 50: Invoices Management Dashboard

A screenshot of a invoice

AI-generated content may be incorrect.

Figure 51: Create Invoice Form

**Feedback Dashboard**

All roles can access to feedback dashboard but only admin and manager can edit this page.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 52:Invoice Dashboard

**Report Dashboard**

In report dashboard, the admin can check reports about income, how much each customer spent etc. He can also download these reports as excel files.

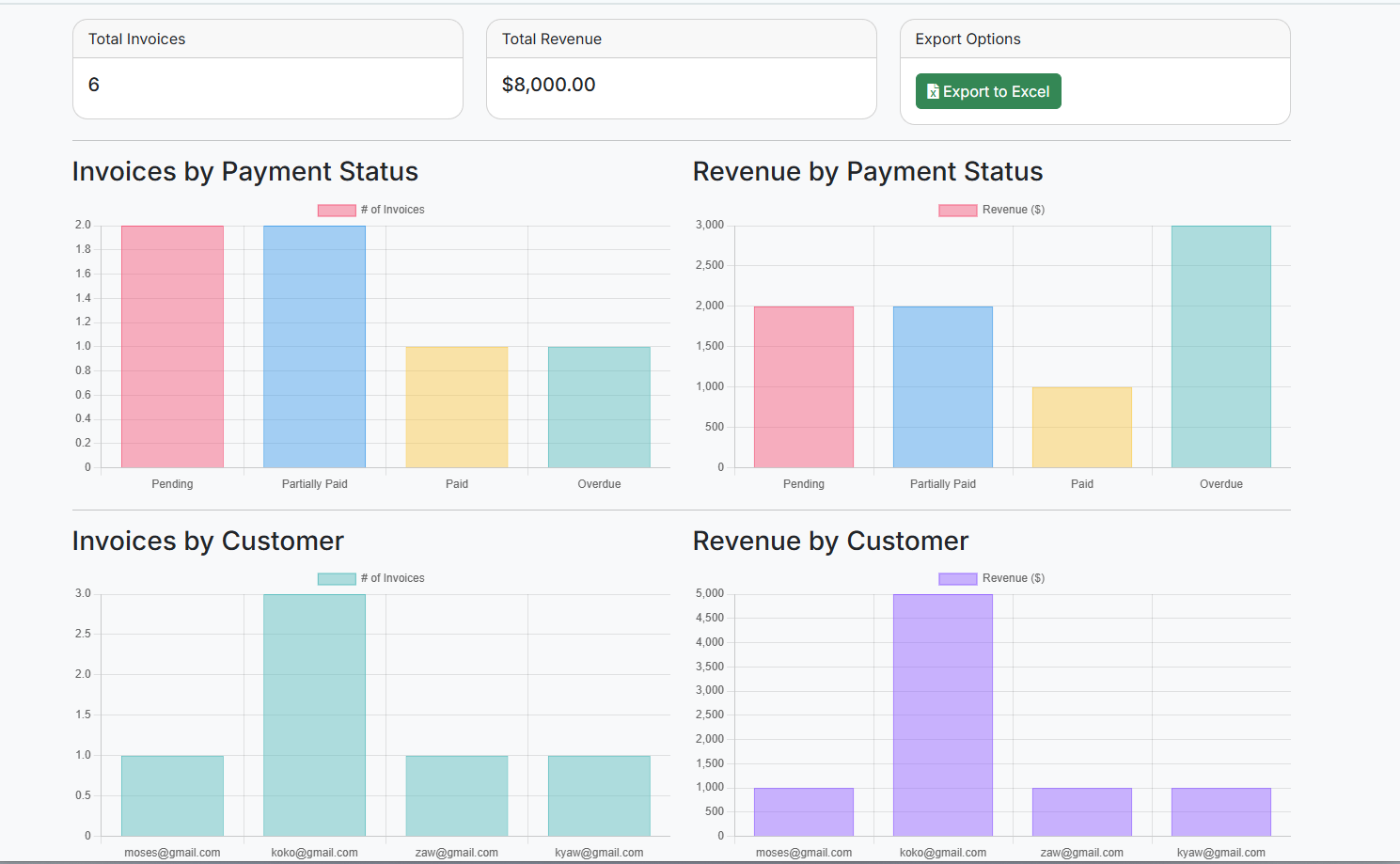


Figure 53: Admin Dashboard

# Reflection

Although my web application works, there are a lot of weakness and need improvements. A big problem is code repetition in controllers like AssistantsController.cs and DriverController.cs. hey do the same "add, edit, delete" stuff over and over which leads to code messy. By using a shared helper for common tasks can solve that problem. Also, error messages are too simple, like just saying "Not Found." We need to make them more helpful, telling the user *why* something went wrong. Another thing is big tasks inside controllers, like making Excel reports in InvoicesController.cs. I think it's better to move these into their own separate "service" classes. This keeps controllers clean and makes complex parts easier to manage.

Finally, linking Invoice and Orderline directly to IdentityUser for customers is a bit rigid. It would be more flexible to have a separate Customer model that can connect to IdentityUser if needed, but isn't forced to. This makes the system more adaptable for different types of customers later. Overall, the main weakness of my application is lack of DRY(Don’t Repeat Yourself).The codes need to be clear with errors and separate concerns so each part of the code has one main job.

# References

* <https://www.lazada.co.th>
* <https://www.lalamove.com/th-th>
* <https://www.flashexpress.com>
* <https://learn.microsoft.com/en-us/aspnet/overview>
* <https://www.geeksforgeeks.org/software-engineering/mvc-framework-introduction>
* https://www.tutorialspoint.com/data\_structures\_algorithms/linear\_search\_algorithm.htm