# Mobile Shop Management System



Session: 2022 – 2026

**Submitted by:**

Afeera Fatima 2022-CS-151

**Submitted To:**

Prof. Dr. Muhammad Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

Contents

[Mobile Shop Management System 1](#_Toc141628393)

[1. Introduction: 5](#_Toc141628394)

[1.1 Project Overview 5](#_Toc141628395)

[1.2 Objectives 5](#_Toc141628396)

[1.2.1. Efficient Inventory Management: 5](#_Toc141628397)

[1.2.2. Customer Relationship Management 5](#_Toc141628398)

[1.2.3. Sales Process: 5](#_Toc141628399)

[1.2.4. Employee Management 6](#_Toc141628400)

[1.3 Functionality 6](#_Toc141628401)

[1.3.1 Inventory Management: 6](#_Toc141628402)

[1.3.2 Sales: 6](#_Toc141628403)

[1.3.3 Customer Management: 6](#_Toc141628404)

[2. OOP Concepts 6](#_Toc141628405)

[2.1 Association: 6](#_Toc141628406)

[2.1.1 Composition 6](#_Toc141628407)

[2.1.2 Aggregation 7](#_Toc141628408)

[2.2 Inheritance: 7](#_Toc141628409)

[2.3 Polymorphism 8](#_Toc141628410)

[2.4 Comparison with Procedural Programming 8](#_Toc141628411)

[3. Design Pattern Implementation: 8](#_Toc141628412)

[3.1 BL (Business Layer) 8](#_Toc141628413)

[3.2 DL (Data Layer) 9](#_Toc141628414)

[3.3 UI (User Interface Layer) 10](#_Toc141628415)

[4. Class Details 10](#_Toc141628416)

[4.1 Category class: 10](#_Toc141628417)

[Responsibilities: 10](#_Toc141628418)

[4.2 Product class: 10](#_Toc141628419)

[Responsibilities: 10](#_Toc141628420)

[4.3 Person class: 10](#_Toc141628421)

[Responsibilities 10](#_Toc141628422)

[4.4 Owner class: 11](#_Toc141628423)

[Responsibilities: 11](#_Toc141628424)

[4.5 User Class: 11](#_Toc141628425)

[Responsibilities: 11](#_Toc141628426)

[4.6 Employee class: 11](#_Toc141628427)

[Responsibilities: 11](#_Toc141628428)

[4.7 Customer class: 11](#_Toc141628429)

[Responsibilities: 11](#_Toc141628430)

[5. Code 12](#_Toc141628431)

[5.1 Business Logic 12](#_Toc141628432)

[5.1.1 Person 12](#_Toc141628433)

[5.1.2 Admin 12](#_Toc141628435)

[5.1.3 User 13](#_Toc141628436)

[5.1.4 Customer 13](#_Toc141628437)

[5.1.5 Employee 14](#_Toc141628438)

[5.1.6 Product 15](#_Toc141628439)

[5.1.7 Category 16](#_Toc141628440)

[5.2 Data Layer 16](#_Toc141628441)

[5.2.1 AdminDL 16](#_Toc141628442)

[5.2.2 CategoriesDL 17](#_Toc141628443)

[5.2.3 CustomerDL 20](#_Toc141628444)

[5.2.4 EmployeeDL 23](#_Toc141628445)

[5.2.5 ExtraDLClass 24](#_Toc141628446)

[5.3 GUI 25](#_Toc141628447)

[5.3.1 Main Forms 25](#_Toc141628448)

[5.3.2 Admin Forms 27](#_Toc141628449)

[5.3.3 Customer Forms 46](#_Toc141628450)

[6. Conclusion: 60](#_Toc141628451)

[6.1 Summary 60](#_Toc141628452)

[6.2 Achievements 61](#_Toc141628453)

[6.3 Challenges Faced 62](#_Toc141628459)

[6.4 Lessons Learned 62](#_Toc141628466)

# Video Link

<https://youtu.be/V7GWD8s1vgk>

# Introduction:

## 1.1 Project Overview

The objective of the "**Mobile Shop Management System**" is to help small shop owners efficiently handle their daily records. Shop owners can manage their records more effectively and take advantage of technology in their daily operations. The Basic purpose of the system aims to automate various tasks involved in running a mobile shop, including Inventory management, employee management, sales tracking and customer management.

## 2.2 Objectives

The primary objectives of the Mobile Shop Management System are as follows:

2.2.1. Efficient Inventory Management:

* Enable real-time tracking of mobile devices, accessories ensuring accurate inventory control.

2.2.2. Customer Relationship Management:

* + - Maintain a comprehensive customer database, including contact details, purchase history.
    - Give some recommendation of bestselling products.

2.2.3. Sales Process:

* Automate the sales process, including generating invoices, tracking customer purchases.

### 2.2.4. Employee Management

* Manage Employee Details including their personal info, attendance and salary.

## 2.3 Functionality

The Mobile Shop Management System will offer the following key functionalities:

* + 1. Inventory Management:
* Real-time tracking of mobile devices and accessories, including product details, stock levels, and prices.
* Validating the input before saving the information.

### 2.3.2 Sales:

* Sales history tracking
* Total Profit history.
* Keeping the history of customers and detail of products in which customer showed interest.

### 2.3.3 Customer Management:

* Customer registration and maintenance of contact details.
* Keeping in record the detail of products in which customer is interested.
* Recommending bestselling items to the customer.
* Recording of customer purchases

# OOP Concepts

Let's discuss how the project incorporates the concepts of association, inheritance, and polymorphism,

## 3.1 Association:

There is association in this system and explained as below:

### 3.1.1 Composition

* **Customer and Product Cart:**

The Customer class has a composition relationship with the cart that is the list of product class. Each Customer is associated with a specific cart items, and this cannot exist independently of the customer.

* **Customer and Product History:**

The Customer class exhibits composition with the ProductHistory which is the list of product class. Each Customer has a unique ProductHistory which is also the list of product class, representing their purchase history. The product history is tightly bound to the customer, and it cannot exist independently.

* **Categories and Product:**

The Category class and the Product class are associated to reflect the relationship between product items and their respective categories. Each Product object is linked to a specific Category object, indicating that products belong to specific categories.

If a category is deleted than all the products in it will be deleted as well.

### 3.1.2 Aggregation

* **Admin and Employees:**

Admin class and the Employee class are associated to establish a connection between administrators and employees.

The Admin class might interact with multiple instances of the Employee class to manage employees and their details.

* **Admin and Categories or Products:**

There is aggregation between admin and categories class as admin can perform various actions related to products or categories.

Admin have the ability to add, edit or delete a new category or he can also add the products in the existing category.

## 3.2 Inheritance:

In the Mobile Shop Management System, we have implemented a class hierarchy using the concept of inheritance in Object-Oriented Programming (OOP). The Persons class serves as the superclass, while the subclasses Admin and User are derived from it. Additionally, two further subclasses, Customer and Employee, are inherited from the User class.

By using inheritance my project, I achieved a modular and organized design, ensuring that common features are centralized in the superclass and subclasses can extend or customize functionality based on their roles.

## 3.3 Polymorphism

There is Static and Dynamic Polymorphism in my project.

Static Polymorphism is observed in dispalying the cart of customer and to view the stock of Shop.

## 3.4 Comparison with Procedural Programming

* Object oriented programming uses classes and objects to model real-world entities and their interactions while procedural programming focuses on procedures or functions that manipulate data step by step.
* In object oriented programming, the code is more manageable and well designed and by developing relations between classes we can make backend design better.
* Object Oriented Programming encourages data encapsulation and abstraction, hiding implementation details from outside and the security of data is ensured by class itself but procedural programming does not provide this property..
* Object Oriented Programming enables polymorphism, allowing objects of different classes to be treated interchangeably but in procedural programming code organization is based on functions, not class-based modularity.

# Design Pattern Implementation:

## 4.1 BL (Business Layer)

This project contains the BL Design Pattern in such a way that all the BL classes contains the Business logic functions and all the attributes are declared in it which are kept private for security purposes and all getter( ) and setter( ) functions are also included in this layer.

The Business Logic (BL) layer serves as the heart of my project. It encapsulates the essential functionalities and algorithms responsible for processing and managing specific tasks. This layer acts as an intermediary between the UI and DL layers, ensuring that business rules and logic are separate from presentation and data handling concerns.

The Classes in BL are:

* Person.cs
* Admin.cs
* User.cs
* Employee.cs
* Customer.cs
* Product.cs
* Category.cs

## 4.2 DL (Data Layer)

The Data Layer (DL) plays a crucial role in our project's execution. It is responsible for handling data storage, retrieval, and manipulation. By abstracting the data access from the Business Logic and User Interface layers, we ensure that the core functionalities remain unaffected by changes in data sources or databases. This layer enables seamless integration with various data storage solutions while maintaining data consistency and integrity.

This project utilizes the DL Design Pattern in such a way that the DL classes contain all the static Lists, and all functions related to Lists and all the functions of classes.

The classes in DL are:

* + - AdminDL.cs
    - EmployeeDL.cs
    - CustomerDL.cs
    - CategoriesDL,cs
    - ExtraDLClass.cs

The "extra DL class" is a class that serves as a static container or repository for storing and managing data during runtime. This class is designed to improve data manipulation efficiency and facilitate data retrieval and storage for specific entities, such as customers, admin, product etc.

This class contains the static members and stores the information during the runtime that can be accessed throughout the application without creating an instance of the class.

## 4.3 UI (User Interface Layer)

The User Interface (UI) layer serves as the front-end of our project. It focuses on providing an intuitive and user-friendly interface for our users to interact with the system. This layer collects user inputs, displays relevant information, and communicates with the Business Logic layer and Data Logic Layer to execute user requests.

# Class Details

## 5.1 Category class:

### Responsibilities:

* + Represents a category or classification for products in the mobile shop.
  + Stores information such as name, and list of products.
  + Manages the association between products and their respective categories.

## 5.2 Product class:

## Responsibilities:

* + Represents a product available in the shop.
  + Stores information such as name, price, quantity, actual price.
* Manages inventory and tracks product availability.

## 5.3 Person class:

Responsibilities:

* Serves as a base class for other classes related to individuals in the system.
* Stores common attributes such as name and password.
* Provides basic functionality shared among classes like Employee, Customer, and Owner.

## 5.4 Owner class:

## Responsibilities:

* + Represents the owner or manager of the mobile shop.
  + Inherits from the Person class and adds specific attributes such as profit or methods related to ownership or management responsibilities.
  + Manages overall shop operations, employee management, and products management.

## 5.5 User Class:

## Responsibilities:

* + It is inherited from Person class and contains specific attributes.
  + Stores information such as phone number.
  + Provides basic functionality shared among Employee and Customer

## 5.6 Employee class:

## Responsibilities:

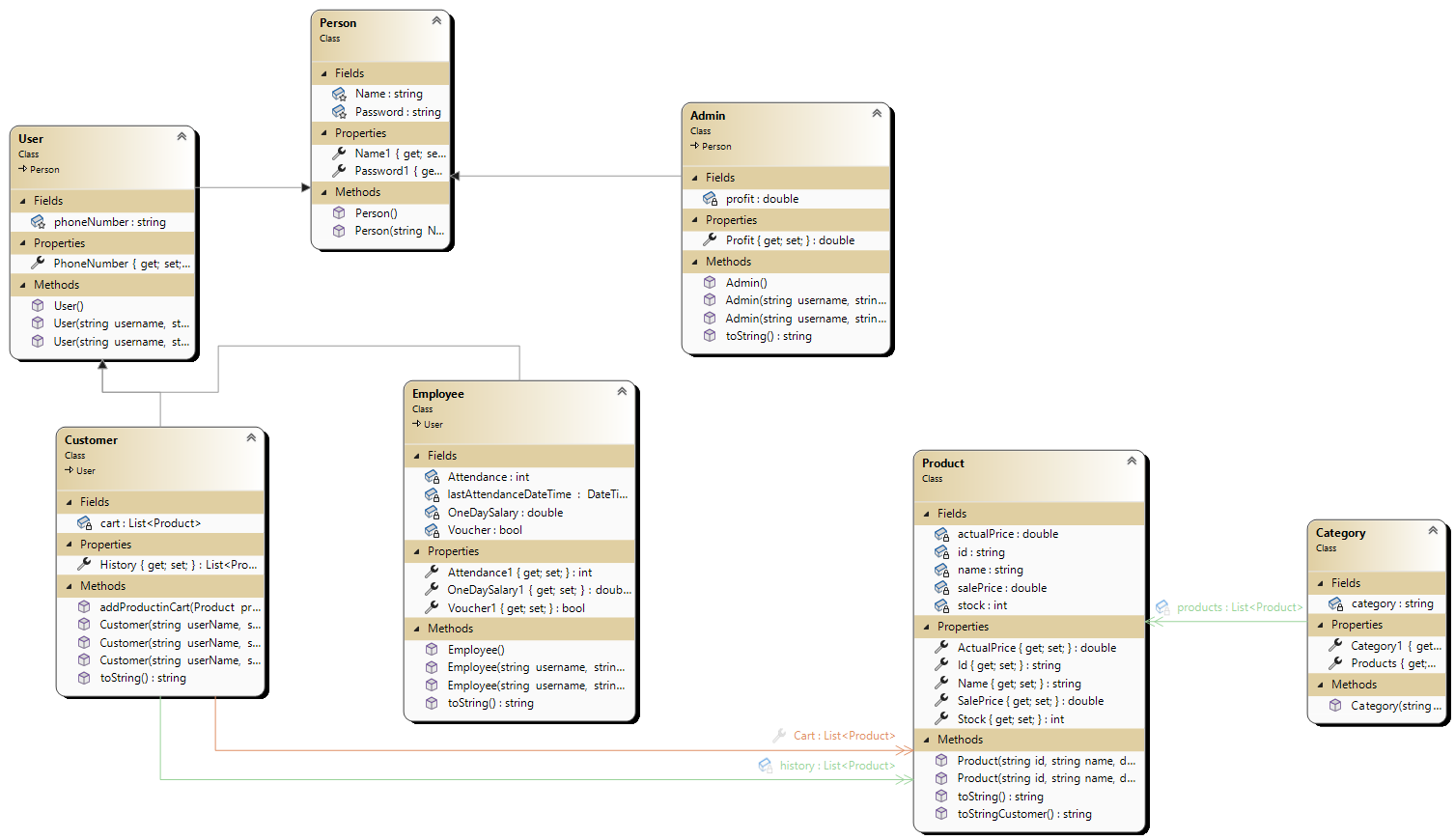
* + Represents an employee working in the mobile shop.
  + Inherits from the Person class and adds specific attributes like, Salary Vouchers and attendance.
  + Manages employee-related operations such as attendance, managing customers.

## 5.7 Customer class:

## Responsibilities:

* + Represents a customer of the mobile shop.
  + Inherits from the Person class and adds specific attributes like ,cart and purchase history.
  + Manages customer-related operations such as buying products and viewing cart.

# CRC Diagram



# 7. Code:

## 7.1 Business Logic

### 7.1.1 Person

public class Person

{

public Person() { }

public Person(string Name, string Password)

{

this.Name = Name;

this.Password = Password;

}

protected string Name;

protected string Password;

public string Name1 { get => Name; set => Name = value; }

public string Password1 { get => Password; set => Password = value; }

### }

### 7.1.2 Admin

public class Admin:Person

{

public Admin() { }

public Admin(string username,string password):base(username,password)

{

}

public Admin(string username, string password, double profit):this(username,password)

{

this.Profit = profit;

}

private double profit;

public double Profit { get => profit; set => profit = value; }

public string toString()

{

return this.Name1 + "," + this.Password1 + "," + this.Profit;

}

}

### 7.1.3 User

public class User:Person

{

public User()

{

}

public User(string username,string password,string PhoneNumber) :base(username,password)

{

this.PhoneNumber = PhoneNumber;

}

public User(string username, string password):base(username, password)

{

}

protected string phoneNumber;

public string PhoneNumber { get => phoneNumber; set => phoneNumber = value; }

}

### 7.1.4 Customer

public class Customer:User

{

public Customer(string userName, string Password):base(userName,Password)

{

}

public Customer(string userName,string Password,string Number):base(userName,Password,Number)

{

Cart = new List<Product>();

history = new List<Product>();

}

public Customer(string userName,string Password,string Number,List<Product> cart,List<Product> history):base(userName,Password,Number)

{

this.Cart = cart;

this.history= history;

}

private List<Product> cart;

private List<Product> history;

public List<Product> Cart { get => cart; set => cart = value; }

public List<Product> History { get => history; set => history = value; }

public void addProductinCart(Product product)

{

Cart.Add(product);

}

public string toString()

{

string line = this.Name + "," + this.Password + "," + this.phoneNumber +",";

foreach(Product product in Cart)

{

line = line + "#" + product.toStringCustomer();

}

line = line + ",";

foreach(Product product in history)

{

line = line + "#" + product.toStringCustomer();

}

return line;

}

}

### 7.1.5 Employee

public class Employee:User

{

public Employee(string username,string password,string phonenumber,int attendance,double OneDaySalary,

bool Voucher):base(username,password,phonenumber)

{

this.Attendance = attendance;

this.OneDaySalary = OneDaySalary;

this.Voucher = Voucher;

lastAttendanceDateTime= DateTime.Now;

}

public Employee(string username, string password, string phonenumber, int attendance, double OneDaySalary, bool Voucher,DateTime lastAttendanceDateTime) : base(username, password, phonenumber)

{

this.Attendance = attendance;

this.OneDaySalary = OneDaySalary;

this.Voucher = Voucher;

this.lastAttendanceDateTime = lastAttendanceDateTime;

}

private bool Voucher;

private int Attendance;

private double OneDaySalary;

private DateTime lastAttendanceDateTime;

public bool Voucher1 { get => Voucher; set => Voucher = value; }

public int Attendance1 { get => Attendance; set => Attendance = value; }

public double OneDaySalary1 { get => OneDaySalary; set => OneDaySalary = value; }

public string toString()

{

return this.Name + "," + this.Password + "," + this.PhoneNumber + "," + this.Attendance + "," + this.OneDaySalary + "," + this.Voucher+","+lastAttendanceDateTime.ToString();

}

}

### 7.1.6 Product

public class Product

{

private string id;

private string name;

private double actualPrice;

private double salePrice;

private int stock;

public Product(string id, string name, double actualPrice, double salePrice, int stock)

{

Id = id;

Name = name;

ActualPrice = salePrice;

SalePrice = actualPrice;

Stock = stock;

}

public Product(string id, string name, double salePrice, int stock)

{

Id = id;

Name = name;

this.salePrice = salePrice;

Stock = stock;

}

public string Id { get => id; set => id = value; }

public string Name { get => name; set => name = value; }

public int Stock { get => stock; set => stock = value; }

public double ActualPrice { get => actualPrice; set => actualPrice = value; }

public double SalePrice { get => salePrice; set => salePrice = value; }

public string toString()

{

return id + "," + name + "," + actualPrice + "," + salePrice + "," + stock;

}

public string toStringCustomer()

{

return id + ";" + name + ";" + salePrice + ";" + stock;

}

}

### 7.1.7 Category

public class Category

{

public Category(string category, List<Product> products)

{

Category1 = category;

Products = products;

}

private string category;

private List<Product> products;

public List<Product> Products { get => products; set => products = value; }

public string Category1 { get => category; set => category = value; }

}

## 7.2 Data Layer

### 7.2.1 AdminDL

class AdminDL

{

private static Admin admin= new Admin();

public static Admin Admin { get => admin; set => admin = value; }

public static void changeUsername(string username)

{

admin.Name1 = username;

}

public static void changePassword(string password)

{

admin.Password1 = password;

}

public static void readData()

{

StreamReader myFile = new StreamReader("Admin.txt");

string line;

if (File.Exists("Admin.txt"))

{

while ((line = myFile.ReadLine()) != null)

{

string[] temp = line.Split(',');

Admin a = new Admin(temp[0], temp[1], double.Parse(temp[2]));

admin = a;

}

myFile.Close();

}

}

public static void writeData()

{

StreamWriter myFile = new StreamWriter("Admin.txt");

if(File.Exists("Admin.txt"))

{

myFile.WriteLine(Admin.Name1 + "," + Admin.Password1);

}

myFile.Flush();

myFile.Close();

}

}

### 7.2.2 CategoriesDL

public class CategoriesDL

{

private static string path = "Categories.txt";

private static List<Category> categories=new List<Category>();

public static List<Category> Categories { get => categories; set => categories = value; }

public static void readData()

{

StreamReader myFile = new StreamReader(path);

string line = "";

while((line = myFile.ReadLine()) != null)

{

string[] temp = line.Split(',');

Product product = new Product(temp[1], temp[2], double.Parse(temp[3]), double.Parse(temp[4]), int.Parse(temp[5]));

// Check if the category already exists

string categoryName = temp[0];

Category existingCategory = categories.Find(c => c.Category1 == categoryName);

if (existingCategory != null)

{

existingCategory.Products.Add(product);

}

else

{

List<Product> products = new List<Product> { product };

Category newCategory = new Category(categoryName, products);

categories.Add(newCategory);

}

}

myFile.Close();

}

public static void writeData()

{

StreamWriter myFile = new StreamWriter(path);

for (int i = 0; i < categories.Count(); i++)

{

for (int j = 0; j < categories[i].Products.Count; j++)

{

myFile.WriteLine(categories[i].Category1.ToString() + "," + categories[i].Products[j].toString());

}

}

myFile.Flush();

myFile.Close();

}

public static List<Product> GetProducts(string category)

{

List<Product> a = new List<Product>();

for (int idx = 0; idx < categories.Count; idx++)

{

if (categories[idx].Category1.ToLower() == category.ToLower())

{

return categories[idx].Products;

}

}

return a;

}

public static bool isCategoryExist(string name)

{

foreach (Category category in categories)

{

if (category.Category1.ToLower() == name)

{

return true;

}

}

return false;

}

public static int getCategoryIndex(string category)

{

for (int idx = 0; idx < categories.Count; idx++)

{

if (categories[idx].Category1.ToLower() == category.ToLower())

{

return idx;

}

}

return -1;

}

public static int getProductIndex(int category,string productId)

{

List<Product> products = categories[category].Products;

for (int i = 0; i < products.Count;i++)

{

if (productId == products[i].Id)

{

return i;

}

}

return -1;

}

public static void addCategory(Category category)

{

categories.Add(category);

}

public static Product GetProduct(int CategoryIdx,int ProductIdx)

{

return categories[CategoryIdx].Products[ProductIdx];

}

public static Product GetProduct(Category category, Product product)//from customer data

{

foreach(Category c in Categories)

{

if(c.Category1== category.Category1)

{

foreach (Product p in c.Products)

{

if(p.Id==product.Id&&p.Name==product.Name)

{

return p;

}

}

}

}

return null;

}

public static void AddProduct(int categoryIdx,Product product)

{

categories[categoryIdx].Products.Add(product);

}

public static void DeleteProduct(int categoryIdx,Product product)

{

categories[categoryIdx].Products.Remove(product);

}

public static void DeleteProduct(int categoryIdx,int productIdx)

{

categories[categoryIdx].Products.RemoveAt(productIdx);

}

public static void UpdateProduct(int categoryIdx,int productIdx,Product product)

{

categories[categoryIdx].Products.Insert(productIdx, product);

categories[categoryIdx].Products.RemoveAt(productIdx+1);

}

public static Category GetCategory(Product cartProduct)

{

foreach (var category in Categories)

{

var product = category.Products.Find(p => p.Id == cartProduct.Id && p.Name == cartProduct.Name);

if (product != null)

{

return category;

}

}

return null;

}

}

### 7.2.3 CustomerDL

public class CustomerDL

{

private static string path = "Customers.txt";

private static List<Customer> customers = new List<Customer>();

public static List<Customer> Customers { get => customers; set => customers = value; }

public static void writeData()

{

StreamWriter myFile = new StreamWriter(path);

if (File.Exists(path))

{

for (int i = 0; i < customers.Count; i++)

{

if (i == customers.Count - 1)

{

myFile.Write(customers[i].toString());

}

else

{

myFile.WriteLine(customers[i].toString());

}

}

}

myFile.Flush();

myFile.Close();

}

public static void readData()

{

StreamReader myFile = new StreamReader(path);

string line = " ";

while ((line = myFile.ReadLine()) != null)

{

string[] temp = line.Split(',');

List<Product> cart = new List<Product>();

List<Product> history = new List<Product>();

if (temp.Length >= 4)

{

string[] cartItems = temp[3].Split('#');

for (int j = 0; j < cartItems.Length; j++)

{

string[] temp2 = cartItems[j].Split(';');

for (int i = 0; i < temp2.Length - 1; i += 4)

{

Product product = new Product(temp2[i], temp2[i + 1], double.Parse(temp2[i + 2]), int.Parse(temp2[i + 3]));

cart.Add(product);

}

}

if (temp.Length >= 5)

{

string[] HistoryItem = temp[4].Split('#');

for (int j = 0; j < HistoryItem.Length; j++)

{

string[] temp2 = HistoryItem[j].Split(';');

for (int i = 0; i < temp2.Length - 1; i += 4)

{

Product product = new Product(temp2[i], temp2[i + 1], double.Parse(temp2[i + 2]), int.Parse(temp2[i + 3]));

history.Add(product);

}

}

}

}

Customer customer = new Customer(temp[0], temp[1], temp[2], cart, history);

addCustomer(customer);

}

myFile.Close();

}

public static void addCustomer(Customer customer)

{

customers.Add(customer);

}

public static int getCustomerIndex(Customer customer)

{

for (int i = 0; i < customers.Count; i++)

{

if (customers[i].Name1 == customer.Name1 && customers[i].Password1 == customer.Password1)

{

return Customers.IndexOf(customers[i]);

}

}

return -1;

}

public static Customer GetCustomer(int index)

{

return customers[index];

}

public static bool isCustomerExist(Customer customer)

{

foreach (Customer customer1 in customers)

{

if (customer1 == customer)

{

return true;

}

}

return false;

}

}

### 7.2.4 EmployeeDL

public class EmployeeDL

{

private static string path = "Employees.txt";

private static List<Employee> employees = new List<Employee>();

//Store Data in File

public static void writeData()

{

StreamWriter myFile = new StreamWriter(path);

if (File.Exists(path))

{

foreach (Employee employee in employees)

{

if (Employees.IndexOf(employee) != Employees.Count - 1)

{

myFile.WriteLine(employee.toString());

}

else

{

myFile.Write(employee.toString());

}

}

}

myFile.Flush();

myFile.Close();

}

public static void writeData(Employee emp)

{

StreamWriter myFile = new StreamWriter(path,true);

if (File.Exists(path))

{

myFile.WriteLine(emp.toString());

}

myFile.Flush();

myFile.Close();

}

//Read Data from file

public static void readData()

{

StreamReader myFile = new StreamReader(path);

string line = " ";

while ((line = myFile.ReadLine()) != null)

{

string[] temp = line.Split(',');

Employee employee = new Employee(temp[0],temp[1],temp[2],int.Parse(temp[3]),double.Parse(temp[4]),boo l.Parse(temp[5]), DateTime.Parse(temp[6]));

employees.Add(employee);

}

myFile.Close();

}

public static void addEmployee(Employee emp)

{

employees.Add(emp);

}

public static int EmployeeExists(string username,string password)

{

foreach(Employee employee in employees)

{

if((employee.Name1==username) && ( employee.Password1 == password))

{

return employees.IndexOf(employee);

}

}

return -1;

}

public static Employee GetEmployee(int idx)

{

return employees[idx];

}

public static void removeEmployee(int idx)

{

employees.RemoveAt(idx);

}

public static void updateEmployeeInfo(int idx,Employee emp)

{ employees.Insert(idx, emp);

employees.RemoveAt(idx+1);}

public static List<Employee> Employees { get => employees; set => employees = value; } }

### 7.2.5 ExtraDLClass

public class ExtraDLClass

{

private static Customer customer;

public static Customer Customer { get => customer; set => customer = value; }

private static int Customeridx;

public static int getCustomerIdx()

{ Customeridx = CustomerDL.getCustomerIndex(Customer);

return Customeridx;}}

## 7.3 GUI

### 7.3.1 Main Forms

#### 7.3.1.1 Splash

public partial class Splash : Form

{ public Splash()

{ InitializeComponent(); }

int startPoint = 0;

private void timer\_tick(object sender, EventArgs e)

{ startPoint++;

VProgressBar.Value = startPoint;

HProgressBar.Value = startPoint;

if (HProgressBar.Value == 100 && VProgressBar.Value == 100)

{ VProgressBar.Value = startPoint;

HProgressBar.Value = startPoint;

timer.Stop();

GUI.Main.MainMenu menu = new GUI.Main.MainMenu();

menu.Show();

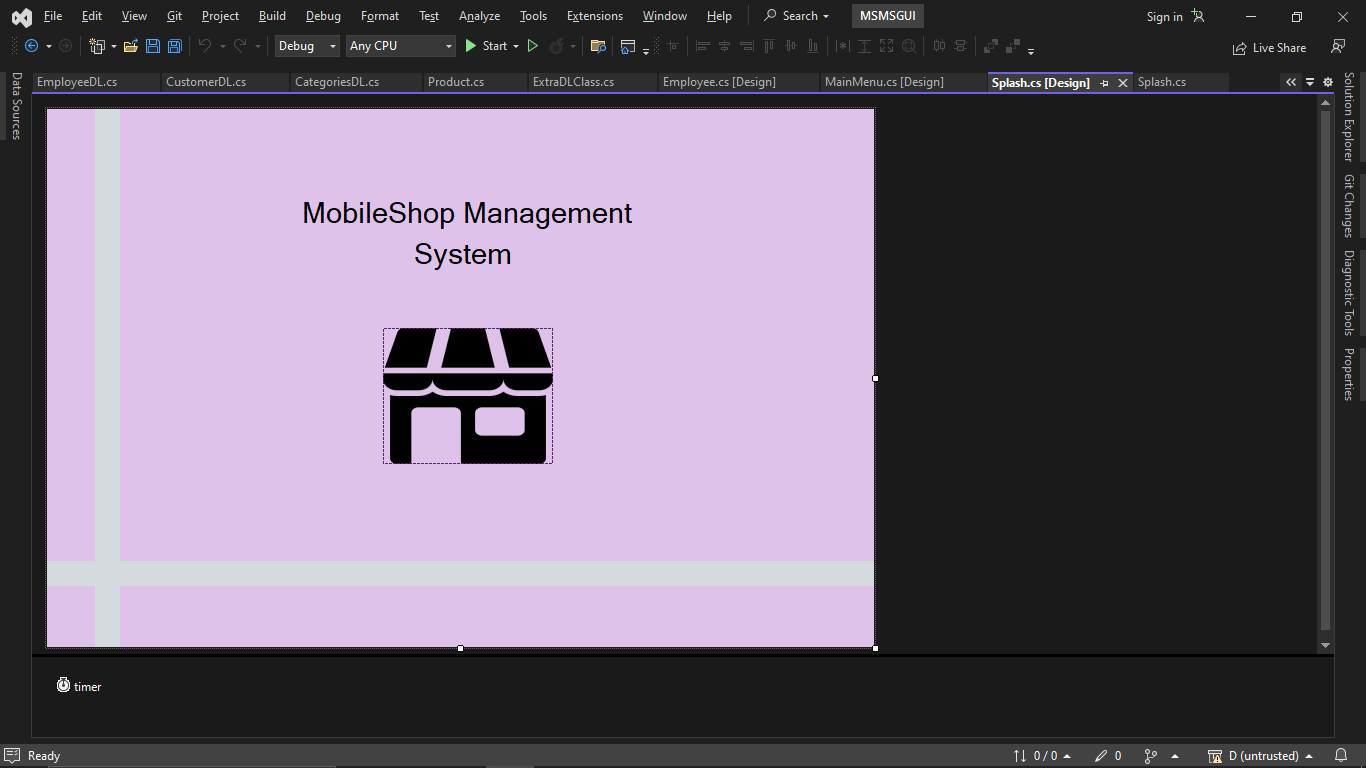
 this.Hide(); } }}

Figure 1: Loading Form

#### 7.3.1.2 Main Menu

public partial class MainMenu : Form

{ public MainMenu()

{ InitializeComponent();

}

private void BtnAdmin\_Click(object sender, EventArgs e)

{

SignIn signIn = new SignIn();

MainMenuRightPnl.Controls.Clear();

MainMenuRightPnl.Controls.Add(signIn);

}

private void BtnCustomer\_Click(object sender, EventArgs e)

{

CustomerSignIn customerSignIn = new CustomerSignIn();

MainMenuRightPnl.Controls.Clear();

MainMenuRightPnl.Controls.Add(customerSignIn);

}

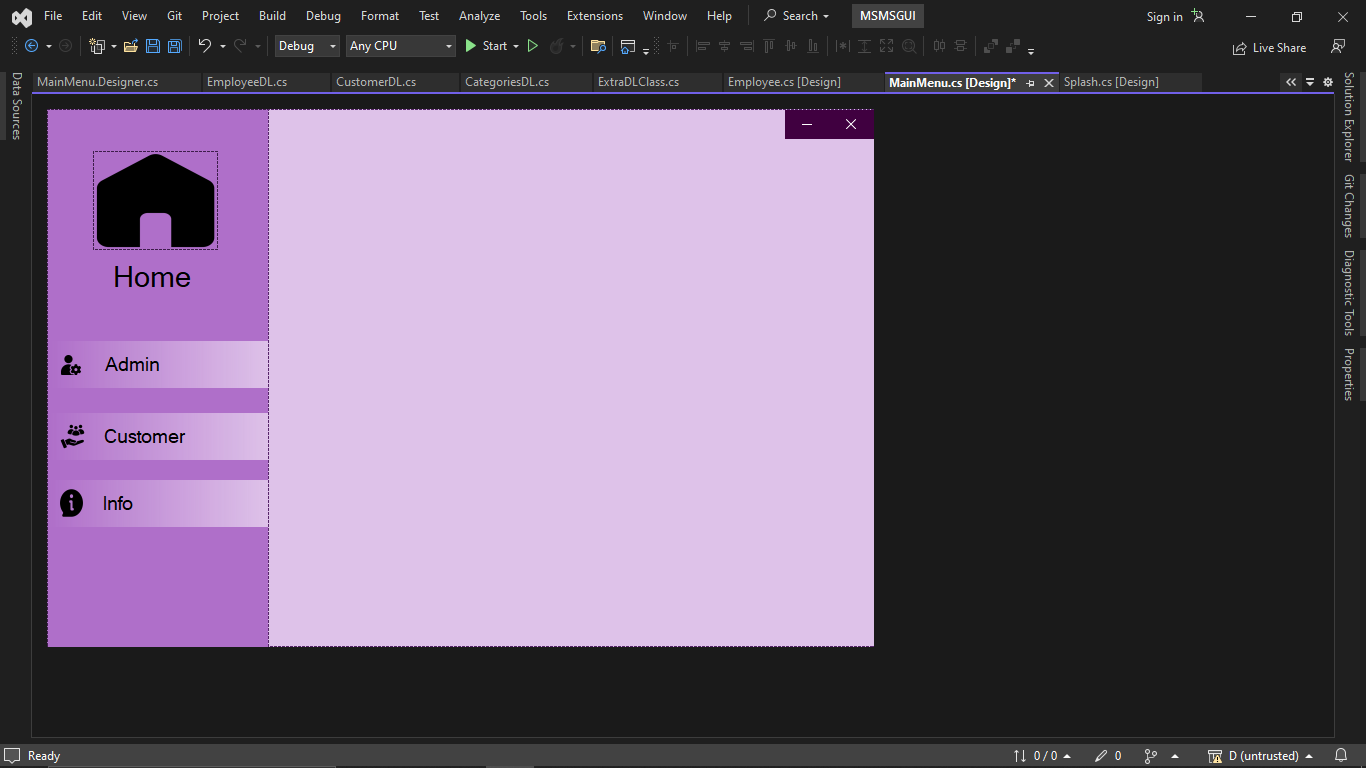
 }

Figure 2:Main Menu

### 7.3.2 Admin Forms

#### 7.3.2.1 Personal Info

public partial class AdminPersonalInfo : UserControl

{

public AdminPersonalInfo()

{ InitializeComponent();

txtUsername.Text = MSMSGUI.DL.AdminDL.Admin.Name1;

txtPassword.Text = MSMSGUI.DL.AdminDL.Admin.Password1;

}

private void btnEdit\_Click(object sender, EventArgs e)

{ EditAdminPersonalInfo editAdminPersonalInfo = new EditAdminPersonalInfo();

MainPanel.Controls.Clear();

MainPanel.Controls.Add(editAdminPersonalInfo); }

private void ToggleSwitchViewPassword\_CheckedChanged(object sender, EventArgs e)

{

if (ToggleSwitchViewPassword.Checked)

{

txtPassword.PasswordChar = '\0';

}

else

{ txtPassword.PasswordChar = '\*';

}} }

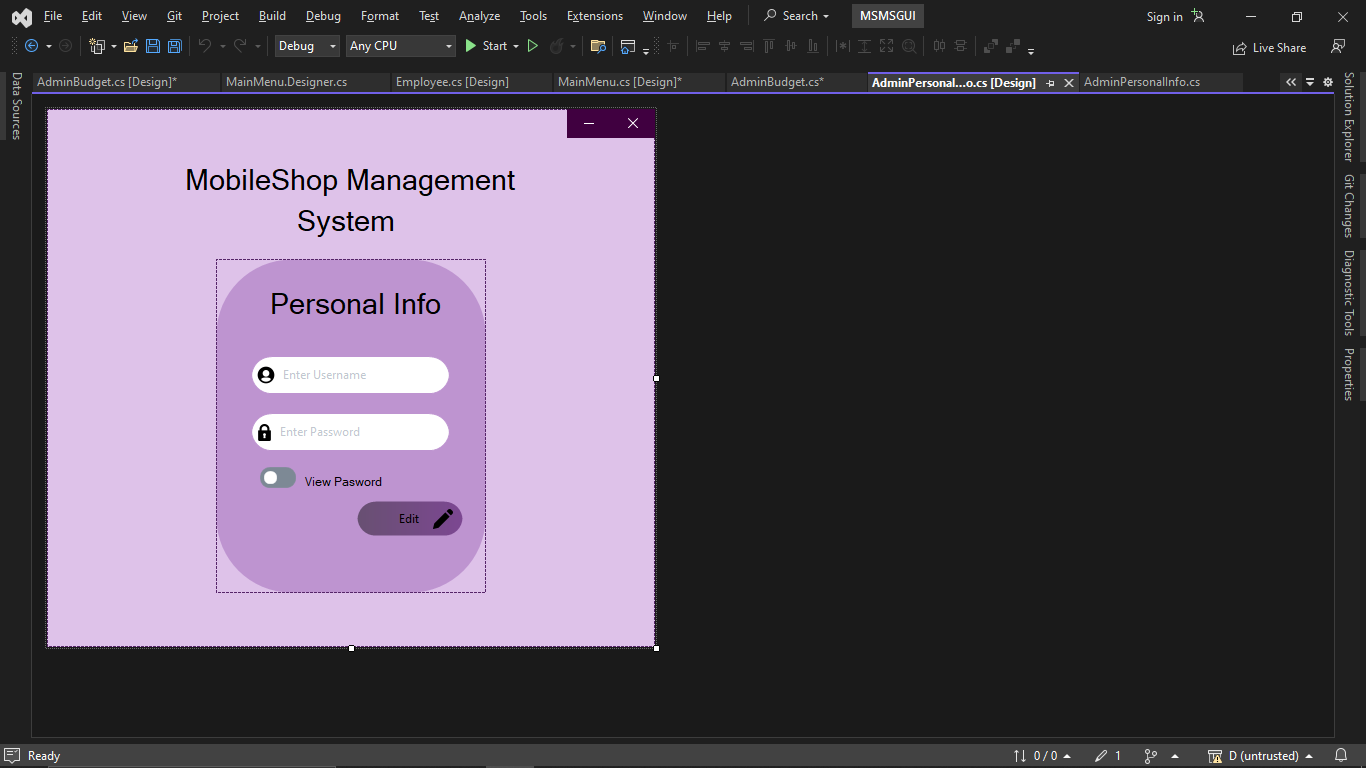


Figure 3:Admin Personal Info UserControl

#### 7.3.2.2 Edit Personal Info

public partial class EditAdminPersonalInfo : UserControl

{

public EditAdminPersonalInfo()

{

InitializeComponent();

btnCancel.Enabled = true;

timer.Enabled = true;

txtUsername.Text = AdminDL.Admin.Name1;

}

private void timer\_Tick(object sender, EventArgs e)

{

if(Validations.IsValidUsername(txtUsername.Text)&& Validations.IsValidPassword(txtPassword.Text))

{

btnUpdate.Enabled = true;

}

else

{

btnUpdate.Enabled = false;

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

timer.Enabled = false;

AdminPersonalInfo adminPersonalInfo = new AdminPersonalInfo();

MainPanel.Controls.Clear();

MainPanel.Controls.Add(adminPersonalInfo);

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

BL.Admin admin =new BL.Admin(txtUsername.Text,txtPassword.Text);

AdminDL.Admin = admin;

AdminDL.writeData();

MessageBox.Show("Updated Successfully");

AdminPersonalInfo adminPersonalInfo=new AdminPersonalInfo();

MainPanel.Controls.Clear();

MainPanel.Controls.Add(adminPersonalInfo);

}

private void ToggleSwitchViewPassword\_CheckedChanged(object sender, EventArgs e)

{ if (ToggleSwitchViewPassword.Checked)

{ txtPassword.PasswordChar = '\0'; }

else

{ txtPassword.PasswordChar = '\*'; }}

private void txtUsername\_TextChanged(object sender, EventArgs e)

{ if (!Validations.IsValidUsername(txtUsername.Text))

{

errorProvider.SetError(txtUsername, "Username Can not be Empty\nUsername only consist of alphabets\nUsername length must be greater that 8 and less than 20\n Username must contain at least one lowercase letter, one uppercase letter...");

}

else

{ errorProvider.Clear(); }}

private void txtPassword\_TextChanged(object sender, EventArgs e)

{if (!Validations.IsValidPassword(txtPassword.Text))

{errorProvider1.SetError(txtPassword, "Password cannot be empty\nPassword length must be greater that 8 and less than 20\nPassword must contain at least one lowercase letter, one uppercase letter, and one digit");}

else

{errorProvider1.Clear();}}

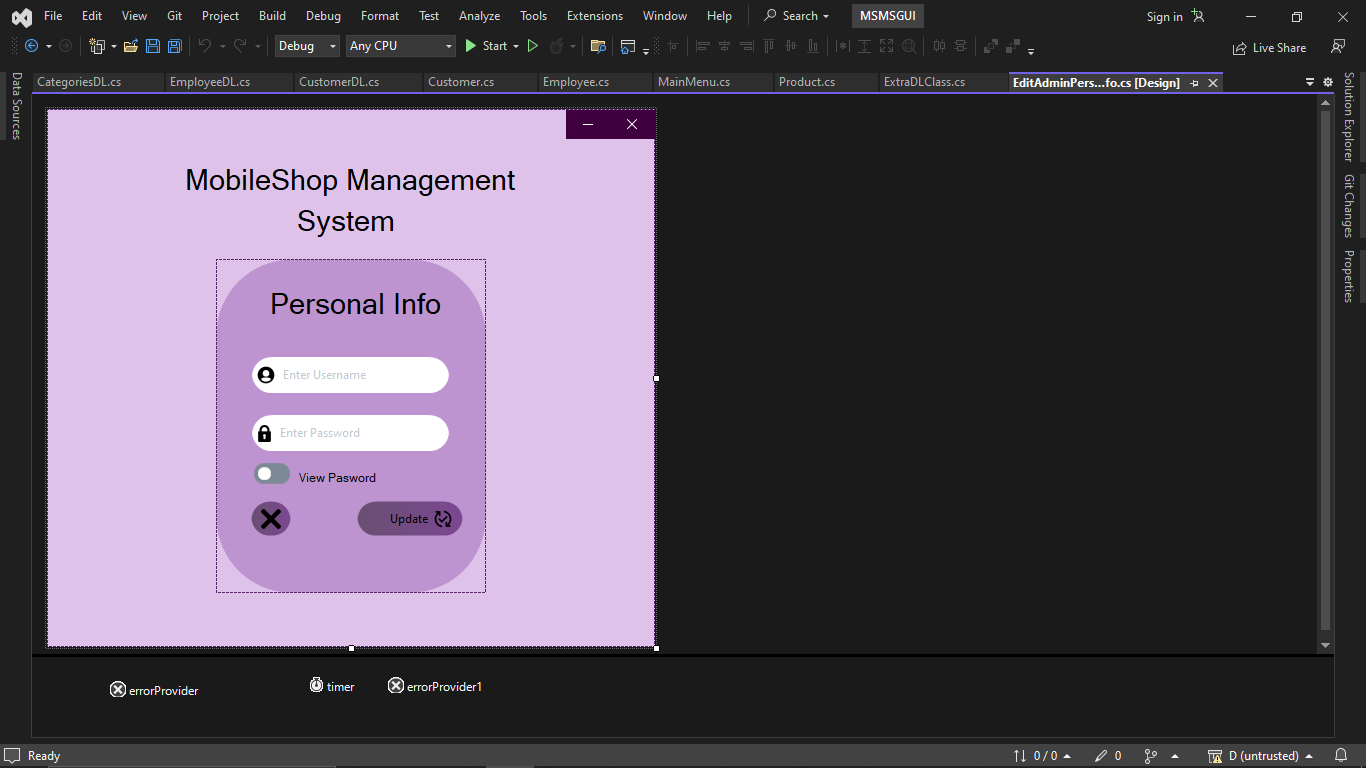


Figure 4:Edit Personal Info

#### 7.3.2.3 Employees

public partial class AdminEmployees : UserControl

{

private int employeeidx;

private Employee Employee=new Employee();

private List<Employee> Employees=EmployeeDL.Employees;

public AdminEmployees()

{

InitializeComponent();

timer.Enabled = true;

btnCancel.Enabled = true;

InitializeDataTable();

dataBinding();

}

private void AddEmployees\_Load(object sender, EventArgs e)

{

}

private void Username\_txtChanged(object sender, EventArgs e)

{

if (!Validations.IsValidUsername(txtUsername.Text))

{

errorProvider.SetError(txtUsername, "Enter Valid Username");

}

else

{

errorProvider.Clear();

}

}

private void Password\_txtChanged(object sender, EventArgs e)

{

if (!Validations.IsValidPassword(txtPassword.Text))

{

errorProvider.SetError(txtPassword, "Enter Valid Password");

}

else

{

errorProvider.Clear();

}

}

private void timer\_Tick(object sender, EventArgs e)

{

employeeidx = EmployeeDL.EmployeeExists(txtUsername.Text, txtPassword.Text);

if (employeeidx != -1)

{

Employee = EmployeeDL.GetEmployee(employeeidx);

txtPhoneNumber.Text = Employee.PhoneNumber;

txtAttendance.Text = Employee.Attendance1.ToString();

txtOneDaySalary.Text = Employee.OneDaySalary1.ToString();

txtTotalSalary.Text = (Employee.Attendance1 \* Employee.OneDaySalary1).ToString();

checkboxVoucher.Checked = Employee.Voucher1;

EnableButtons(false, true, true, true, true, true);

}

else if (employeeidx == -1)

{

//all inputs are valid and not existing employee

if (validInputs())

{

EnableButtons(true, false, false, true, true, true);

}

else

{

EnableButtons(false, false, false, true, true, true);

}

}

}

private void Phonenumber\_txtChanged(object sender, EventArgs e)

{

if (!Validations.IsValidPhoneNumber(txtPhoneNumber.Text))

{

errorProvider.SetError(txtPhoneNumber, "Enter Valid PhoneNumber");

}

else

{

errorProvider.Clear();

}

}

private void Attendance\_txtChanged(object sender, EventArgs e)

{

if (!Validations.isValidNumber(txtAttendance.Text))

{

errorProvider.SetError(txtAttendance, "Enter Valid Integer Value");

}

else

{

errorProvider.Clear();

}

}

private void oneDaySalary\_txtChanged(object sender, EventArgs e)

{

if (!Validations.isValidNumber(txtOneDaySalary.Text))

{

errorProvider.SetError(txtOneDaySalary, "It can only consist of Numeric Values");

}

else

{

errorProvider.Clear();

}

}

private void txtTotalSalary\_TextChanged(object sender, EventArgs e)

{

if (Validations.isValidNumber(txtAttendance.Text) && Validations.isValidNumber(txtOneDaySalary.Text))

{

txtTotalSalary.Text = (int.Parse(txtAttendance.Text) \* double.Parse(txtOneDaySalary.Text)).ToString();

}

}

private void EnableButtons(bool add, bool edit, bool delete, bool print, bool save, bool update)

{

btnAdd.Enabled = add;

btnDelete.Enabled = delete;

btnEdit.Enabled = edit;

btnSave.Enabled = save;

btnPrint.Enabled = print;

btnUpdate.Enabled = update;

}

private void VisibleButtons(bool add, bool edit, bool delete, bool print, bool save, bool cancel, bool update)

{

btnAdd.Visible = add;

btnDelete.Visible = delete;

btnEdit.Visible = edit;

btnSave.Visible = save;

btnPrint.Visible = print;

btnCancel.Visible = cancel;

btnUpdate.Visible = update;

}

private void TextBoxClear()

{

txtAttendance.Text = null;

txtOneDaySalary.Text = null;

txtPassword.Text = null;

txtPhoneNumber.Text = null;

txtTotalSalary.Text = null;

txtUsername.Text = null;

}

private void btnAdd\_Click(object sender, EventArgs e)

{

Employee employee = new Employee(txtUsername.Text, txtPassword.Text, txtPhoneNumber.Text, int.Parse(txtAttendance.Text), double.Parse(txtOneDaySalary.Text), checkboxVoucher.Checked);

Employee = employee;

EmployeeDL.addEmployee(employee);

MessageBox.Show("Employee Added Successfully\nPress Save Button to Save your Changed Information");

dataBinding();

}

private void btnSave\_Click(object sender, EventArgs e)

{

List<Employee> list = EmployeeDL.Employees;

foreach(Employee employee in list)

{

MessageBox.Show(employee.toString());

}

EmployeeDL.writeData();

MessageBox.Show("You Saved your data Successfully");

}

private void btnEdit\_Click(object sender, EventArgs e)

{

VisibleButtons(false, false, false, false, false, true, true);

btnCancel.Enabled = true;

timer.Enabled = false;

}

private void btnCancel\_Click(object sender, EventArgs e)

{

VisibleButtons(true, true, true, true, true, false, false);

EnableButtons(false, false, false, true, true, true);

TextBoxClear();

}

private void btnDelete\_Click(object sender, EventArgs e)

{

EmployeeDL.removeEmployee(employeeidx);

MessageBox.Show("Employee Removed Succesfully\nPress Save Button to save your changes");

TextBoxClear();

dataBinding();

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

if (validInputs())

{

Employee employee = new Employee(txtUsername.Text, txtPassword.Text, txtPhoneNumber.Text, int.Parse(txtAttendance.Text), double.Parse(txtOneDaySalary.Text), checkboxVoucher.Checked);

EmployeeDL.updateEmployeeInfo(employeeidx, employee);

MessageBox.Show("Update Successfully");

VisibleButtons(true, true, true, true, true, false, false);

TextBoxClear();

dataBinding();

}

timer.Enabled = true; }

private bool validInputs()

{if (Validations.IsValidUsername(txtUsername.Text) && Validations.IsValidPassword(txtPassword.Text) && Validations.IsValidPhoneNumber(txtPhoneNumber.Text) && Validations.isValidNumber(txtAttendance.Text) && Validations.isValidNumber(txtOneDaySalary.Text))

{return true; }

return false; }

private void txtTotalSalary\_Click(object sender, EventArgs e)

{if (Validations.isValidNumber(txtAttendance.Text) && Validations.isValidNumber(txtOneDaySalary.Text))

{txtTotalSalary.Text = (int.Parse(txtAttendance.Text) \* double.Parse(txtOneDaySalary.Text)).ToString();}}

private void dataBinding()

{DataTable.DataSource = null;

DataTable.Rows.Clear();

foreach (var o in EmployeeDL.Employees)

{double Salary = o.Attendance1 \* o.OneDaySalary1;

DataTable.Rows.Add(o.Name1, o.Password1, o.PhoneNumber, o.Attendance1, o.OneDaySalary1, Salary, o.Voucher1);

}

DataTable.Refresh();}

private void InitializeDataTable()

{// Column headers for the DataTable

string[] columnHeaders = { "Name", "Password","PhoneNumber", "Attendance", "OneDaySalary", "Salary", "Voucher" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col); }}

private void fillTextBox()

{txtUsername.Text = Employee.Name1;

txtPassword.Text = Employee.Password1;

txtPhoneNumber.Text = Employee.PhoneNumber;

txtOneDaySalary.Text = Employee.OneDaySalary1.ToString();

txtAttendance.Text = Employee.Attendance1.ToString();

txtTotalSalary.Text = (Employee.OneDaySalary1 \* Employee.Attendance1).ToString();}

private void DataTable\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{employeeidx = DataTable.CurrentRow.Index;

Employee=EmployeeDL.GetEmployee(employeeidx);

fillTextBox();}

private void btnPrint\_Click(object sender, EventArgs e)

{PdfGenerator pdf = new PdfGenerator();

pdf.GeneratePdfReport(Employees);}}

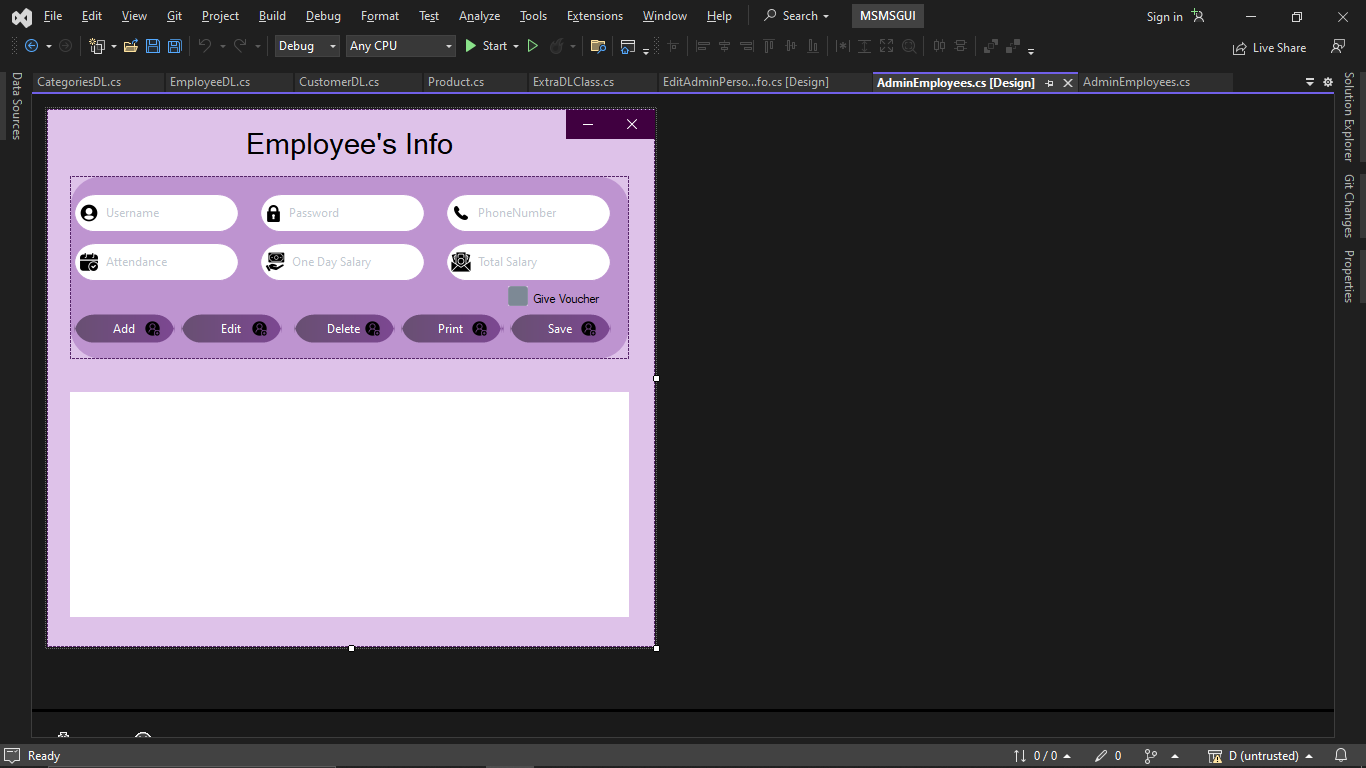


Figure 5: Employee’s Details User Control

#### 7.3.2.4 Shop Products

public partial class AdminProducts : System.Windows.Forms.UserControl

{ int categoryIdx = -1;

int productIdx = -1;

public AdminProducts()

{ InitializeComponent();

timer.Enabled = true;

InitializeDataTable();

dataBinding(); }

private void EnableButtons(bool add, bool edit, bool delete, bool print, bool save, bool update)

{ btnAdd.Enabled = add;

btnDelete.Enabled = delete;

btnEdit.Enabled = edit;

btnSave.Enabled = save;

btnPrint.Enabled = print;

btnUpdate.Enabled = update; }

private void VisibleButtons(bool add, bool edit, bool delete, bool print, bool save, bool cancel, bool update)

{ btnAdd.Visible = add;

btnDelete.Visible = delete;

btnEdit.Visible = edit;

btnSave.Visible = save;

btnPrint.Visible = print;

btnCancel.Visible = cancel;

btnUpdate.Visible = update; }

private void timer\_Tick(object sender, EventArgs e)

{ categoryIdx = CategoriesDL.getCategoryIndex(txtCategory.Text);

if (categoryIdx > -1)

{ productIdx = CategoriesDL.getProductIndex(categoryIdx, txtProductId.Text);

if (productIdx > -1)

{ Product product = CategoriesDL.GetProduct(categoryIdx, productIdx);

txtName.Text = product.Name;

txtQuantity.Text = product.Stock.ToString();

txtSalePrice.Text = product.SalePrice.ToString();

txtActualPrice.Text = product.ActualPrice.ToString();

EnableButtons(false, true, true, true, true, false); }

else

{if (validInputs())

{EnableButtons(true, false, false, true, true, false); }

else

{EnableButtons(false, false, false, true, true, false); }} }

else

{if (validInputs())

{EnableButtons(true, false, false, true, true, false); }

else

{EnableButtons(false, false, false, true, true, false); } } }

private bool validInputs()

{if(Validations.isValidInteger(txtQuantity.Text) && Validations.isValidNumber(txtActualPrice.Text) && Validations.isValidNumber(txtSalePrice.Text))

{if (double.Parse(txtActualPrice.Text) <double.Parse(txtSalePrice.Text))

{ return true; } }

return false: }

private void txtQuantity\_TextChanged(object sender, EventArgs e)

{if (!Validations.isValidInteger(txtQuantity.Text))

{errorProvider.SetError(txtQuantity, "Enter only integer value");}

else

{errorProvider.Clear();}}

private void txtActualPrice\_TextChanged(object sender, EventArgs e)

{if (!Validations.isValidNumber(txtActualPrice.Text))

{errorProvider.SetError(txtActualPrice, "Enter Valid Numeric Value... ");}

else

{errorProvider.Clear();}}

private void txtSalePrice\_TextChanged(object sender, EventArgs e)

{if (!Validations.isValidNumber(txtSalePrice.Text))

{errorProvider.SetError(txtSalePrice, "Enter Valid Numeric Value... ");}

else

{errorProvider.Clear();}}

private Product GetProduct()

{ Product product = new Product(txtProductId.Text, txtName.Text, double.Parse(txtActualPrice.Text), double.Parse(txtSalePrice.Text), int.Parse(txtQuantity.Text));

return product; }

private void btnAdd\_Click(object sender, EventArgs e)

{ Product product = GetProduct();

if (categoryIdx == -1)

{List<Product> products = new List<Product> { product };

Category category = new Category(txtCategory.Text, products);

CategoriesDL.addCategory(category);

MessageBox.Show("New Category Added Successfully");}

else if (categoryIdx > -1)

{CategoriesDL.AddProduct(categoryIdx, product);

MessageBox.Show("New Product added successfully");}

dataBinding();

clearTextbox();}

private void btnEdit\_Click(object sender, EventArgs e)

{VisibleButtons(false, false, false, false, false, true, true);

EnableButtons(false, false, false, false, false, false);

btnCancel.Enabled = true;

timer.Enabled = false;

UpdateTimer.Enabled = true;

txtCategory.Enabled = true;

txtProductId.Enabled = true; }

private void btnDelete\_Click(object sender, EventArgs e)

{CategoriesDL.DeleteProduct(categoryIdx,productIdx);

MessageBox.Show("Deleted Successfully");

clearTextbox();}

private void btnCancel\_Click(object sender, EventArgs e)

{VisibleButtons(true,true,true,true,true,false,false);

EnableButtons(false,true,true,true,true,false);

timer.Enabled = true;

UpdateTimer.Enabled= false;

txtCategory.Enabled = false;

txtProductId.Enabled = false;}

private void UpdateTimer\_Tick(object sender, EventArgs e)

{Product product = CategoriesDL.GetProduct(categoryIdx, productIdx);

if(validInputs()&&(txtActualPrice.Text!=product.ActualPrice.ToString()||txtSalePrice.Text!=product.SalePrice.ToString()||txtQuantity.Text!=product.Stock.ToString()))

{btnUpdate.Enabled = true; btnCancel.Enabled = true; }

else

{btnUpdate.Enabled=false; }}

private void clearTextbox()

{txtCategory.Text = null;

txtName.Text = null;

txtProductId.Text= null;

txtQuantity.Text= null;

txtSalePrice.Text= null;

txtActualPrice.Text= null; }

private void btnSave\_Click(object sender, EventArgs e)

{for(int i=0; i < CategoriesDL.Categories.Count;i++)

{MessageBox.Show(CategoriesDL.Categories[i].Category1.ToString());

for(int j=0;j< CategoriesDL.Categories[i].Products.Count;j++)

{ MessageBox.Show(CategoriesDL.Categories[i].Products[j].toString()); } }

CategoriesDL.writeData();

MessageBox.Show(CategoriesDL.Categories[0].Products[0].toString());

MessageBox.Show("Now you won't loss your data!!!"); }

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Category", "Product ID", "Name", "Stock", "Actual Price" , "Sale Price" };

// Add columns using a loop\

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataGrid.Columns.Add(col); } }

private void dataBinding()

{DataGrid.DataSource = null;

DataGrid.Rows.Clear();

foreach (Category category in CategoriesDL.Categories)

{foreach (Product product in category.Products) {DataGrid.Rows.Add(category.Category1,product.Id,product.Name,product.Stock,product.ActualPrice,product.SalePrice); }}

DataGrid.Refresh();}

private void DataTable\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

categoryIdx = DataGrid.CurrentRow.Index;

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{DataGridViewRow clickedRow =DataGrid.Rows[e.RowIndex];

// Retrieve the value of the clicked cell

string category = clickedRow.Cells[0].Value?.ToString();

string ProductID = clickedRow.Cells[1].Value?.ToString();

txtCategory.Text = category;

txtProductId.Text = ProductID;

categoryIdx=CategoriesDL.getCategoryIndex(category);

productIdx = CategoriesDL.getProductIndex(categoryIdx,ProductID);

fillTextBox();}}

private void fillTextBox()

{Product product = CategoriesDL.GetProduct(categoryIdx, productIdx);

txtCategory.Text = CategoriesDL.Categories[categoryIdx].Category1;

txtName.Text = product.Name;

txtProductId.Text = product.Id;

txtQuantity.Text = product.Stock.ToString();

txtSalePrice.Text = product.SalePrice.ToString();

txtActualPrice.Text = product.ActualPrice.ToString();}

private void btnUpdate\_Click(object sender, EventArgs e)

{if (validInputs())

{Product product= new Product(txtProductId.Text,txtName.Text,double.Parse(txtActualPrice.Text), double.Parse(txtSalePrice.Text),int.Parse(txtQuantity.Text));

CategoriesDL.UpdateProduct(categoryIdx,productIdx,product);

MessageBox.Show("Update Successfully");

VisibleButtons(true, true, true, true, true, false, false);

TextBoxClear();

dataBinding();}

timer.Enabled = true;

UpdateTimer.Enabled = false; }

private void TextBoxClear()

{txtCategory.Text = null; txtName.Text=null; txtProductId.Text = null;

txtQuantity.Text=null;

txtSalePrice.Text = null;

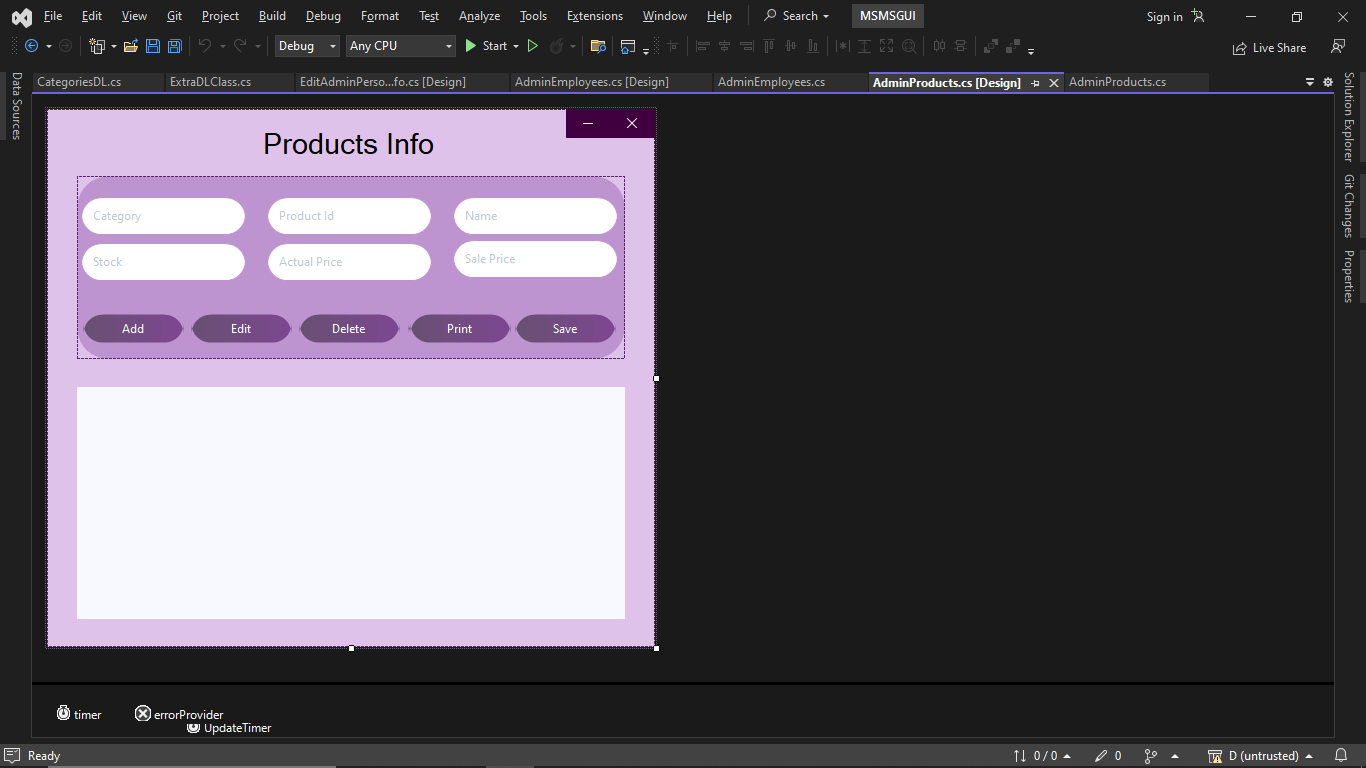
 txtActualPrice.Text = null; }}

Figure 6:Product Details User Control

#### 7.3.2.5 Profit

public partial class Profit : UserControl

{

public Profit()

{

InitializeComponent();

txtProfit.Text=AdminDL.Admin.Profit.ToString()+" RS.";

}

}

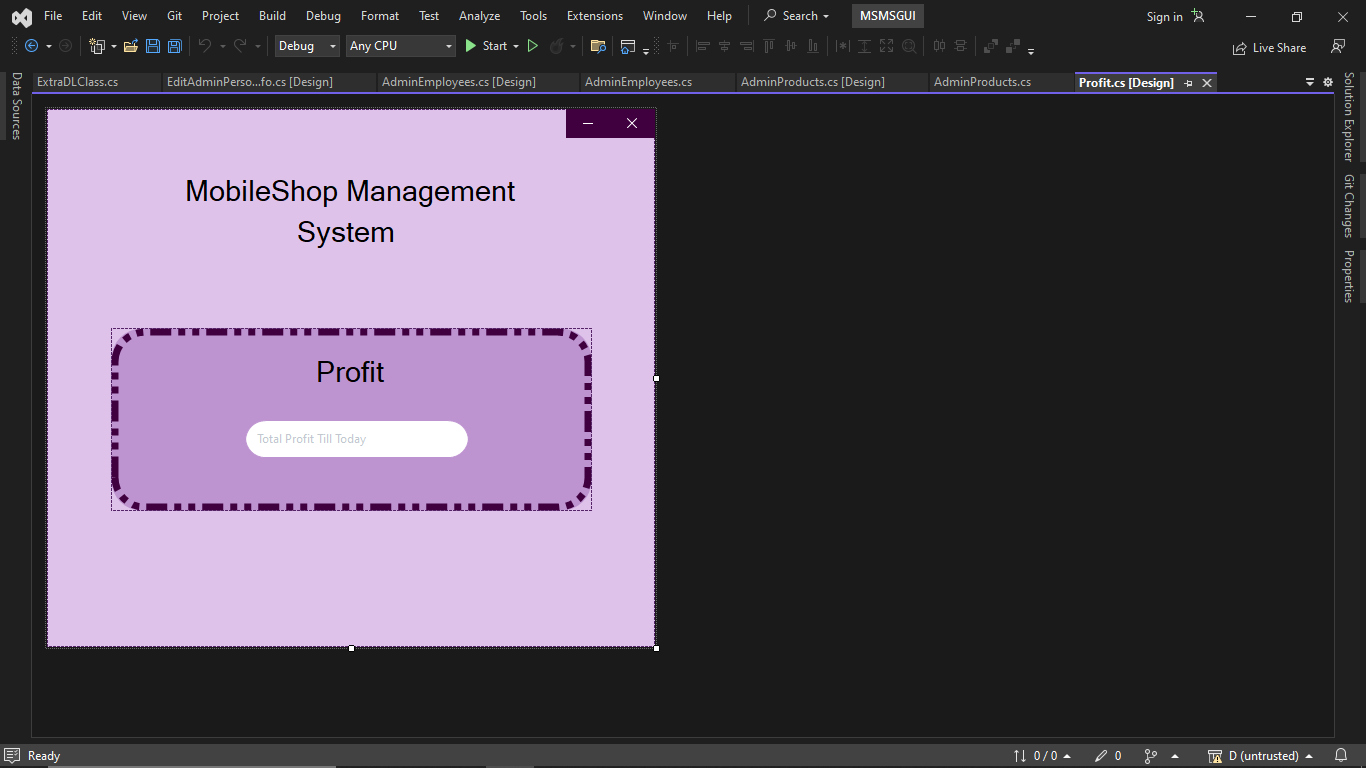


Figure 7:Shop Profit User Control

#### 7.3.2.6 Sold Products

public partial class SoldProducts : UserControl

{

public SoldProducts()

{InitializeComponent();

InitializeDataTable();

dataBinding();}

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Category", "Product ID", "Name", "ActualPrice", "SalePrice", "Profit" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col);}}

private void dataBinding()

{DataTable.DataSource = null;

DataTable.Rows.Clear();

for(int i=0;i<CustomerDL.Customers.Count;i++)

{

for(int j=0;j<CustomerDL.Customers[i].History.Count; j++)

{

Category category = CategoriesDL.GetCategory(CustomerDL.Customers[i].History[j]);

Product product = CategoriesDL.GetProduct(category, CustomerDL.Customers[i].History[j]);

if (category != null && product != null)

{

double profit = product.SalePrice - product.ActualPrice;

DataTable.Rows.Add(category.Category1, product.Id,product.Name, product.ActualPrice.ToString(), product.SalePrice.ToString(), profit.ToString());

DataTable.Refresh();

}

### 7.3.3 Customer Forms

#### 7.3.3.1 Sign In

public partial class CustomerSignIn : UserControl

{

public CustomerSignIn()

{

InitializeComponent();

}

private void linlLblSignUp\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

CustomerSignUp customerSignUp = new CustomerSignUp();

panel.Controls.Clear();

panel.Controls.Add(customerSignUp);

}

private void btnSignIn\_Click(object sender, EventArgs e)

{

BL.Customer customer = new BL.Customer(txtUsername.Text,txtPassword.Text);

int index = CustomerDL.getCustomerIndex(customer);

if (index>-1)

{

ExtraDLClass.Customer = CustomerDL.GetCustomer(index);

Main.Customer admin = new Main.Customer();

Form form = FindForm();

form.Hide();

admin.Show();

}

else if(index==-1)

{MessageBox.Show("Invalid Username or Password");}

}

}

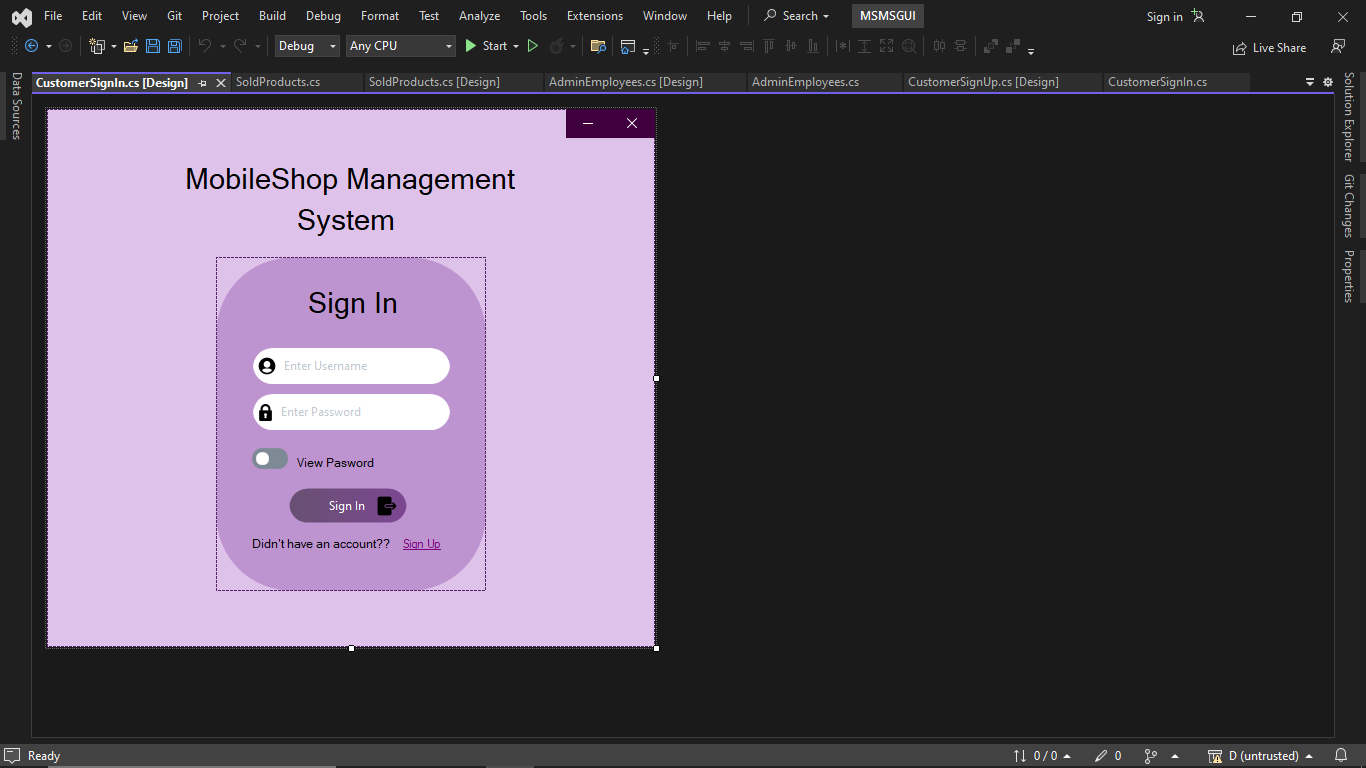


Figure 8:Sign In User Control

#### 7.3.3.2 Sign Up

public partial class CustomerSignUp : UserControl

{

public CustomerSignUp()

{InitializeComponent();}

private void LinkLblSignIn\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{CustomerSignIn customerSignIn = new CustomerSignIn();

Panel.Controls.Clear();

Panel.Controls.Add(customerSignIn);}

private void btnSignUp\_Click(object sender, EventArgs e)

{BL.Customer customer =new BL.Customer(txtUsername.Text,txtPassword.Text,txtPhoneNumber.Text);

CustomerDL.addCustomer(customer);

CustomerDL.writeData();

List<BL.Customer> customers = CustomerDL.Customers;

for (int i = 0; i < customers.Count; i++)

{

if (i == customers.Count - 1)

{

MessageBox.Show(customers[i].toString());

}

else

{

MessageBox.Show(customers[i].toString());

}

}

MessageBox.Show("Sucessfully SignedUp");

CustomerSignIn customerSignIn = new CustomerSignIn();

Panel.Controls.Clear();

Panel.Controls.Add(customerSignIn);

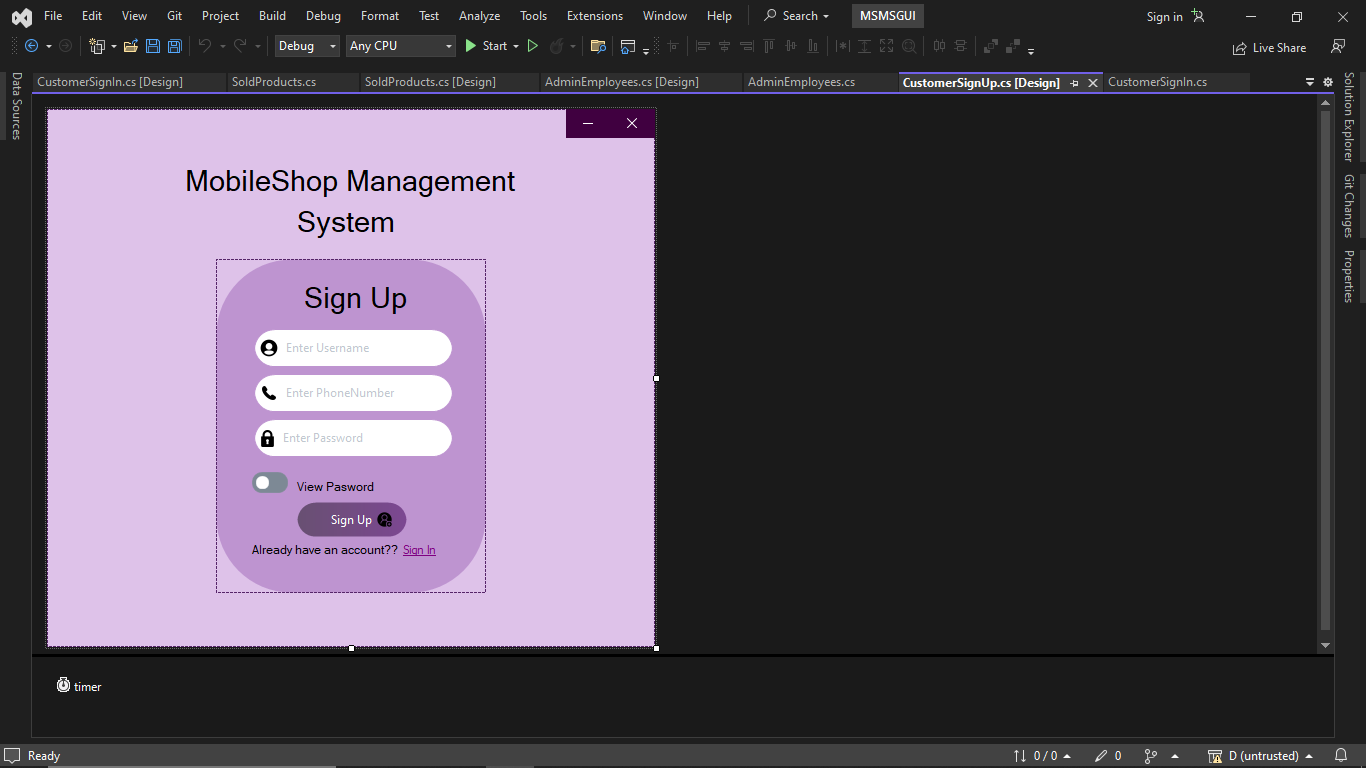
}}

Figure 9: Sign Up User Control

#### 7.3.3.3 Customer Form

public partial class Customer : Form

{

public Customer()

{

InitializeComponent();

}

private void BtnPersonalInfo\_Click(object sender, EventArgs e)

{

CustomerPersonalInfo customerPersonalInfo = new CustomerPersonalInfo();

pnlRight.Controls.Clear();

pnlRight.Controls.Add(customerPersonalInfo);

pnlRight.BringToFront();

}

private void btnLogOut\_Click(object sender, EventArgs e)

{ MainMenu mainMenu = new MainMenu();

this.Hide();

mainMenu.Show();}

private void BtnProducts\_Click(object sender, EventArgs e)

{ CustomerProducts customerProducts = new CustomerProducts();

pnlRight.Controls.Clear();

pnlRight.Controls.Add(customerProducts);

pnlRight.BringToFront();}

private void BtnCart\_Click(object sender, EventArgs e)

{ CustomerCart customerCart = new CustomerCart();

pnlRight.Controls.Clear();

pnlRight.Controls.Add(customerCart);

pnlRight.BringToFront();}

private void BtnBestSelling\_Click(object sender, EventArgs e)

{ BestSellingProducts bestSellingProducts = new BestSellingProducts();

pnlRight.Controls.Clear();

pnlRight.Controls.Add(bestSellingProducts);

pnlRight.BringToFront();}

private void btnHistory\_Click(object sender, EventArgs e)

{ CustomerHistory customerHistory = new CustomerHistory();

pnlRight.Controls.Clear();

pnlRight.Controls.Add(customerHistory);

pnlRight.BringToFront();}}

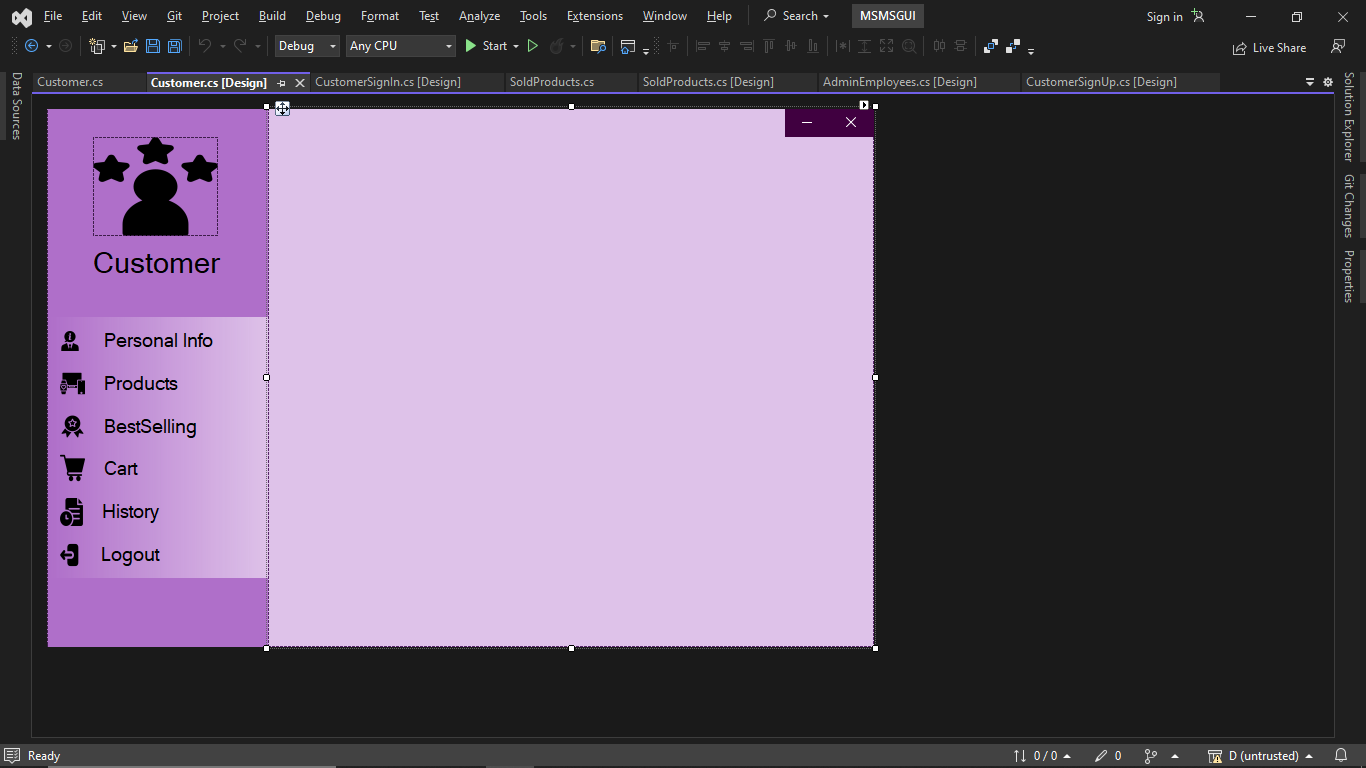


Figure 10:Customer Form

#### 7.3.3.4 Personal Info

public partial class CustomerPersonalInfo : UserControl

{

BL.Customer customer = ExtraDLClass.Customer;

public CustomerPersonalInfo()

{

InitializeComponent();

txtPhoneNumber.Text=customer.PhoneNumber;

txtPassword.Text = customer.Password1;

txtUsername.Text = customer.Name1;

}

private void btnEdit\_Click(object sender, EventArgs e)

{

EditCustomerPersonalInfo editCustomerPersonalInfo = new EditCustomerPersonalInfo();

pnlMain.Controls.Clear();

pnlMain.Controls.Add(editCustomerPersonalInfo);

} }

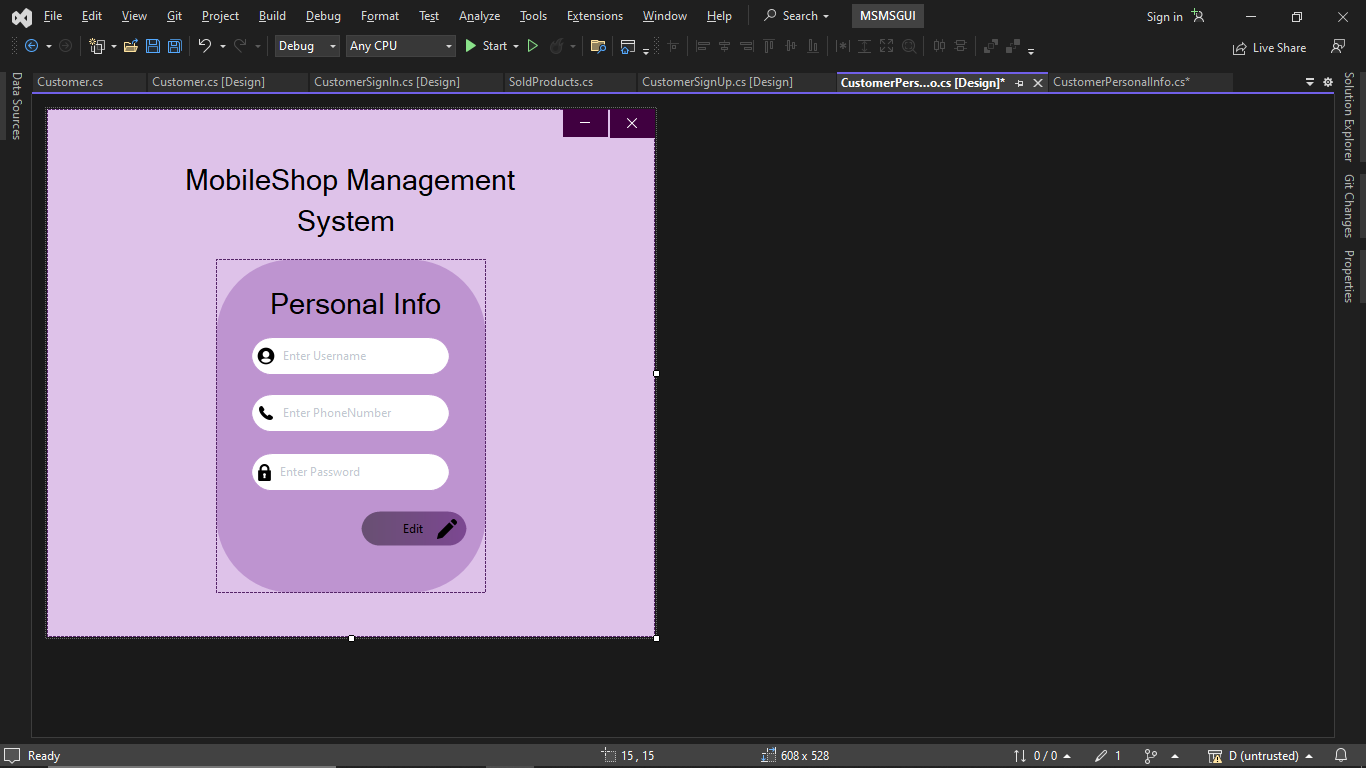


Figure 11:Customer Personal Info User Control

#### 7.3.3.5 Edit Personal Info

public partial class EditCustomerPersonalInfo : UserControl

{

BL.Customer Customer=ExtraDLClass.Customer;

public EditCustomerPersonalInfo()

{ InitializeComponent();

txtUsername.Text = Customer.Name1;

txtPhoneNumber.Text = Customer.PhoneNumber; }

private void btnCancel\_Click(object sender, EventArgs e)

{ CustomerPersonalInfo customerPersonalInfo = new CustomerPersonalInfo();

pnlMain.Controls.Clear();

pnlMain.Controls.Add(customerPersonalInfo);}

private void btnUpdate\_Click(object sender, EventArgs e)

{ BL.Customer customer = ExtraDLClass.Customer;

customer.Password1 = txtPassword.Text;

customer.PhoneNumber = txtPhoneNumber.Text;

CustomerDL.writeData();}

private void timer\_Tick(object sender, EventArgs e)

{ if(Validations.IsValidPassword( txtPassword.Text)&&Validations.isValidNumber(txtPhoneNumber.Text))

{ if(txtPassword.Text!=ExtraDLClass.Customer.Password1||txtPhoneNumber.Text!=Ex traDLClass.Customer.PhoneNumber)

{ btnUpdate.Enabled = true;

return; }

btnUpdate.Enabled = false;} }

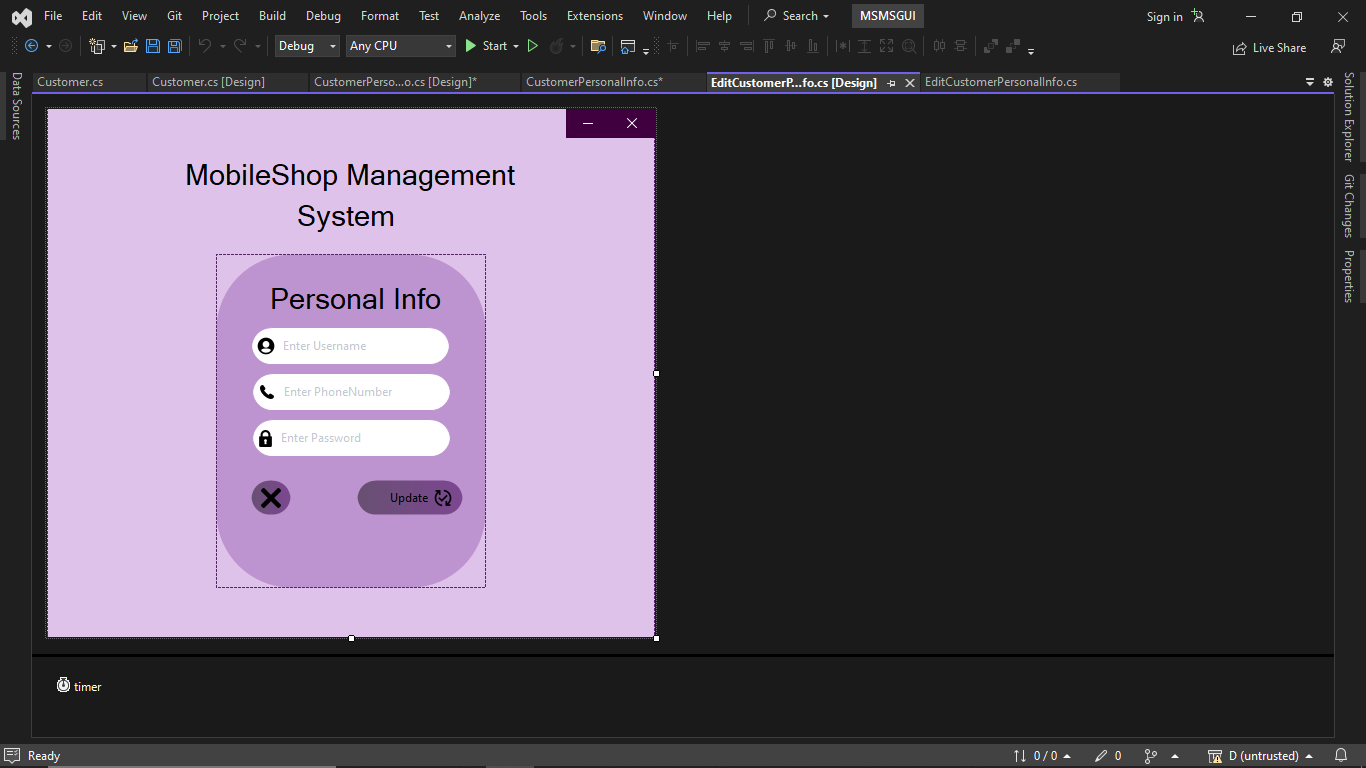


Figure 12: Edit Personal Info User Control

#### 7.3.3.6 Shop Products

public partial class CustomerProducts : UserControl

{ List<Category> categories;

int categoryIdx;

int productIdx;

Product SelectedProduct;

Product cartProduct;

public CustomerProducts()

{ categories = CategoriesDL.Categories;

InitializeComponent();

InitializeComboBox();

InitializeDataTable();

dataBinding(); }

private void InitializeComboBox()

{ foreach(Category c in categories)

{ comboBoxCategories.Items.Add(c.Category1); }

for(int i=0;i<6;i++)

{ ComboBoxQuantity.Items.Add(i); } }

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Category", "Product ID", "Name", "Price" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col); } }

private void dataBinding()

{ DataTable.DataSource = null;

DataTable.Rows.Clear();

foreach (Category category in CategoriesDL.Categories)

{ foreach (Product product in category.Products)

{ DataTable.Rows.Add(category.Category1, product.Id, product.Name,product.SalePrice); } }

DataTable.Refresh(); }

private void DataTable\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{ categoryIdx = DataTable.CurrentRow.Index;

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{ DataGridViewRow clickedRow = DataTable.Rows[e.RowIndex];

// Retrieve the value of the clicked cell

string category = clickedRow.Cells[0].Value?.ToString();

string ProductID = clickedRow.Cells[1].Value?.ToString();

categoryIdx = CategoriesDL.getCategoryIndex(category);

productIdx = CategoriesDL.getProductIndex(categoryIdx, ProductID);

fillTextBox(); } }

private void fillTextBox()

{

SelectedProduct = CategoriesDL.GetProduct(categoryIdx, productIdx);

comboBoxCategories.Text = CategoriesDL.Categories[categoryIdx].Category1;

txtName.Text = SelectedProduct.Name;

txtProductId.Text = SelectedProduct.Id;

ComboBoxQuantity.Text = "1";

txtPrice.Text = SelectedProduct.SalePrice.ToString();

txtTotalPrice.Text = (SelectedProduct.SalePrice\*double.Parse(ComboBoxQuantity.Text)).ToString();

}

private void btnAdd\_Click(object sender, EventArgs e)

{

cartProduct = new Product(SelectedProduct.Id, SelectedProduct.Name, SelectedProduct.SalePrice, int.Parse(ComboBoxQuantity.Text));

int custIdx=CustomerDL.getCustomerIndex(ExtraDLClass.Customer);

if (SelectedProduct.Stock > 1)

{

CustomerDL.Customers[custIdx].addProductinCart(cartProduct);

SelectedProduct.Stock -= int.Parse(ComboBoxQuantity.Text);

AdminDL.Admin.Profit = AdminDL.Admin.Profit + (SelectedProduct.SalePrice-SelectedProduct.ActualPrice);

MessageBox.Show("Added in Cart Successfully!!!");

CustomerDL.writeData(); }

else

{ MessageBox.Show("Error!!!\nNot Enough Stock"); } } }

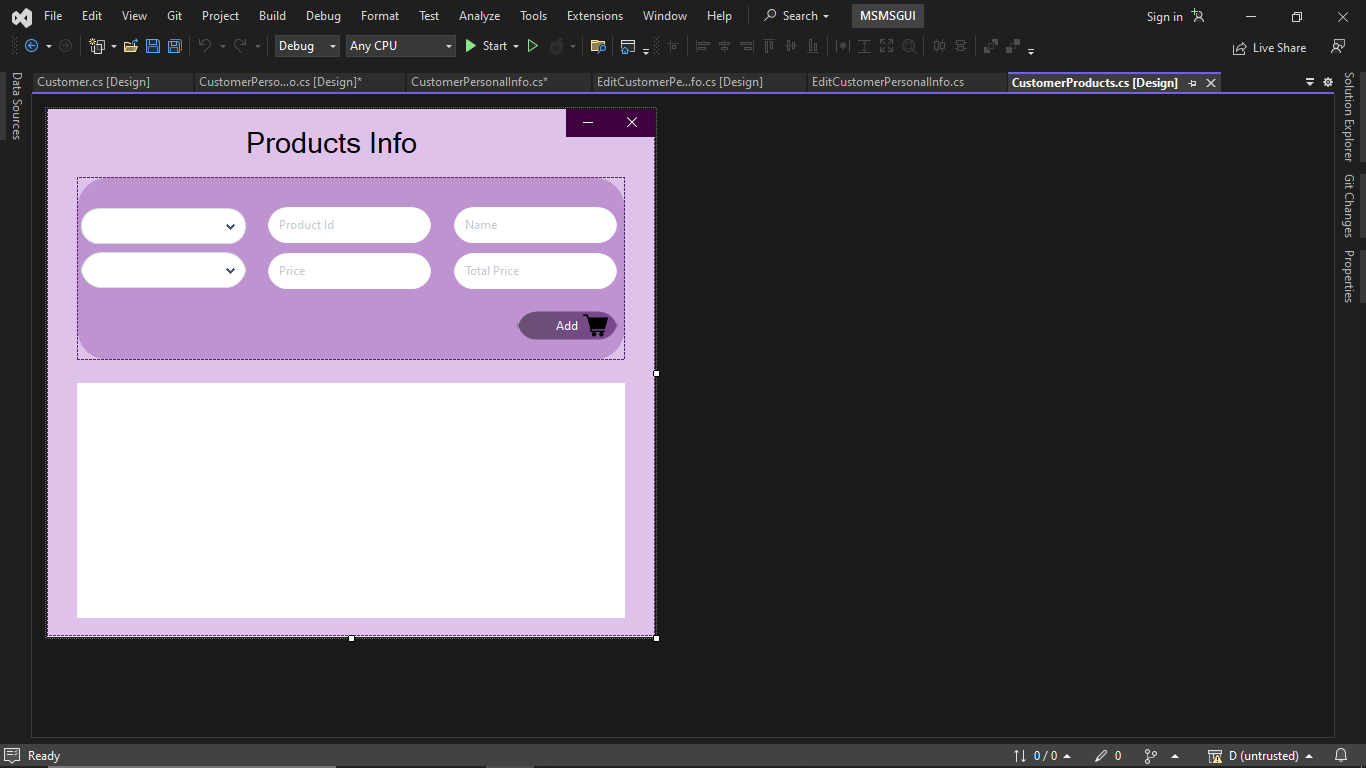


Figure 13:Customer Products User Control

#### 7.3.3.7 Best Selling Products

public partial class BestSellingProducts : UserControl

{ List<Category> categories;

int categoryIdx;

int productIdx;

Product SelectedProduct;

Product cartProduct;

public BestSellingProducts()

{ categories = CategoriesDL.Categories;

InitializeComponent();

InitializeDataTable();

dataBinding(); }

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Category","Product ID", "Name", "Price" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col); } }

private void dataBinding()

{ DataTable.DataSource = null;

DataTable.Rows.Clear();

for (int i = 0; i < CustomerDL.Customers.Count; i++)

{ for (int j = 0; j < CustomerDL.Customers[i].History.Count; j++)

{Category category=CategoriesDL.GetCategory(CustomerDL.Customers[i].History[j]);

Product product = CategoriesDL.GetProduct(category, CustomerDL.Customers[i].History[j]);

if (category != null && product != null)

{ DataTable.Rows.Add(category.Category1, product.Id, product.Name, product.SalePrice.ToString());} }}

DataTable.Refresh();}

private void fillTextBox()

{ SelectedProduct = CategoriesDL.GetProduct(categoryIdx, productIdx);

comboBoxCategories.Text = CategoriesDL.Categories[categoryIdx].Category1;

txtName.Text = SelectedProduct.Name;

txtProductId.Text = SelectedProduct.Id;

ComboBoxQuantity.Text = "1";

txtPrice.Text = SelectedProduct.SalePrice.ToString();

txtTotalPrice.Text = (SelectedProduct.SalePrice \* double.Parse(ComboBoxQuantity.Text)).ToString();}

private void btnAdd\_Click\_1(object sender, EventArgs e)

{ cartProduct = new Product(SelectedProduct.Id, SelectedProduct.Name, SelectedProduct.SalePrice, int.Parse(ComboBoxQuantity.Text));

int custIdx = CustomerDL.getCustomerIndex(ExtraDLClass.Customer);

if (SelectedProduct.Stock > 1){

CustomerDL.Customers[custIdx].addProductinCart(cartProduct);

SelectedProduct.Stock -= int.Parse(ComboBoxQuantity.Text);

AdminDL.Admin.Profit = AdminDL.Admin.Profit + (SelectedProduct.SalePrice - SelectedProduct.ActualPrice);

MessageBox.Show("Added in Cart Successfully!!!");

CustomerDL.writeData(); }

else { MessageBox.Show("Error!!!\nNot Enough Stock");} }

private void DataTable\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{ categoryIdx = DataTable.CurrentRow.Index;

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{ DataGridViewRow clickedRow = DataTable.Rows[e.RowIndex];

// Retrieve the value of the clicked cell

string category = clickedRow.Cells[0].Value?.ToString();

string ProductID = clickedRow.Cells[1].Value?.ToString();

categoryIdx = CategoriesDL.getCategoryIndex(category);

productIdx = CategoriesDL.getProductIndex(categoryIdx, ProductID);

fillTextBox();} } }

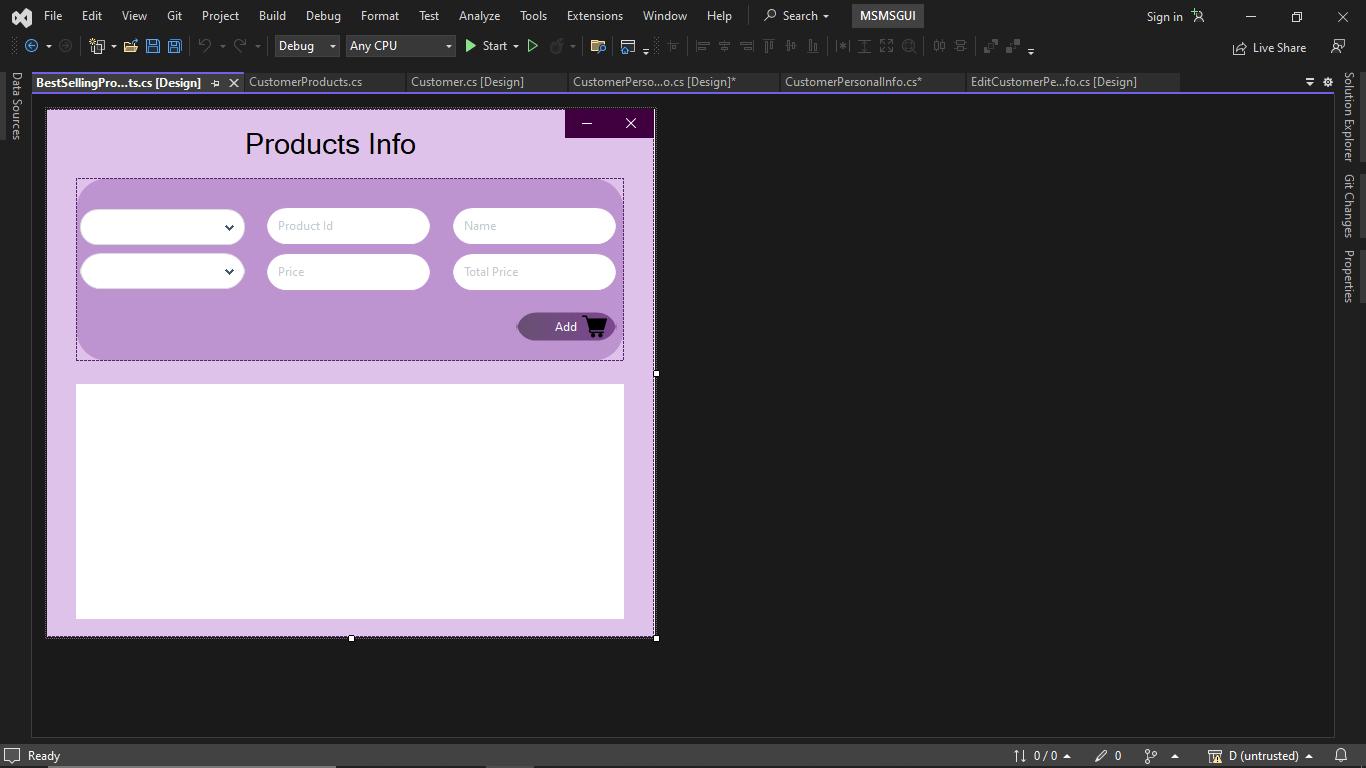


Figure 14: Best Selling Products User Control

#### 7.3.3.8 Cart

public partial class CustomerCart : UserControl

{ double TotalBill = 0;

Product cartProduct;

Product shopProduct;

public CustomerCart()

{ InitializeComponent();

CalculateBill();

InitializeDataTable();

dataBinding(); }

private void timer\_Tick(object sender, EventArgs e)

{ if (txtName.Text != null && txtProductId.Text != null)

{ btnRemove.Visible = true;

btnRemove.Enabled = true;}

else

{ btnRemove.Visible = false;

btnRemove.Enabled = false;} }

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Product ID", "Name", "Price", "Quantity", "TotalPrice" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col);}}

private void dataBinding()

{ DataTable.DataSource = null;

DataTable.Rows.Clear();

foreach (Product product in CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart)

{ double price = product.SalePrice \* product.Stock;

DataTable.Rows.Add(product.Id, product.Name, product.SalePrice.ToString(), product.Stock.ToString(), price.ToString()); }

DataTable.Refresh(); }

private void DataTable\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{

int idx = DataTable.CurrentRow.Index;

MessageBox.Show(idx.ToString());

cartProduct = CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart[idx];

fillTextBox();

}

}

private void CalculateBill()

{

TotalBill = 0;

foreach (Product product in CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart)

{

TotalBill = TotalBill + product.SalePrice; }

txtBill.Text = TotalBill.ToString() + " RS.";

}

private void fillTextBox()

{

txtName.Text = cartProduct.Name;

txtBoxQuantity.Text = cartProduct.Stock.ToString();

txtPrice.Text = cartProduct.SalePrice.ToString();

txtProductId.Text = cartProduct.Id.ToString();

txtTotalPrice.Text = (cartProduct.SalePrice \* cartProduct.Stock).ToString();

CalculateBill();

}

private void btnRemove\_Click(object sender, EventArgs e)

{

int idx = DataTable.CurrentRow.Index;

if (idx < CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart.Count)

{

cartProduct = CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart[idx];

CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].Cart.RemoveAt(idx);

dataBinding();

MessageBox.Show("Product removed from the cart..."); }

CustomerDL.writeData(); }

private void btnConfirmOrder\_Click(object sender, EventArgs e)

{

MessageBox.Show("Thank you for shopping.!!\n Your total bill is "+TotalBill.ToString()+"Rs.");

int customerIdx = ExtraDLClass.getCustomerIdx();

List<Product> History = CustomerDL.Customers[customerIdx].History;

List<Product> CartItems = CustomerDL.Customers[customerIdx].Cart;

foreach (Product cartItem in CartItems)

{

History.Add(cartItem);

}

CustomerDL.Customers[customerIdx].History = History;

CustomerDL.Customers[customerIdx].Cart.Clear();

dataBinding();

CustomerDL.writeData();

}}

#### History

Figure 15: Customer Cart User Control

public partial class CustomerHistory : UserControl

{ public CustomerHistory()

{ InitializeComponent();

InitializeDataTable();

dataBinding(); }

private void InitializeDataTable()

{ // Column headers for the DataTable

string[] columnHeaders = { "Product ID", "Name", "Price", "Quantity", "TotalPrice" };

// Add columns using a loop

foreach (var columnHeader in columnHeaders)

{ DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();

col.HeaderText = columnHeader;

DataTable.Columns.Add(col); } }

private void dataBinding()

{ DataTable.DataSource = null;

DataTable.Rows.Clear();

foreach (Product product in CustomerDL.Customers[ExtraDLClass.getCustomerIdx()].History)

{ double price = product.SalePrice \* product.Stock;

DataTable.Rows.Add(product.Id, product.Name, product.SalePrice.ToString(), product.Stock.ToString(), price.ToString());}

DataTable.Refresh();

}

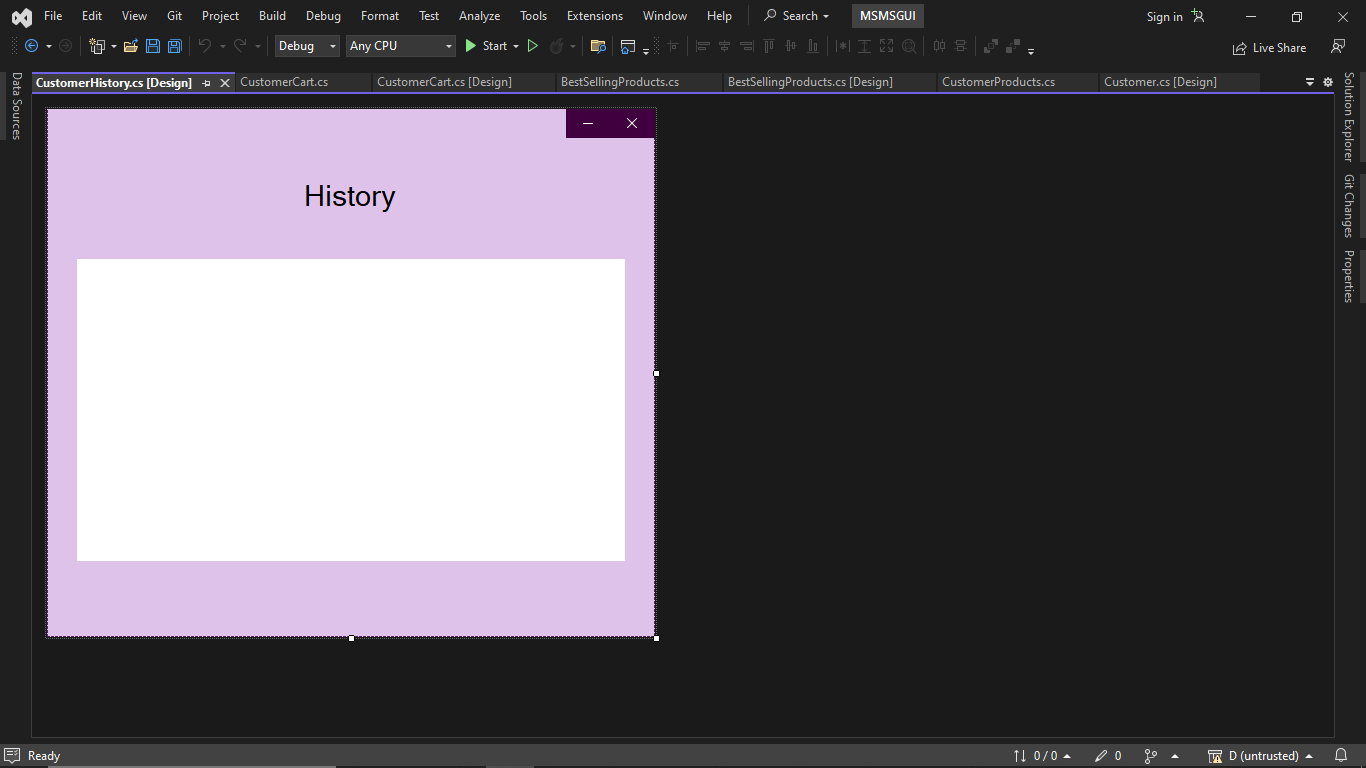
 }

Figure 16: Customer Buying History User Control

# 8. Conclusion

## Summary

* + The Mobile Shop Management System has successfully addressed the challenges faced by a store in managing inventory, sales, customers, employees , history and profit of the shop. Mobile shop management system is a well-established software for the business running as a small shop in the markets. In this project, I tried to achieve every single module of such business.
  + The project aimed to develop a multi-layered application using Business Logic (BL), Data Layer (DL), and User Interface (UI) components. The BL layer encapsulated core functionalities and algorithms, ensuring separation of concerns. The DL layer managed data storage and retrieval, while the UI layer provided a user-friendly interface for interactions. The project emphasized modularity, scalability, and maintainability through its architecture.

## Achievements

* Successful Implementation:

I accomplished the project's primary goal of building a functioning application with a robust architecture, showcasing the capabilities of each layer.

* User-Friendly Interface:

The UI layer provided a seamless and intuitive experience for end-users, contributing to improved user satisfaction and engagement.

* Scalable Solution:

The modular architecture allowed the application to accommodate new features and handle increased data loads without significant modifications to the existing codebase.

* Maintenance and Updates:

The project's architecture and design made it easier to maintain and update the application , reducing downtime and ensuring a smooth user experience during updates.

* Learning and Growth:

Throughout the project, the team members expanded their knowledge of software design, best practices, and collaboration, paving the way for continued growth and improvement in future projects.

## Challenges Faced

* Requirements Gathering and Management:

Difficulty in understanding and defining clear and comprehensive requirements, as well as managing changes to requirements during the project's lifecycle.

* Technical Complexity:

Dealing with intricate technical aspects, such as complex algorithms, integration with external systems, or emerging technologies.

* Bug Fixing and Testing:

Identifying and resolving software bugs and conducting thorough testing to ensure the product's quality and reliability.

* User Acceptance:

Ensuring that the final product meets user expectations and addresses their needs effectively.

* Security and Privacy:

Implementing robust security measures and ensuring data privacy in the application.

* Integration and Compatibility:

Ensuring seamless integration with existing systems and compatibility across different platforms and devices.

## Lessons Learned

* Architecture Matters:

The project underscored the significance of choosing an appropriate architecture for software development. A well-designed architecture with clear separation of responsibilities enables smoother collaboration among team members and facilitates future enhancements.

* Layered Approach:

Adopting a layered approach helped us manage complexity, making it easier to focus on specific aspects of the application. This approach also minimized dependencies and allowed better testing and debugging.

* **Modularity and Flexibility:**

By breaking the application into smaller, modular components, we gained flexibility and reduced the risk of potential code conflicts. This approach made it easier to maintain and extend the project over time.

* Team Collaboration:

Working on a multi-layered project required effective communication and collaboration among team members. We learned the importance of defining clear interfaces between layers and maintaining consistent coding standards for seamless integration.

* **Resource Management:**

Utilizing resources effectively, such as using transparent images for optimal UI design, taught us the significance of optimizing application performance and minimizing resource overhead.