AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH (AIUB)

Dept. of Computer Science
Faculty of Science and Technology

CSC2210: OBJECT ORIENTED PROGRAMMING 2

Fall 2024-2025

Section: B

Group No: 12

Project Report On

*Project Name [**Filling Station Management**]*

Supervised By

Md. Hasibul Hasan

Submitted By:

Name	ID
1. Faysal Ahammed Chowdhury	**22-49046-3**
2. Md Shamsul Alam Pranto	**22-50000-3**
3. Asef Abdullah	**22-49923-3**

Obtained Marks for CO2 and CO3 (Description given in the following page)

Assessment Criteria	Not Attended/ Incorrect (0)	Inadequate (1-2)	Average (3)	Good (4)	Excellent (5)

Evaluation Criteria (CO2)	Total =	Evaluation Criteria (CO3)	Total =
Requirement fulfillment		Organization of the application	
Validation		Representation and Integration of Database	
Verification		Graphical User Interface	

CO2: Display and verify the mean of a real-life Project using the concepts of C# Graphical User Interface based environment with database integration to depict a desktop-based application.					
Assessment Criteria	**Not Attended/ Incorrect (0)**	**Inadequate (1-2)**	**Average (3)**	**Good (4)**	**Excellent (5)**
Evaluation Criteria	**Evaluation Definition**				
Requirement fulfillment	Fails to demonstrate any understanding of real-life scenario-based project development or functional requirement identification. There is no attempt to depict a project or identify functional requirements accurately.	Demonstrates limited understanding of real-life scenario-based project development and functional requirement identification. The project depicted lacks coherence or relevance to real-life scenarios, and functional requirements are inaccurately identified or insufficiently described.	Presents a basic depiction of a real-life scenario-based project and identifies some functional requirements. However, the project lacks depth or complexity, and some functional requirements may be vaguely defined or missing key details.	Effectively demonstrates a realistic scenario-based project and accurately identifies most functional requirements. The project is well-developed with appropriate complexity, and functional requirements are clearly articulated with relevant details.	Exhibits an exceptional understanding of real-life scenario-based project development and accurately identifies all functional requirements. The project is meticulously developed with thorough attention to detail, reflecting a comprehensive understanding of Object-Oriented Programming project development activities.
Validation	Fails to demonstrate any understanding or implementation of validation forms in their system. There is no attempt to deal with data validation, and validation requirements are completely ignored or incorrectly applied.	Demonstrates limited understanding of validation forms and data validation techniques. While some attempt may be made to implement validation, it is incomplete or poorly executed, leading to inadequate handling of data validation.	Shows a basic understanding of validation forms and data validation techniques. They attempt to implement validation, but some aspects may be missing or incorrectly implemented, resulting in partial or inconsistent handling of data validation.	Effectively demonstrates the use of validation forms and implements data validation techniques. Validation is mostly accurate and comprehensive, ensuring the proper handling of data input and verification in the system.	Exhibits an exceptional understanding and implementation of validation forms and data validation techniques. Validation is meticulously implemented with thorough attention to detail, ensuring robust data validation procedures and contributing to the overall reliability and integrity of the system.
Verification	Fails to demonstrate any attempt to verify the system data or functional requirements. There is no evidence of understanding or implementation	Demonstrates limited understanding of verification processes and data flow in the system. Verification attempts are incomplete or	Shows a basic understanding of verification processes and attempts to verify system data. However, verification efforts may be inconsistent or	Identifies and verifies system data, ensuring proper functional requirements are met. Verification efforts are mostly accurate and thorough, with attention to	Exhibits an exceptional understanding of verification processes and meticulously verifies system data. Verification efforts are comprehensive

	of verification processes, and data flow is not considered.	inaccurate, and there is insufficient consideration given to ensuring data integrity and functionality.	lack thoroughness, and there may be gaps in ensuring proper functional requirements and data flow.	ensuring data integrity and appropriate data flow within the system.	and precise, with a keen focus on ensuring all functional requirements are met and maintaining proper data flow throughout the system.

CO3: Prepare and Explain a real life desktop based application synthesizing several component of C# along with development tools to adhere the given requirements.

Assessment Criteria	Not Attended/ Incorrect (0)	Inadequate (1-2)	Average (3)	Good (4)	Excellent (5)
Evaluation Criteria	**Evaluation Definition**				
Organization of the application	Fails to identify any suitable real time application or requirements for project development activities related to OOP.	Limited understanding about the project scopes and scenarios or identification of functional requirements.	Lacks depth or relevance to OOP project development activities and may contain inaccuracies. Real-life scenarios are mentioned, but the discussion lacks depth or clarity.	Consider and integrate the idea of several core aspects of the project along with relevance to real-life scenarios. Demonstrating a solid understanding of the application presentation.	Generalize and exhibits an exceptional understanding of project preparation according to a to real-life scenarios. Also contains proper and insightful identification of the system which is comprehensive and precise.
Representation and Integration of Database	Fails to identify and present any understanding or implementation of database. Also failed to integrate the data with the project itself.	Limited understanding of the database concepts or their proper way of using in a real time project. While some attempt may be made to implement but it is incomplete or poorly executed, leading to inadequate design.	Lacks depth or relevance to database integration with the application. Shows a basic understanding but some aspects may be missing or incorrectly implemented, resulting in partial or inconsistency. May lack proper normalization.	Integrate the database with the forms properly and implements it with proper validation which is mostly accurate and comprehensive, ensuring the proper handling of data input and verification along with general normalization.	Exhibits an exceptional understanding and implementation of database ensuring attention to detail, and robust data manipulation procedures and contributing to the overall clarity.
Graphical User Interface	Fails to present or prepare GUI based application interfaces. There is no evidence of creating or integrating such things according to their usefulness.	Limited understanding of graphical user interfaces. Lack of design knowledge. Very poor attempt to make such things which are currently obsolete or can't be identified as coherent.	Shows a basic understanding of creating user interfaces. Most of them are interconnected but maybe some of them lack it. However, most of it can be described as user friendly.	Effectively identifies and meet the consider the simplicity. Design related works are mostly accurate and taken proper attention to ensuring a user-friendly coherent system.	Exhibits an exceptional work design following a high standard of simple and elegant work. Several controls and mechanism has been organized in a preferred way according to the coherent usage .

Table of Contents: **Page no.**

1. Introduction

The **Filling Station Management System** is a desktop-based application developed using C# and MSSQL to help manage daily operations at a filling station more efficiently. It provides a simple graphical user interface that allows users to handle sales, inventory, expenses, and employee/user records with ease. The system reduces manual work, ensures proper record-keeping, and helps in tracking fuel stock, transactions, and employee activities. There are two types of users: Admins, who have full access to all features, and Employees, who can only add sales and expenses. This system makes managing a filling station easier by keeping all important data organized and reducing errors.

2. User Story

The Filling Station Management System has two types of users: **Admin** and **Employee**.

Admin

Sales Management

- Can add, view, and delete sales records.
- Can search sales history by Sale ID or date range.
- Can view sales reports (daily, weekly, monthly, or custom date range).
- After a sale is completed, an invoice is generated, which can be accessed later by clicking on the sale details.
- When adding a sale, the inventory is displayed on the left side, and fuel can be searched by name.

Inventory Management

- Can view, search, add, edit, and delete inventory records.
- Can search inventory by fuel name for quick access.

User Management

- Can view, search, add, edit, and delete users.
- Can search users by name and filter by Admin, Employee, or All.
- Employees cannot register themselves; only an Admin can add them.

Expense Management

- Can add, view, and delete expense records.
- Can search expenses by category and filter by date range.
- Can view expense reports (daily, weekly, monthly, or custom date range).

System Behavior and Updates

- If an Admin updates their role, they are logged out immediately.
- If an Admin changes their name/information, the updated name/information is instantly reflected across the system.

Employee

- Can add sales and add expenses.
- After completing a sale, an invoice is generated for reference.
- On the dashboard, can view today's sales list, the total amount made by her/him, and a real-time clock.
- When adding a sale, the inventory is displayed on the left side, and fuel can be searched by name.

3. ***ER Diagram***

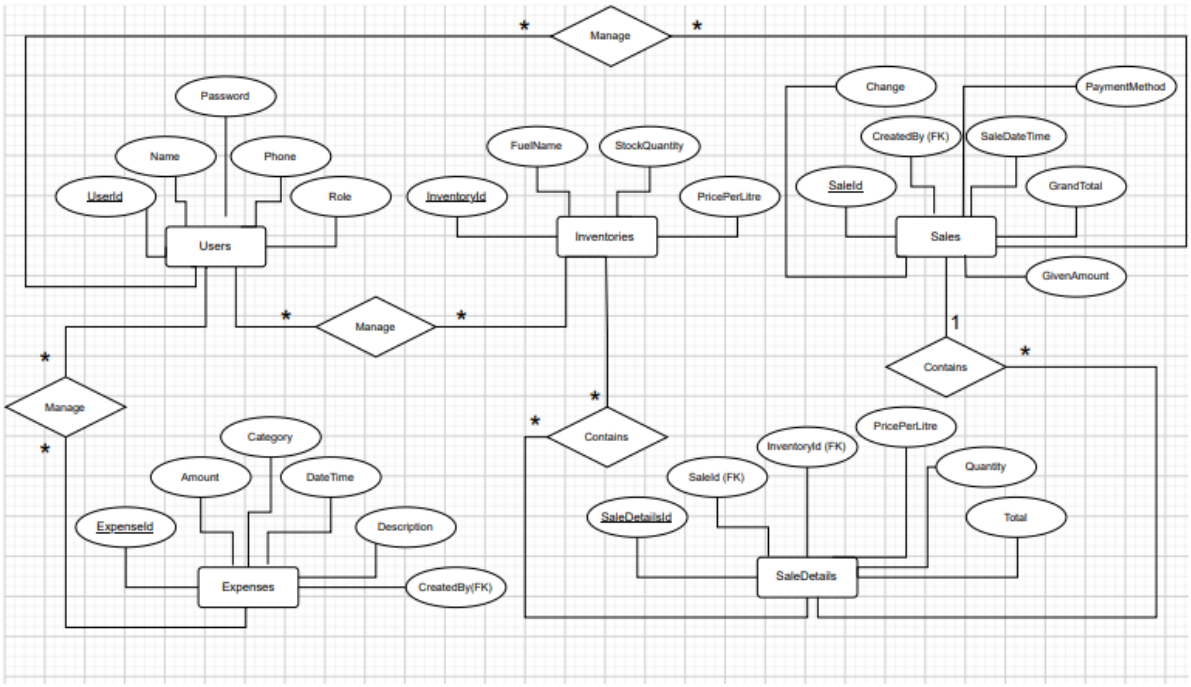

Fig-3.1 Entity Relationship Diagram

4. SQL Queries:

1. **Login**
   1.1. SELECT * FROM Users WHERE Phone = '{phone}' and Password = '{pass}'

2. **Admin Dashboard**
   2.1. SELECT * FROM Sales ORDER BY SaleDateTime DESC
   2.2. SELECT * FROM Expenses ORDER BY DateTime DESC
   2.3. SELECT Name FROM Users WHERE UserId = '{row["CreatedBy"]}'

3. **Employee Dashboard**
   3.1. SELECT * FROM Sales
   WHERE CreatedBy = '{this.currentUser["UserId"].ToString()}' AND SaleDateTime BETWEEN '{startDate}' AND '{endDate}' ORDER BY SaleDateTime DESC

4. **User**
   4.1. SELECT * FROM Users
   4.2. SELECT * FROM Users WHERE UserId = '{userId}'
   4.3. SELECT MAX(UserId) FROM Users
   4.4. SELECT * FROM Users WHERE Phone = '{phone}'
   4.5. INSERT INTO Users VALUES ('{id}', '{name}', '{phone}', '{password}', '{role}')
   4.6. UPDATE Users
   SET Name = '{name}',
   Phone = '{phone}',
   Password = '{password}',
   Role = '{role}'
   WHERE UserId = '{id}'
   4.7. string part = "";
   if (!this.txtSearch.Text.IsNullOrEmpty())
   part = $"WHERE Name LIKE '%{this.txtSearch.Text}%'";
   if (!this.txtSearch.Text.IsNullOrEmpty() && this.cboUserType.SelectedIndex != 0)
   part += " AND ";
   if (this.cboUserType.SelectedIndex != 0 && this.txtSearch.Text.IsNullOrEmpty())
   part = "WHERE ";
   if (this.cboUserType.SelectedIndex != 0)
   part += $"Role = '{this.cboUserType.SelectedItem.ToString()}'";
   string query = $"SELECT * FROM Users {part}";
   4.8. DELETE FROM Users WHERE UserId = '{userId}'

5. **Inventory**
   5.1. SELECT * FROM Inventories
   5.2. SELECT MAX(InventoryId) FROM Inventories
   5.3. INSERT INTO Inventories VALUES ('{id}', '{fuelName}', '{stockQuantity}', '{pricePerLitre}')
   5.4. SELECT * FROM Inventories WHERE InventoryId = '{inventoryId}'
   5.5. UPDATE Inventories
   SET FuelName = '{fuelName}',

StockQuantity = '{stockQuantity}',
PricePerLitre = '{pricePerLitre}'
WHERE InventoryId = '{id}'

5.6. DELETE FROM Users WHERE UserId = '{userId}'

5.7. string part = "";
if (!this.txtSearch.Text.IsNullOrEmpty())
part = $"WHERE Name LIKE '%{this.txtSearch.Text}%'";
if (!this.txtSearch.Text.IsNullOrEmpty() && this.cboUserType.SelectedIndex != 0)
part += " AND ";
if (this.cboUserType.SelectedIndex != 0 && this.txtSearch.Text.IsNullOrEmpty())
part = "WHERE ";
if (this.cboUserType.SelectedIndex != 0)
part += $"Role = '{this.cboUserType.SelectedItem.ToString()}'";
string query = $"SELECT * FROM Users {part}";

6. **Sales**

6.1. SELECT * FROM Sales ORDER BY SaleDateTime DESC

6.2. SELECT Name FROM Users WHERE UserId = '{row["CreatedBy"]}'

6.3. string part = "";
if (!this.txtSearch.Text.IsNullOrEmpty())
part = $"SaleId LIKE '%{this.txtSearch.Text}%' AND ";
string query = $"SELECT * FROM Sales WHERE {part} SaleDateTime BETWEEN '{startDate}' AND '{endDate}' ORDER BY SaleDateTime DESC;";

6.4. SELECT * FROM Sales WHERE SaleDateTime BETWEEN '{startDate}' AND '{endDate}';

6.5. SELECT * FROM SaleDetails WHERE SaleId = '{saleId}'

6.6. UPDATE Inventories
SET StockQuantity = '{currentStock[dr["InventoryId"].ToString()]}'
WHERE InventoryId = '{dr["InventoryId"].ToString()}'

6.7. DELETE FROM Sales WHERE SaleId = '{saleId}'

6.8. DELETE FROM SaleDetails WHERE SaleId = '{saleId}'

6.9. SELECT * FROM SaleDetails WHERE SaleId = '{saleId}'

6.10. SELECT FuelName FROM Inventories WHERE InventoryId = '{row["InventoryId"]}'

6.11. SELECT * FROM Sales WHERE SaleId = '{saleId}'

6.12. SELECT Name FROM Users WHERE UserId = '{dt.Rows[0]["CreatedBy"]}'

6.13. SELECT MAX(SaleId) FROM Sales

6.14. SELECT MAX(SaleDetailId) FROM SaleDetails

6.15. INSERT INTO SaleDetails VALUES ('{saleDetailId}', '{this.lblSaleId.Text}', '{inventoryId}', '{pricePerLitre}', '{quantity}', '{total}')

6.16. SELECT StockQuantity FROM Inventories WHERE InventoryId = '{inventoryId}'

6.17. UPDATE Inventories
SET StockQuantity = '{newStockQuantity}'
WHERE InventoryId = '{inventoryId}'

6.18. INSERT INTO Sales VALUES ('{id}', '{saleDateTime}', '{currentUser["UserId"].ToString()}', '{grandTotal.ToString()}', '{givenAmount.ToString()}', '{changeAmount.ToString()}', '{method}')

7. **Expense**

7.1. SELECT * FROM Expenses ORDER BY DateTime DESC

7.2. SELECT Name FROM Users WHERE UserId = '{row["CreatedBy"]}'

7.3. string part = "";
if (!this.txtSearch.Text.IsNullOrEmpty())
part = $"Category LIKE '%{this.txtSearch.Text}%' AND ";
string query = $"SELECT * FROM Expenses WHERE {part} DateTime BETWEEN '{startDate}' AND '{endDate}' ORDER BY DateTime DESC;";

7.4. SELECT * FROM Expenses WHERE DateTime BETWEEN '{startDate}' AND '{endDate}';

7.5. DELETE FROM Expenses WHERE ExpenseId = '{expenseId}'

7.6. SELECT * FROM Expenses WHERE ExpenseId = '{expenseId}'

7.7. UPDATE Expenses

7.8. SET ExpenseId = '{id}',
Amount = '{amount}',
Category = '{category}',
DateTime = '{dateTime}',
Description = '{description}'
WHERE ExpenseId = '{id}'"

7.9. SELECT MAX(ExpenseId) FROM Expenses

7.10. INSERT INTO Expenses VALUES ('{id}', '{amount}', '{category}', '{dateTime}', '{description}', '{currentUser["UserId"].ToString()}')

5. ***Screenshots***

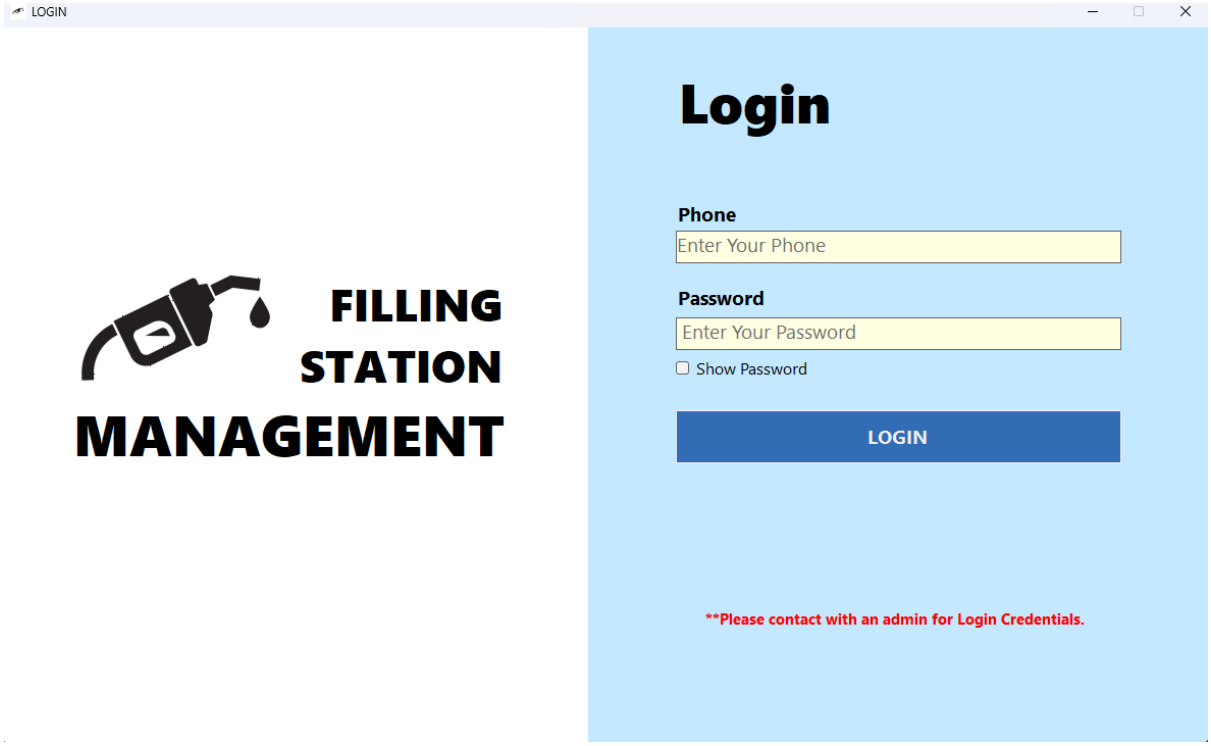

Fig-5.1 Login

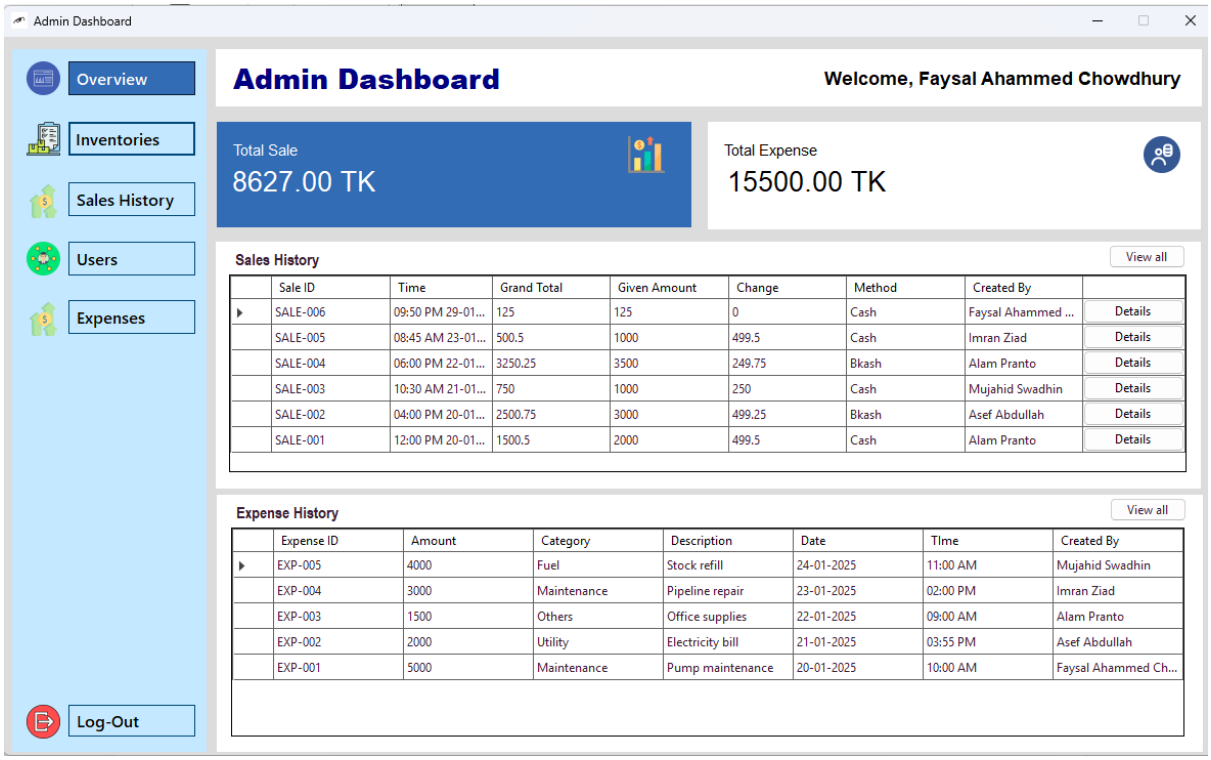

Fig-5.2 Admin Dashboard

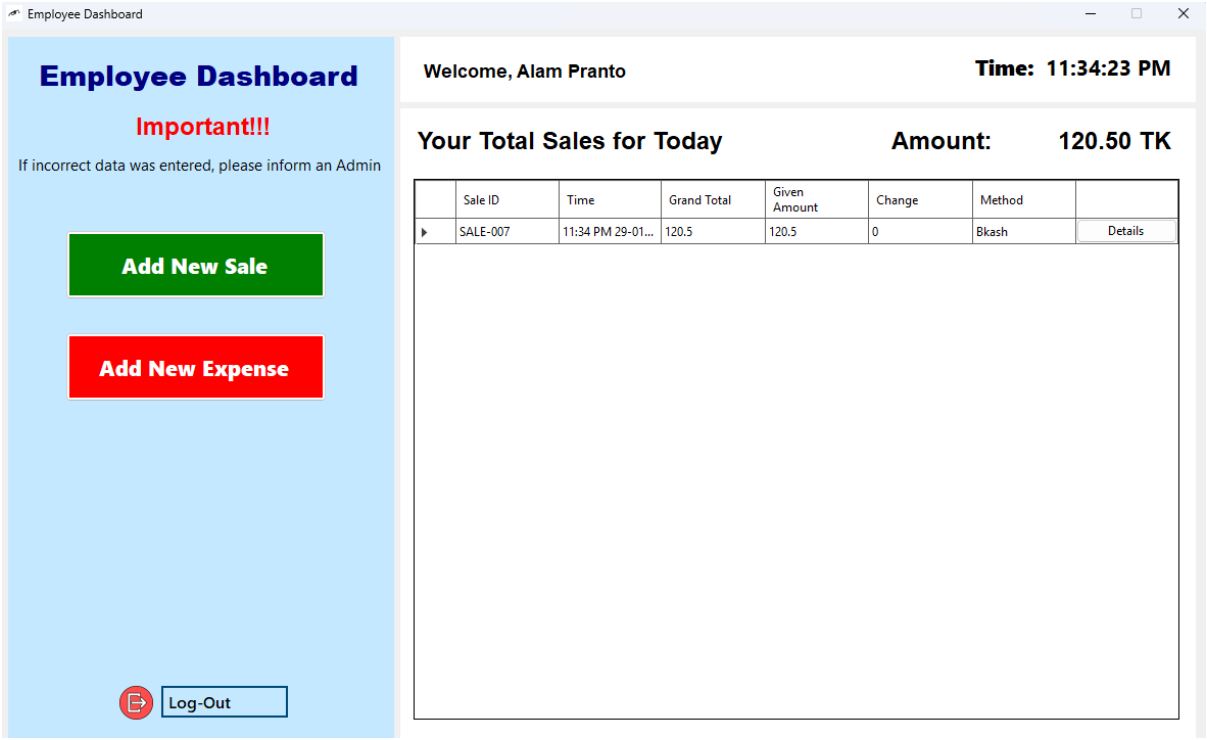

Fig-5.3 Employee Dashboard

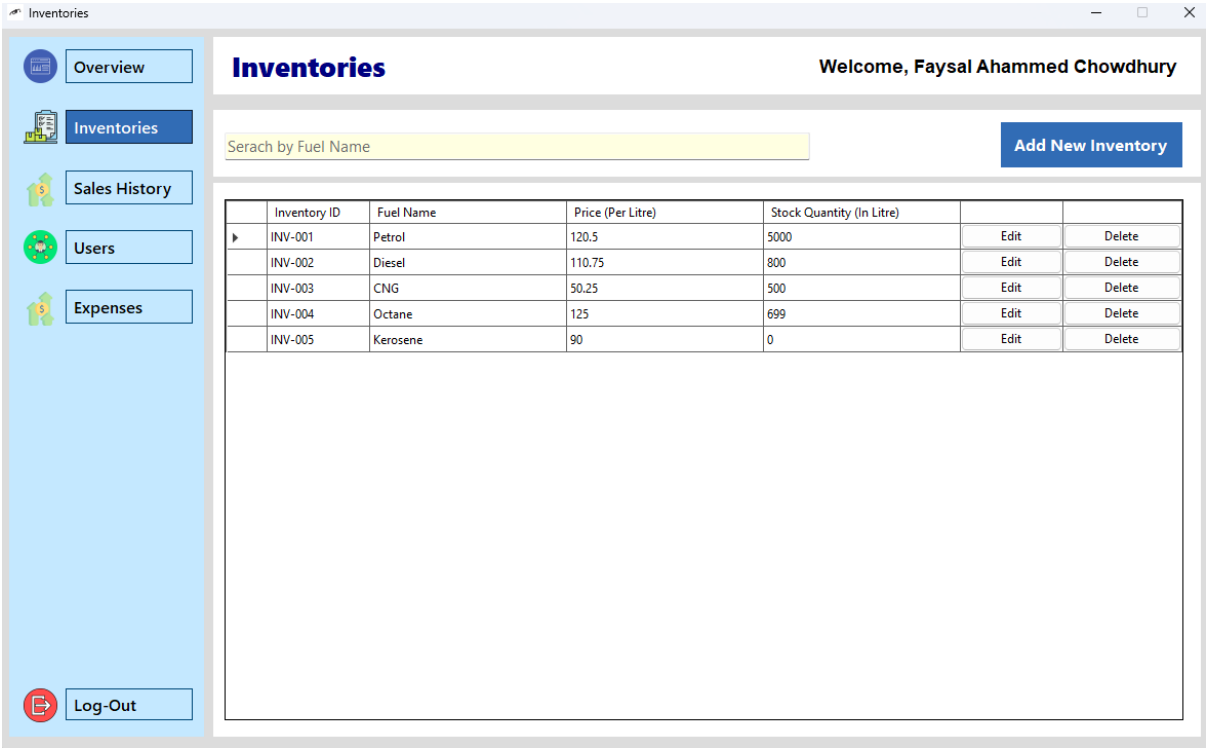

Fig-5.4 Inventories

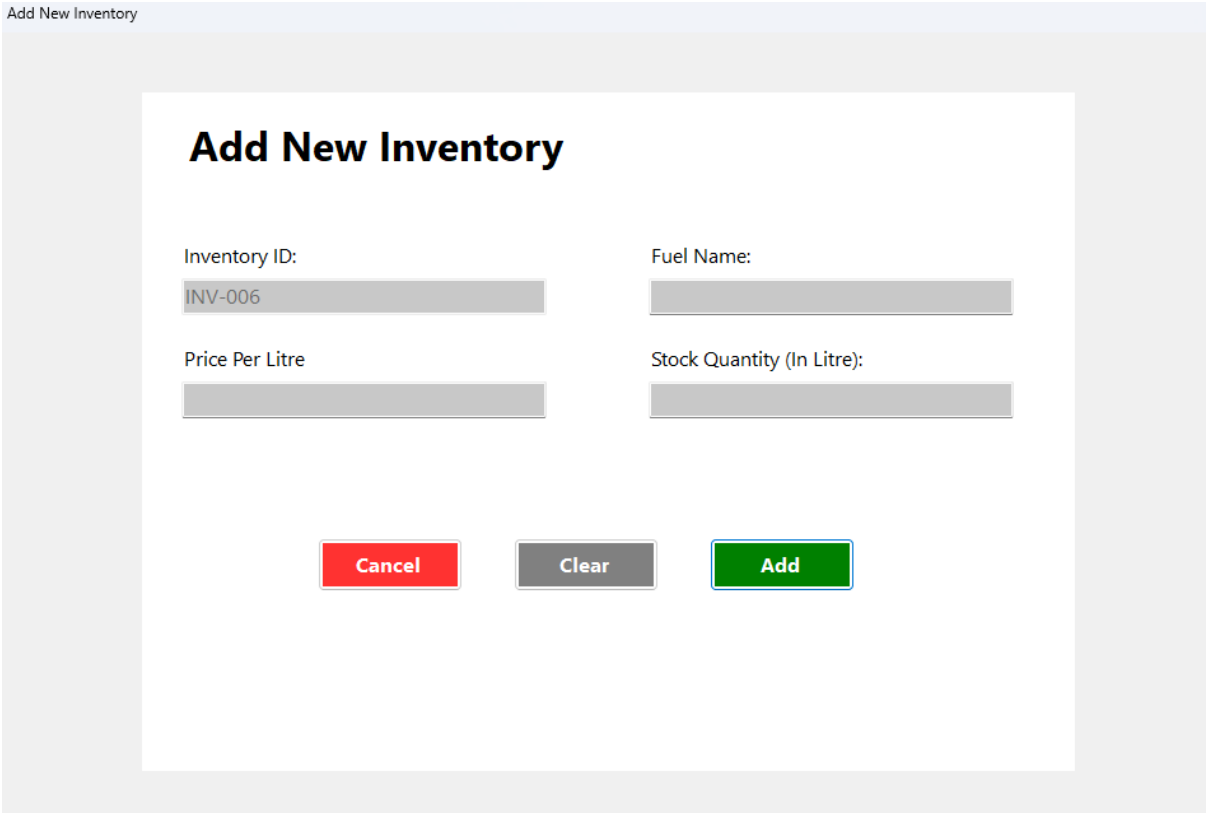

Fig-5.5 Add New Inventory

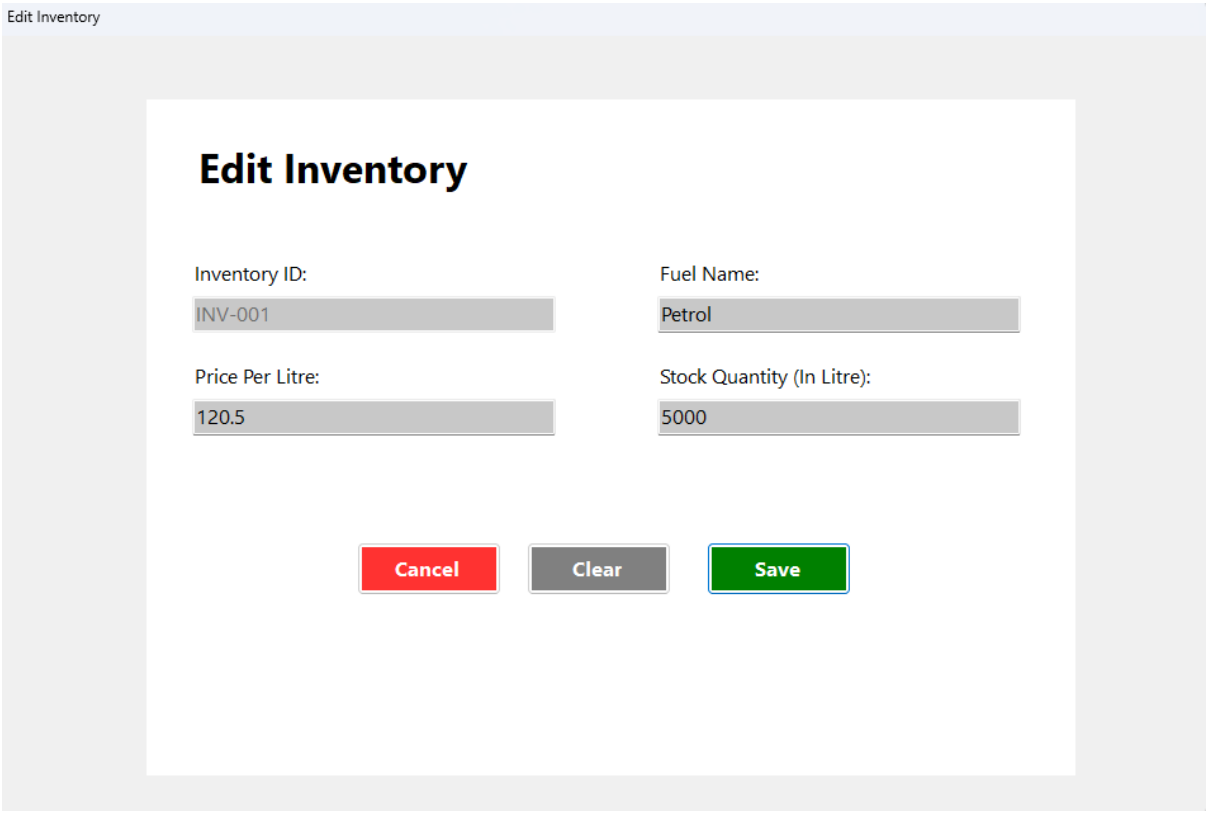

Fig-5.6 Edit Inventory

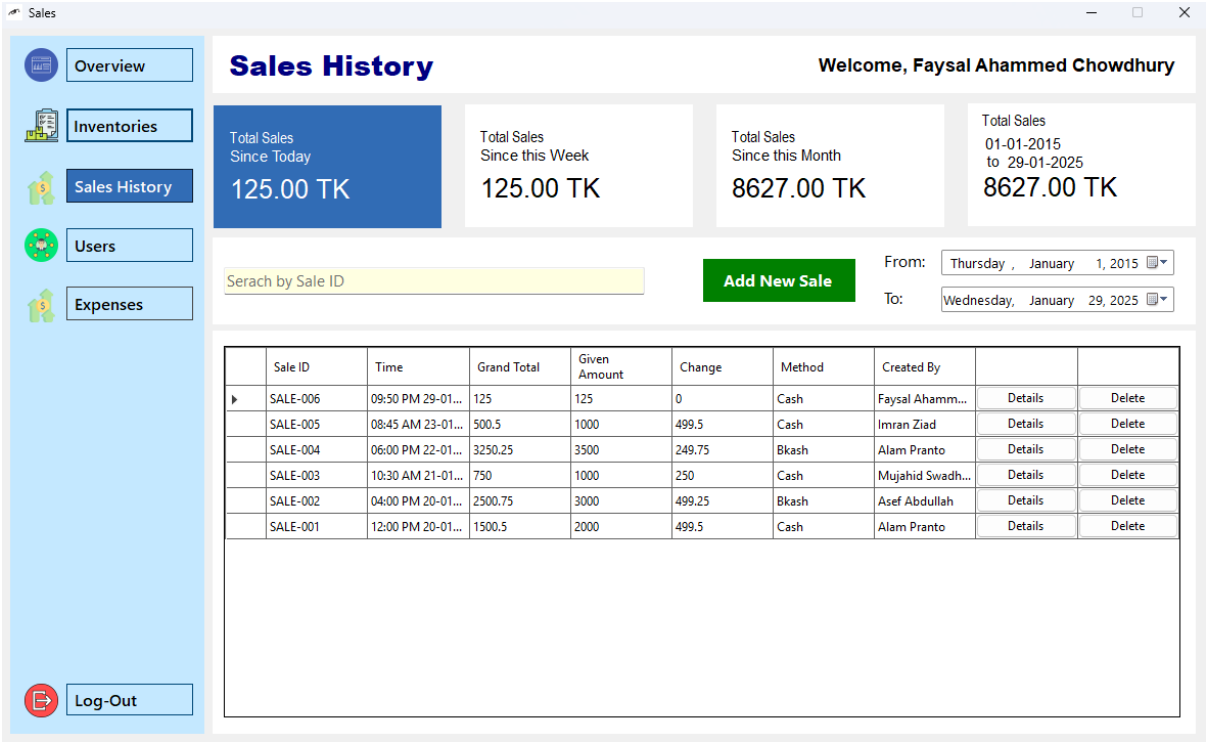

Fig-5.7 Sales History

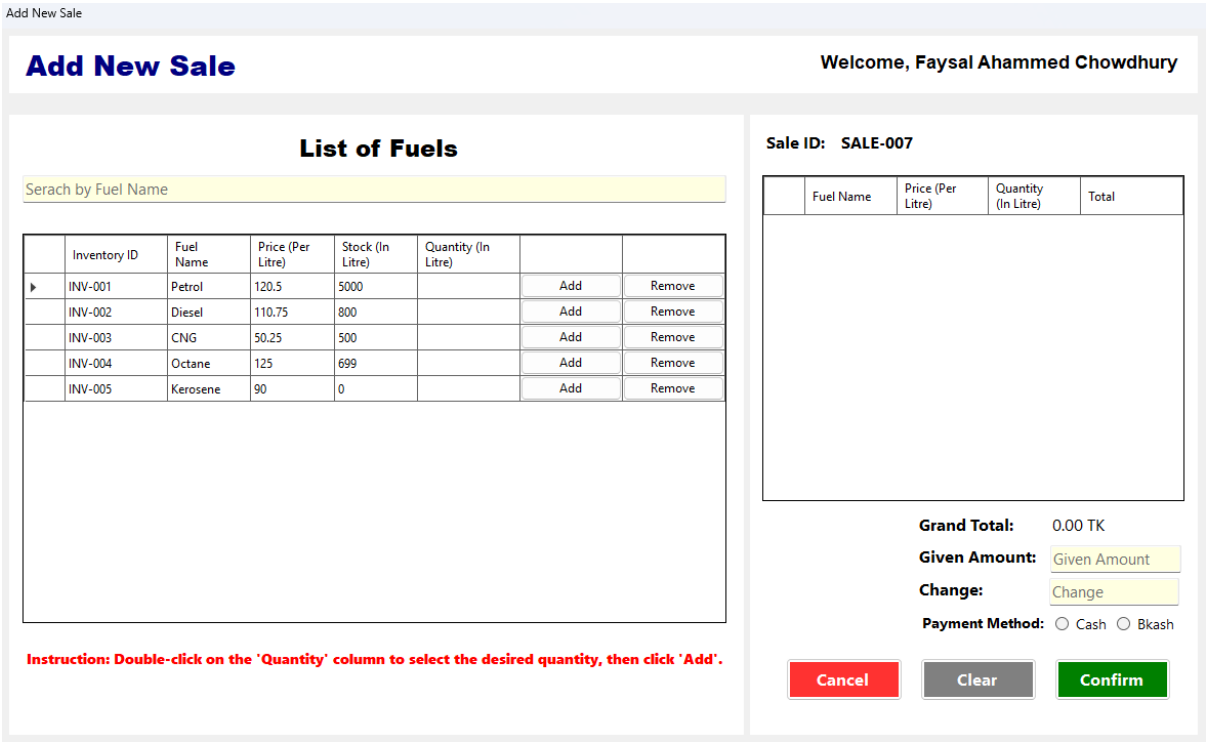

Fig-5.8 Add New Sale

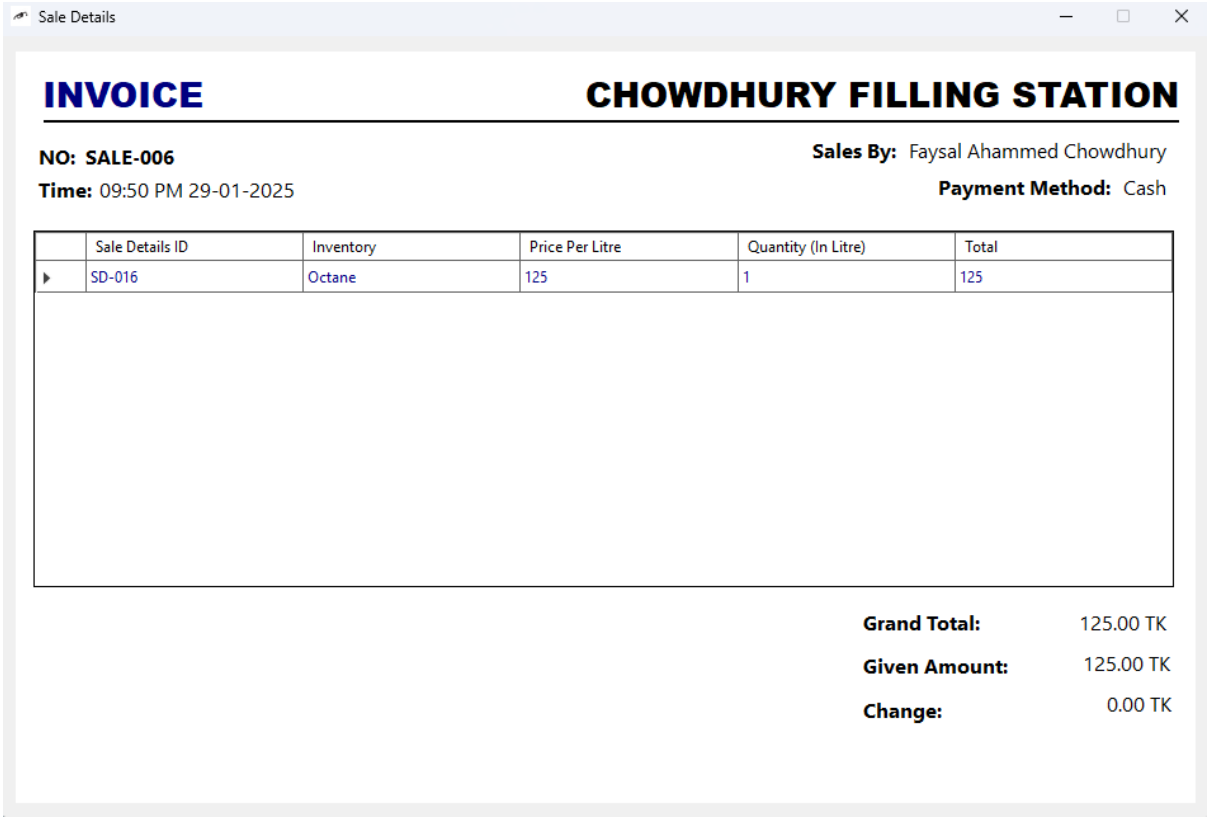

Fig-5.9 Sale Details (Invoice)

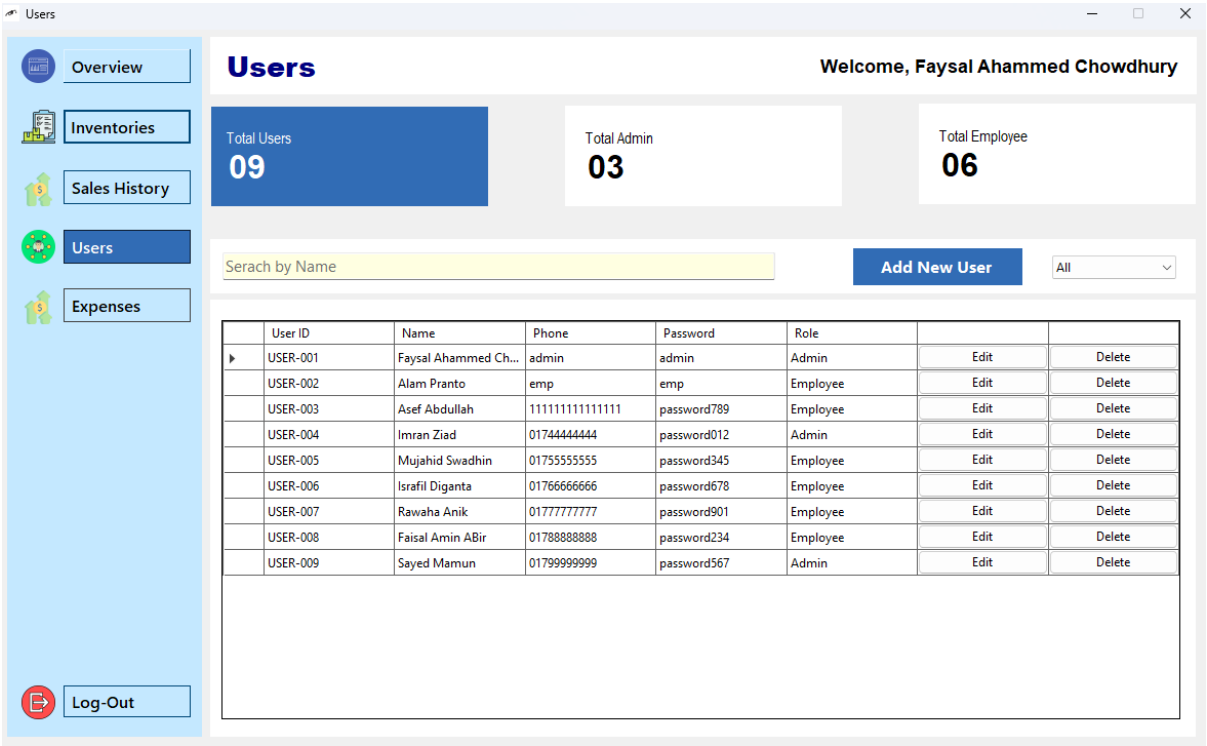

Fig-5.10 Users

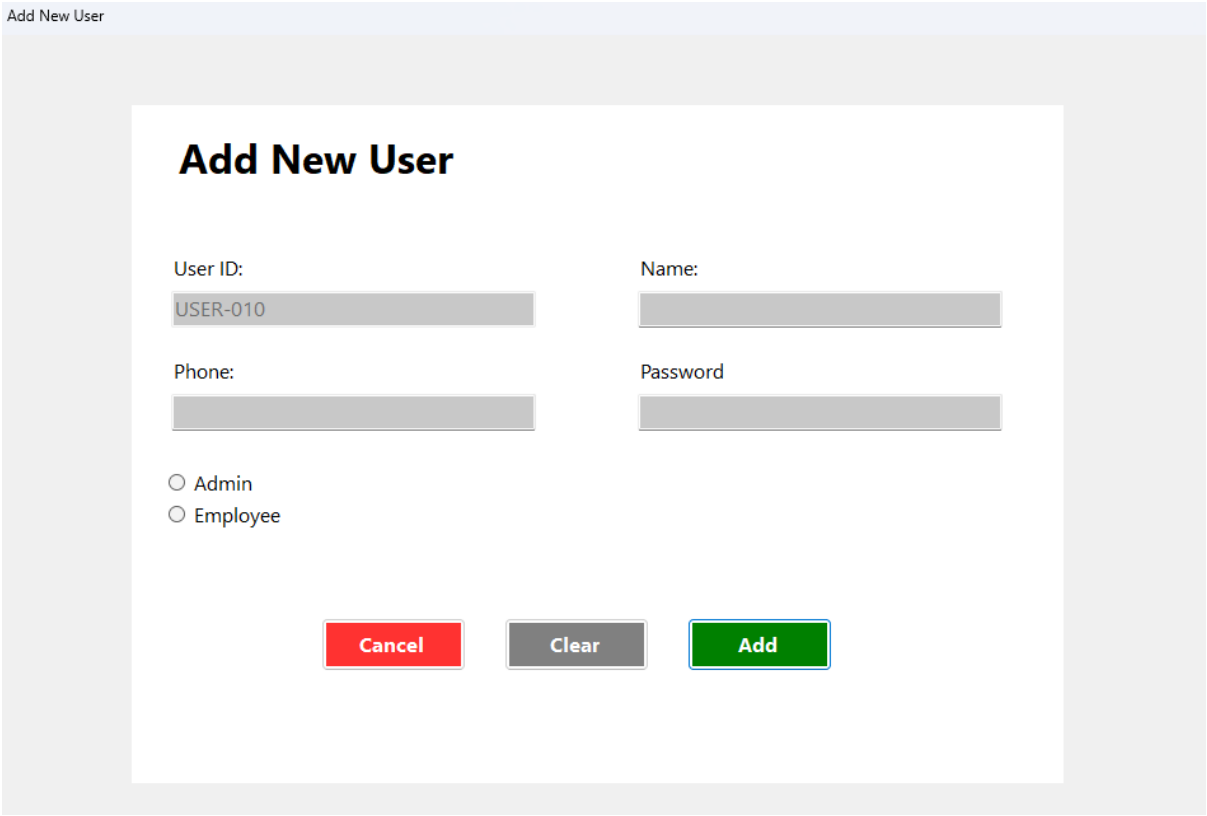

Fig-5.11 Add New User

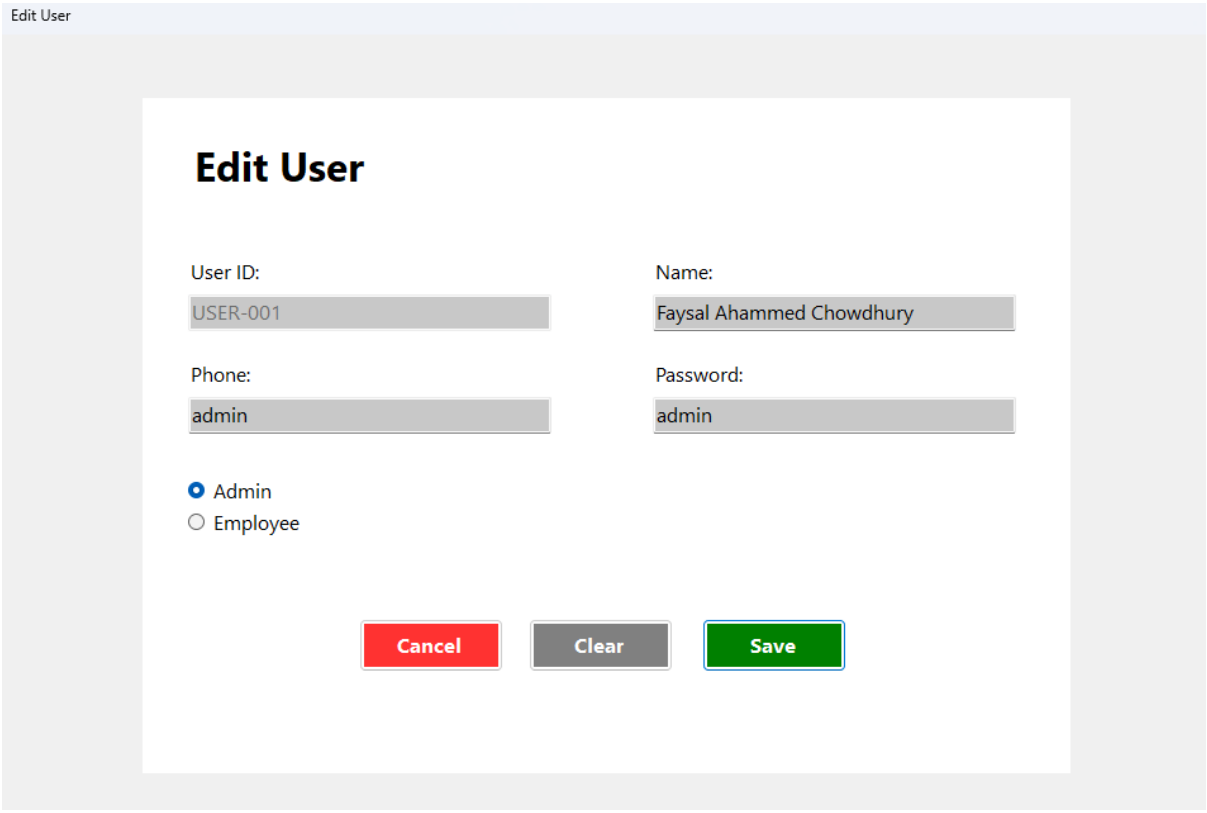

Fig-5.12 Edit User

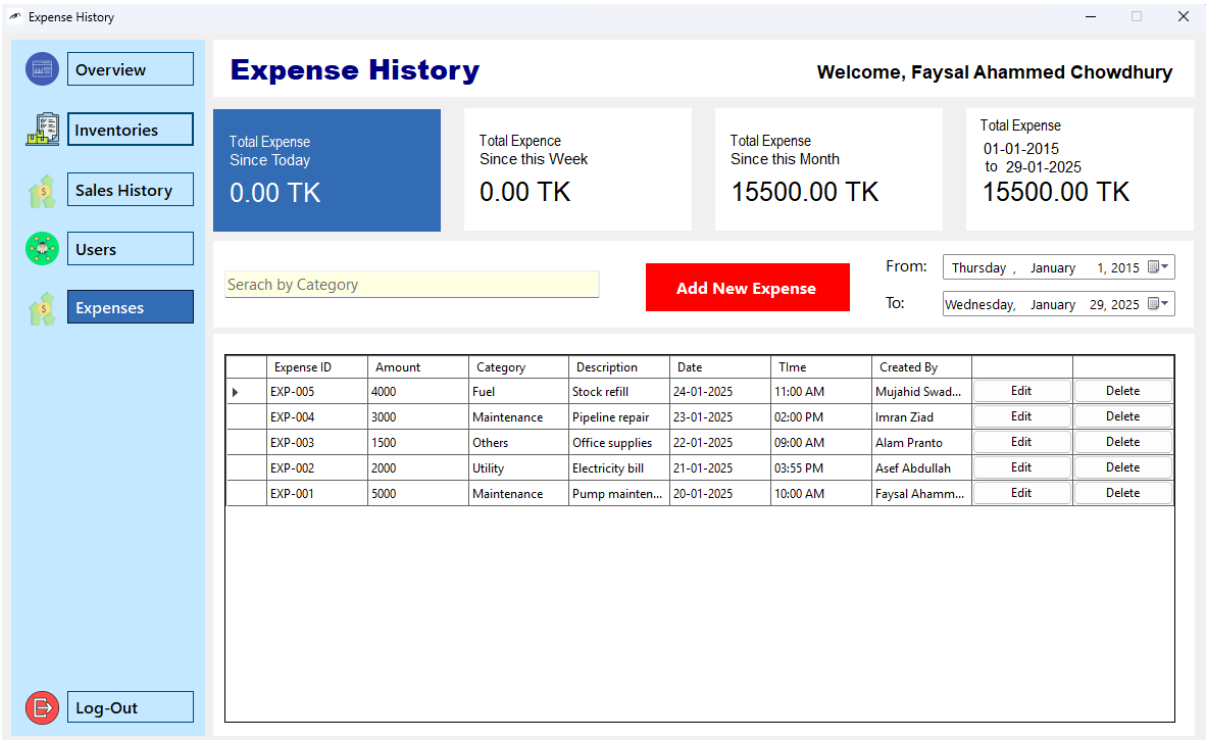

Fig-5.13 Expense

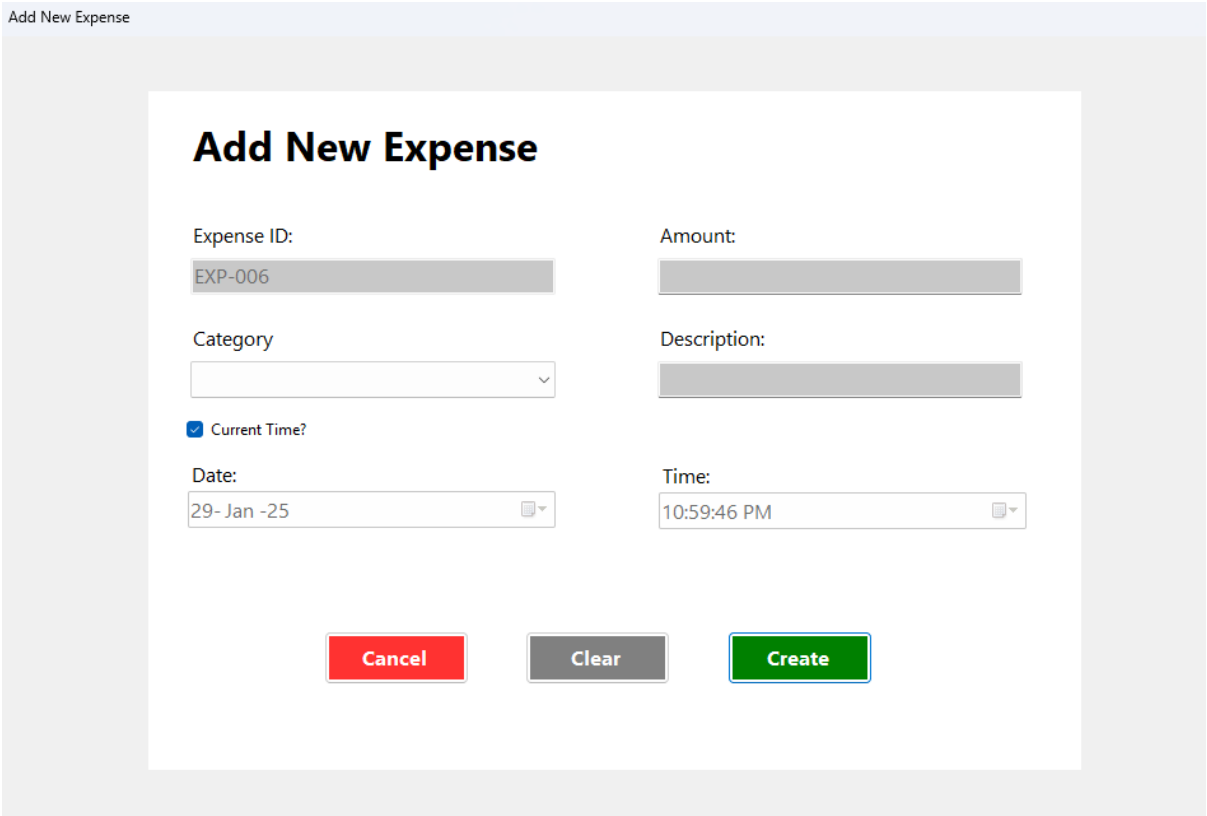

Fig-5.14 Add New Expense

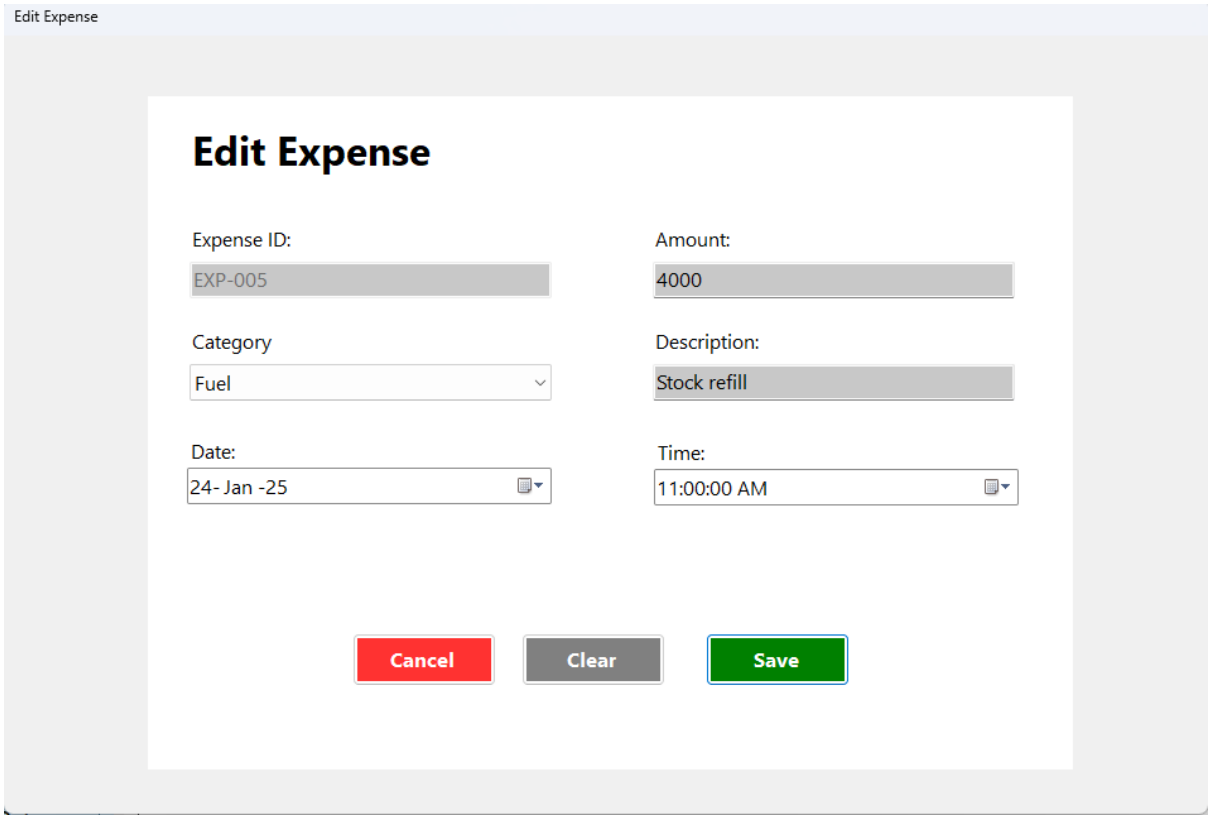

Fig-5.15 Edit Expense

6. ***References***

- **Discussion & Suggestions:**

  Discussed the project and received suggestions.
  - Shamim Ahmed – AIUB Alumni and currently Software Engineer. LinkedIn.
  - Israfil Diganta – AIUB Alumni, currently AIUB ACM ICPC Trainer and Software Engineer. LinkedIn.

- **C# Documentation:**

  Microsoft. *C#* Documentation. https://learn.microsoft.com/en-us/dotnet/csharp/.

- **ER Diagram Lecture Slide:**

  AIUB lecture slide on ER Diagrams. Link.

- **SQL Tutorial:**

  W3Schools SQL tutorial for SQL queries. Link.

- **Adding a Button to DataGridView Column:**

  Stack Overflow discussion on adding a button to a column in DataGridView. Link.

- **Working with DateTime in Sales and Expenses:**

  Microsoft documentation on DateTime for handling date and time in. Link.