

# **Hotel Management System**



Session: 2022 – 2026

**Submitted by:**

Fasi-ur-Rehman

2022-CS-71

**Submitted To:**

Prof. Dr. Muhammad Awais Hassan

Department of Computer Science

**University of Engineering and Technology  
Lahore Pakistan**

## Contents

1. Introduction:.....	4
<b>1.1 Overview:</b> .....	4
<b>1.2 Objectives:</b> .....	4
2. Intended Functionalities:.....	4
<b>2.1. Manager Functionalities:</b> .....	4
<b>2.2. Cashier Functionalities:</b> .....	5
<b>2.3. Customer Functionalities:</b> .....	5
3. OOP Concepts: .....	6
<b>3.1. Inheritance:</b> .....	6
<b>3.2. Polymorphism:</b> .....	6
<b>3.3. Association:</b> .....	6
<b>3.4. Comparison with Procedural Programming:</b> .....	7
4. Design pattern Implementation:.....	8
<b>4.1. Business Logic (BL):</b> .....	8
<b>4.2. Data Layer (DL):</b> .....	8
<b>4.3. User Interface (UI):</b> .....	8
5. Classes in Hotel Management System: .....	8
<b>5.1. Sign-Up Class:</b> .....	9
<b>5.2. Sign-In Class:</b> .....	10
<b>5.3. Credential Class:</b> .....	10
<b>5.4. Manager Class:</b> .....	15
<b>5.5. Product Class:</b> .....	17
<b>5.6 Worker Class:</b> .....	20
<b>5.7. Cashier Class</b> .....	24
<b>5.8. Menu Class:</b> .....	24
<b>5.9. Order Class:</b> .....	27
<b>5.10. Customer Class:</b> .....	30
6. Wire Frames:.....	34
<b>6.1. Login Page:</b> .....	34
<b>6.2. Sign-Up Page:</b> .....	34
<b>6.3. Manager Page:</b> .....	35
<b>6.4 Manage Product Page:</b> .....	35
<b>6.5 Manage Worker Page:</b> .....	36
<b>6.6 Manage Expenses Sub-Menu:</b> .....	36

<b>6.7 Manage Expenses - Product:</b>	37
<b>6.8 Manage Expenses - Worker:</b>	37
<b>6.9 Manage Expenses - Total:</b>	38
<b>6.10 Manage Reviews:</b>	38
<b>6.11 Cashier Page:</b>	39
<b>6.12 Cashier Page – Add Menu:</b>	39
<b>6.13 Cashier Page – View Menu:</b>	40
<b>6.14 Cashier Page – Update Menu:</b>	40
<b>6.15 Cashier Page – Receive Order:</b>	41
<b>6.16 Cashier Page – View Order:</b>	41
<b>6.17 Cashier Page – Calculate Bill:</b>	42
<b>6.18 Customer Page:</b>	42
<b>6.19 Customer Page – View Menu:</b>	43
<b>6.20 Customer Page – Add Review:</b>	43
<b>6.21 Customer Page – Bill Divider:</b>	44
<b>6.22 Customer Page –Tip Calculator:</b>	44
<b>7. Class, Responsibility and Collaboration Diagram (CRC):</b>	45
<b>8. Conclusion:</b>	45
<b>8.1. Summary:</b>	45
<b>8.2. Challenges and lessons:</b>	46

Youtube Link:

[https://www.youtube.com/watch?v=NPw0A5SHSog&ab\\_channel=FasiTahir](https://www.youtube.com/watch?v=NPw0A5SHSog&ab_channel=FasiTahir)

## 1. Introduction:

### 1.1 Overview:

The project is a Restaurant Management System aimed at streamlining restaurant operations and improving overall efficiency. This application have many features which can make many paper things digital just like you don't have to take order on a paper, you don't have to print paper menus when you can do that with few clicks in application. The advantage of usage of application is that it can decrease the extra paper cost as you can change the things easily on it.

### 1.2 Objectives:

The objectives of the project are to enhance operational efficiency and improve customer service. The system aims to simplify the restaurant management tasks and optimize resource utilization. By use of this application you can enhance operational efficiency by managing the things using this application, you can optimize resource usage by keeping track of all the expenses and by reducing extra paper expenses as by usage of this app you will not have to print the things again and again because of just little change in things, you will not have to keep track of all the records of workers in registers, and you will save time by doing this. This app improve the customer service by providing them some very useful features at one place.

## 2. Intended Functionalities:

The intended functionalities of this can be categorized bases of the user roles: manager, cashier, and customer. Here's an overview of the main functionalities for each role:

### 2.1. Manager Functionalities:

**As a                      I want to perform                      So that I can**

Manager	Manage the products by adding, removing and viewing products that are available. We can set price per unit of product and quantity of product.	Can keep the track about the available products of the hotel along with their quantity and their prices
---------	--	---

Manager	<b>Manage workers:</b> By adding new workers, removing the workers and viewing available workers	Can keep track of workers with their role and time of availability and their salary. We can also remove the any worker if needed by just a click.
Manager	<b>Manage expenses:</b> Of salaries, products and other small expenses. We can also see total expenses	So we can keep a record of expenses digitally and more efficiently. Manager can look at those expenses so that he can optimize usage of Hotel's resources.
Manager	<b>Review Rating:</b> Take a look at the reviews left by the customers.	So that he can improve the things according to the given ratings of users.

## 2.2. Cashier Functionalities:

**As a                      I want to perform                      So that I can**

Cashier	<b>Manage Menu:</b> By adding new items, removing or changing old ones.	Reduce the extra expense of paper on menu. As in paper menu they need the change whole piece just to change few things. We can prevent that by this app.
Cashier	<b>Order Processing:</b> Receive customer orders, view the orders, and remove processed orders.	Improve the flow of orders by making it digital. It can reduce the chances of errors in the orders.
Cashier	<b>Billing:</b> Calculate bills for customers based on their orders.	So we can easily calculate total bill that customer need to pay.

## 2.3. Customer Functionalities:

**As a                      I want to perform                      So that I can**

Customer	<b>Menu Viewing:</b> View the available menu items.	So, Customer can view all the available items along with their prices. So that he can place order.
Customer	<b>Add Review:</b> Submit reviews about different elements of the hotel.	Share his/her feedback so that hotel's management can take action on the cons of hotels.
Customer	<b>Bill Division:</b> Divide the bill amount among multiple customers.	Divide bill among the people. As it is very common that customers divide bill among friend friends they can calculate that by using this app.
Customer	<b>Tip Calculator:</b>	So Customer can calculate the tip he can

	Calculate Tip on the basis of remaining money after paying the bill.	give to the staff of hotel from the remaining money after he/she paid the bill.
--	--	---

### 3. OOP Concepts:

#### 3.1. Inheritance:

Inheritance is one of four pillars of Object-Oriented Programming. It allow classes (Child class/sub-Class) to inherit properties from another class (Parent class/Super Class). In this way child class reuse the code of parent class. This allows the subclass to reuse code and behavior defined in the superclass, promoting code reusability and making the code more organized and maintainable. Inheritance is also known as **is-a** relation

In this project Inheritance is used at two different place:

Three main users of application Manager, Cashier and Customer inherit from a parent class User.

Sign-Up and Sign-In classes which are responsible for authorization of users inherit many functions and attributes such as username, password and role and methods like GetUserName() and SetUserName() and many more.

#### 3.2. Polymorphism:

Polymorphism is another pillar of Object-Oriented Programming. It allow functions to behave differently for different objectives. There are two types of polymorphisms:

1. Static Polymorphism
2. Dynamic Polymorphism

**Static Polymorphism** is the one in which the polymorphism is done while writing the code. In this the two or more functions have same name but they have different parameters. The choice of usage of function is determined while coding.

**Dynamic Polymorphism** is one in which polymorphism is done during runtime. In this we use virtual and override keywords.

In this project static polymorphism is used at various places. Like in validation class there are many functions which execute static polymorphism. IntValiation(string number) and IntValidation(string number, int lowerRange, int upperRange) and same goes in validation for double data type.

#### 3.3. Association:

Association is a fundamental concept in object-oriented programming and design that represents a relationship or a connection between classes. It describes how instances of one class are related to instances of another class. Some of types of association is aggregation, composition and simple association.

In simple association classes interact with each other but they are independent of each other. It is represented by a simple solid line.

**Aggregation** is another type of association in this object of one class is initialized in another class and destroying of one class does not destroy other class. It is has-a relation. It is represented by a line with empty diamond on one side. The empty diamond is on the side of class which contains the object.

**Composition** is also a type of association in this object of one class is created in other and so destroying of one class destroys other.

There are association between many classes in this project. Such as:

Manager's class has association with product class, worker class and attendance class.

Cashier class has composition with menu class as cashier class creates the object of menu and cashier class has aggregation.

### 3.4. Comparison with Procedural Programming:

In traditional procedural programming approaches, code is structured around procedures or functions that operate on data. The focus is on step-by-step execution of instructions. In contrast, object-oriented programming (OOP) brings a higher level of abstraction and encapsulates data and behavior within objects.

Advantages of OOP over procedural programming:

**1. Modularity and Code Reusability:** In OOP we work with modular system and it increase the reusability of code at various places.

**2. Encapsulation and Information Hiding:** OOP uses the concept of encapsulation, so by using that we can hide any data we want from user or we can give access to a certain data only to certain functions or users. This thing was absent in procedural programming. So, in this way OOP gives more security and privacy.

**3. Inheritance and Code Extension:** Inheritance allows classes to inherit properties and behaviors from parent classes. This enables code extension and reuse, reducing redundant code and promoting a more efficient development process.

**4. Polymorphism and Flexibility:** Polymorphism allows objects to exhibit different behaviors based on the context in which they are used. This promotes code flexibility, adaptability, and the ability to handle diverse situations without modifying the core implementation.

**5. Easier to update and Find Bugs:** In OOP we use different design models in this way we can easily find any functionality without scrolling much as we do in procedural programming. So we can easily find Bugs or we can update any function without changing in many places because of OOP.

## **4. Design pattern Implementation:**

### **4.1. Business Logic (BL):**

- The main functionality of the whole application is in BL layer.
- To ensure modularity in business application the storing or reading of data from file is kept separate from this layer.
- All functions related to interface are kept separate from this layer.
- This layer has Validations which insure that only correct data move from UI to DL.

If we keep it separate from DL and UI we insure code reusability. Like if we need to move from CLI to GUI we will only change UI if BL is kept separate and same goes with change of data base in DL.

### **4.2. Data Layer (DL):**

- The DL layer handles data storage, retrieval, and interactions with databases or other data sources.
- It handles all the functions related to reading or writing in data storage which in this application is .txt file.
- It also contains the list of the classes in it. We store data in to those lists and from those list to file and vice versa.

### **4.3. User Interface (UI):**

- The UI layer is responsible for presenting information to users and handling user interactions.
- In this project UI layer have windows form and user controls.
- It is responsible for interaction between user and the BL layer. As it takes input from user and from there it send it to BL.
- It shows the output on the screen.

## **5. Classes in Hotel Management System:**



### 5.1. Sign-Up Class:

**Responsibility:**

The Sign-Up class handles the functionality related to user sign-up. It includes a method named Log-Up which takes user credentials (username, password, and role) as input and creates a new user account if user name and password meet the **requirements** like:

- Password should have at least 5 characters.
- It should have at least one character.
- It should have at least one digit.
- Username should be unique
- Username should not have whitespace or special characters or numbers in it

It returns a Boolean indicating the success or failure of the sign-up process.

**Code:**

```
using HotelManagementSystem.DL;

namespace HotelManagementSystem.BL
{
    class SignUp : Credentials
    {
        public bool LogUp(string userName, string password, string role)
        {
            Credentials c = base.IsExist(userName, password);
            if (c == null)
            {
                c = new Credentials();
                if (!c.SetUserName(userName) || !c.SetPassword(password) ||
                    !c.SetRole(role))
                {
                    return false;
                }
                else
                {
                    c.SetUserName(userName);
                    c.SetPassword(password);
                    c.SetRole(role);
                    CredentialsDL.AddToList(c);
                    return true;
                }
            }
        }
    }
}
```

```
        }  
        else  
        {  
            return false;  
        }  
    }  
}
```

## 5.2. Sign-In Class:

### Responsibility:

The Sign-In class manages the functionality for user login. It includes a method named Login which takes user credentials (username and password) as input and verifies them against the stored credentials. If the login is successful, it returns the user's role.

### Code:

```
class SignIn:Credentials  
{  
    public string Login(string userName, string password)  
    {  
        Credentials c = base.IsExist(userName, password);  
  
        if (c != null)  
        {  
            return c.GetRole();  
        }  
        else  
        {  
            return null;  
        }  
    }  
}
```

## 5.3. Credential Class:

### Responsibility:

#### BL Class:

The responsibility of credential BL class is to handle the credentials like Username, password and role. It has username, password and role validations. Which insure the correct storage of data. It has a function is Exist which check

whether the user is present or not, it also have getter and setter for username, password and role as they are protected

**DL Class:**

DL of credential class hold the list of credentials it store the credential to the file and it also read from file and store the credential in the list.

**Credential BL Code:**

```
class Credentials
{
    protected string userName;
    protected string password;
    protected string role;
    //Constructors
    public Credentials()
    {

    }

    public Credentials(string userName, string password, string role)
    {
        this.userName = userName;
        this.password = password;
        this.role = role;
    }

    //Setters
    public bool SetUserName(string userName)
    {
        bool flag = UserNameValidation(userName);
        if (flag)
        {
            this.userName = userName;
        }

        return flag;
    }

    public bool SetPassword(string password)
    {
        bool flag = PasswordValidation(password);
        if (flag)
        {
            this.password = password;
        }
    }
}
```

```

    }
    return flag;

}

public bool SetRole(string role)
{
    bool flag = RoleValidation(role);
    if (flag)
    {
        this.role = role;
    }
    return flag;
}

public string GetRole()
{
    return role;
}

//Getters
public string GetUserName()
{
    return userName;
}

public string GetPassword()
{
    return password;
}

//.....Other Methods.....
public Credentials IsExist(string userName, string password)
{
    foreach (Credentials c in CredentialsDL.credentials)
    {
        if (c.userName == userName && c.password == password)
        {
            return c;
        }
    }
    return null;
}

//Validations
public static bool UserNameValidation(string userName)
{
    bool flag = true;

```

```

        foreach (char i in userName)
        {
            if (!Char.IsLetter(i) || Char.IsSymbol(i))
            {

                flag = false;
                break;
            }
        }
        if (userName == null || userName == "")
        {
            flag = false;
        }
        return flag;
    }
    //Validations for password
    public static bool PasswordValidation(string password)
    {
        bool hasSymbol = false;
        bool hasLetter = false;
        bool hasDigit = false;

        foreach (char c in password)
        {
            if (!Char.IsLetterOrDigit(c))
            {
                hasSymbol = true;
            }
            else if (Char.IsLetter(c))
            {
                hasLetter = true;
            }
            else if (Char.IsDigit(c))
            {
                hasDigit = true;
            }
        }
        return (hasDigit && hasLetter && hasSymbol && password.Length > 5);
    }

    public static bool RoleValidation(string role)
    {
        string lowerCaseRole = role.ToLower();
        if(lowerCaseRole == "manager" || lowerCaseRole == "cashier" ||
lowerCaseRole == "customer")
        {
            return true;
        }
    }

```

---

```
    }  
    else  
    {  
        return false;  
    }  
}
```

**Credential DL Code:**

```
class CredentialsDL  
{  
    public static List<Credentials> credentials = new List<Credentials>();  
  
    public static void AddToList(Credentials c)  
    {  
        credentials.Add(c);  
    }  
  
    public static void CredentialsFile()  
    {  
        string path = "credentials.txt";  
  
        StreamWriter myFile = new StreamWriter(path, true);  
  
        Credentials info;  
        info = credentials[credentials.Count - 1];  
        myFile.WriteLine(info.GetUserName() + "," + info.GetPassword() + "," +  
info.GetRole());  
  
        myFile.Flush();  
        myFile.Close();  
    }  
  
    public static void CredentialsFileRead()  
    {  
        string path = "credentials.txt";  
  
        if (File.Exists(path))  
        {  
            string record;  
            StreamReader myFile = new StreamReader(path);  
            while (!myFile.EndOfStream)  
            {  
                record = myFile.ReadLine();  
                string[] seperatedData = record.Split(',');  
            }  
        }  
    }  
}
```

```

        string userName = seperatedData[0];
        string password = seperatedData[1];
        string role = seperatedData[2];
        Credentials c = new Credentials(userName, password, role);
        AddToList(c);
    }
    myFile.Close();
}
}
}

```

## 5.4. Manager Class:

### Responsibility:

The Manager class handles the manager-specific functionalities. It handle the management of products working along with product class. It also manages the expenses, it add the other expenses and it take the data from product class and worker class to get their expenses. Then it return the total expenses.

### Code:

```

class Manager : Users
{
    public void AddProduct(Product p)
    {
        if(p!=null)
        {
            ProductDL.AddToList(p);
            ProductDL.ProductFile(); // Save the products to a file or perform other
operations
        }
        else
        {
        }
    }

    public int GetProductCount()
    {
        return ProductDL.products.Count;
    }

    public string GetProductNameAtIndex(int index)
    {
        return ProductDL.products[index].GetName();
    }
}

```

```
public int GetProductQuantityAtIndex(int index)
{
    return ProductDL.products[index].GetQuantity();
}

public double GetProductPriceAtIndex(int index)
{
    return ProductDL.products[index].GetPricePerUnit();
}

public static double ManageExpensesWorker()
{
    double totalSalaries = 0;
    foreach(Worker w in WorkerDL.workers)
    {
        totalSalaries += w.GetSalary();
    }
    return totalSalaries;
}

public static double ManageExpensesProduct()
{
    double totalProduct = 0;
    foreach(Product p in ProductDL.products)
    {
        totalProduct += p.GetPricePerUnit() * p.GetQuantity();
    }

    return totalProduct;
}

public static double TotalExpenses(double salary, double product, double
otherExpenses)
{
    double totalExpenses = salary + product + otherExpenses;
    return totalExpenses;
}
}
```



## 5.5. Product Class:

### BL Layer:

#### Responsibility:

This class control the functions and attributes related to products. It has attributes of the product name, product price and quantity. It has getter setter for these attributes.

#### Code:

```
private string name;
private int quantity;
private double pricePerUnit;

public string Name{ get { return this.name; } set { this.name = value; } }
public int Quantity{ get { return this.quantity; } set { this.quantity = value; } }
public double PricePerUnit{ get { return this.pricePerUnit; } set { this.pricePerUnit
= value; } }
public Product(string name, int quantity,double pricePerUnit)
{
    this.name = name;
    this.quantity = quantity;
    this.pricePerUnit = pricePerUnit;
}
public void SetName(string name)
{
    if (Validation.NameValidation(name))
    {
        this.name = name;
    }
}

public string GetName()
{
    return this.name;
}

public void SetQuantity(int quantity)
{
    this.quantity = quantity;
}

public int GetQuantity()
{
    return this.quantity;
}
```

```

    }

    public void SetPricePerUnit(double pricePerUnit)
    {
        this.pricePerUnit = pricePerUnit;
    }

    public double GetPricePerUnit()
    {
        return this.pricePerUnit;
    }
}

```

**DL Layer:****Responsibility:**

It control the functions related with the list of product. It store data into the file and read data from it. It search in the list for the product. It also remove and add products into the list.

**Code:**

```

class ProductDL
{
    public static List<Product> products = new List<Product>();

    //Search from object
    public static bool isProductExist(Product p)
    {
        foreach(Product product in products)
        {
            if(p.Name.ToLower() == product.Name.ToLower())
            {
                return true;
            }
        }
        return false;
    }

    public static double ProductExpenses()
    {
        double sum = 0;
        foreach (Product p in ProductDL.products)
        {
            sum += p.PricePerUnit * p.Quantity;
        }
    }
}

```

```
        }
        return sum;
    }

    public static void AddToList(Product p)
    {
        if (p != null)
        {
            products.Add(p);
        }
    }

    public static bool RemoveProduct(string name)
    {
        foreach (Product p in products)
        {
            if (name.ToLower() == p.GetName().ToLower())
            {
                products.Remove(p);
                return true;
            }
        }
        return false;
    }

    public static void ProductFile()
    {
        string path = "productData.txt";
        StreamWriter myFile = new StreamWriter(path, true);
        myFile.WriteLine(products[products.Count - 1].GetName() + "," +
products[products.Count - 1].GetQuantity() + "," + products[products.Count-
1].GetPricePerUnit());
        myFile.Flush();
        myFile.Close();
    }

    public static void ProductFileRead()
    {
        string path = "productData.txt";
        StreamReader myFile = new StreamReader(path, true);

        string line;
        while (!myFile.EndOfStream)
        {
            line = myFile.ReadLine();
            string[] data = line.Split(',');
            string name = data[0];
```

```

        int quantity = int.Parse(data[1]);
        double pricePerUnit = double.Parse(data[2]);

        Product p = new Product(name, quantity, pricePerUnit);

        AddToList(p);
    }
    myFile.Close();
}

public static void UpdateProductFile()
{
    string path = "productData.txt";
    StreamWriter myFile = new StreamWriter(path);

    for (int i = 0; i < products.Count; i++)
    {
        myFile.WriteLine(products[i].GetName() + "," + products[i].GetQuantity() + ","
+ products[i].GetPricePerUnit());
    }
    myFile.Flush();
    myFile.Close();
}
}

```

## 5.6 Worker Class:

### BL Layer:

#### Responsibility:

The responsibility of worker class is to manage the worker it have the of worker's name, worker's salary, worker's total working hours and worker's role. It have getter and setter for these attributes.

#### Code:

```

class Worker
{
    private string name;
    private string role;
    private double salary;
    private double workingHours;

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

```

```
public string Role { get { return role; } set { role = value; } }

public double Salary { get { return salary; } set { salary = value; } }

public double Hours { get { return workingHours; } set { workingHours =
value; } }

public Worker(string name, string role, double salary, double workingHours)
{
    this.name = name;
    this.role = role;
    this.salary = salary;
    this.workingHours = workingHours;
}

public string GetName()
{
    return this.name;
}

public string GetRole()
{
    return this.role;
}

public double GetSalary()
{
    return this.salary;
}

public double GetWorkingHours()
{
    return this.workingHours;
}
}
```

**DL Layer:****Responsibility:**

It control the functions related with the list of workers. It add and remove worker from the list. It read data from file and store data into the file. It also calculate total salaries of the workers. It have function to search for the worker.

**Code:**

```

class WorkerDL
{
    public static List<Worker> workers = new List<Worker>();

    public static double WorkerExpenses()
    {
        double sum = 0;
        foreach (Worker w in workers)
        {
            sum += w.Salary;
        }
        return sum;
    }

    public static void AddToList(Worker w)
    {
        workers.Add(w);
    }

    public static bool IsWorkerExist(string name, string role)
    {
        foreach (Worker w in workers)
        {
            if(w.GetName().ToLower()==name.ToLower() &&
            w.GetRole().ToLower() == role.ToLower())
            {
                return true;
            }
        }
        return false;
    }

    public static bool RemoveWorker(string name, string role)
    {
        foreach (Worker w in workers)
        {
            if (w.GetName().ToLower() == name.ToLower() &&
            w.GetRole().ToLower() == role.ToLower())
            {
                workers.Remove(w);
                return true;
            }
        }
        return false;
    }

    public static void WorkersFile()

```

---

```

        {
            StreamWriter myFile = new StreamWriter("workersData.txt", true);

            myFile.WriteLine(workers[workers.Count - 1].GetName() + "," +
workers[workers.Count - 1].GetRole() + "," + workers[workers.Count -
1].GetSalary() + "," + workers[workers.Count - 1].GetWorkingHours());
            myFile.Flush();
            myFile.Close();
        }

public static void WorkersFileRead()
{
    StreamReader myFile = new StreamReader("workersData.txt");

    string line;
    while (!myFile.EndOfStream)
    {
        line = myFile.ReadLine();
        string[] data = line.Split(',');

        string name = data[0];
        string role = data[1];
        double salary = double.Parse(data[2]);
        double workingHours = double.Parse(data[3]);
        Worker w = new Worker(name, role, salary, workingHours);
        AddToList(w);
    }
    myFile.Close();
}

public static void UpdateWorkerData()
{
    StreamWriter workersData = new StreamWriter("workersData.txt");
    foreach(Worker w in workers)
    {
        workersData.WriteLine(w.GetName() + "," + w.GetRole() + "," +
w.GetSalary() + "," + w.GetWorkingHours());
    }
    workersData.Flush();
    workersData.Close();
}
}

```

## 5.7. Cashier Class

### Responsibility:

The Cashier class handles the cashier-specific functionalities. It creates an object of menu class. So it adds menu in this app. Other than that it calculates the bill of the customers.

### Code:

```
class Cashier : Users
{
    Menu menu;

    public void AddMenu(string name, string price)
    {
        this.menu = new Menu(name, price);
        MenuDL.AddToList(menu);
        MenuDL.MenuFile();
    }

    public static float BillCalculator(string billItemName , int quantityBill)
    {
        float bill = 0.0F;

        foreach (Menu m in MenuDL.menu)
        {
            if (m.GetName().ToLower() == billItemName.ToLower() )
            {
                bill = quantityBill * int.Parse(m.GetPrice());
                break;
            }
        }

        return bill;
    }
}
```

## 5.8. Menu Class:

### Responsibility:

It has attributes and methods related to menu. It has item name and item price. It



have getter setter for these. It have validation functions for price validation and name validation. It also have a function which check whether the item is already present in list or not. It also have function which can change the name of the item in the menu.

**Code:**

```
class Menu
{
    private string name;
    private string price;

    public string Name { get { return name.ToUpper(); } set { name = value; } }
    public string Price { get { return price; } set { price = value; } }

    public Menu(string name, string price)
    {
        this.name = name;
        this.price = price;
    }

    public bool SetName(string name)
    {
        bool flag = NameValidation(name);
        if (flag)
        {
            this.name = name;
        }
        return flag;
    }

    public bool SetPrice(string price)
    {
        bool flag = PriceValidation(price);
        if (flag)
        {
            this.price = price;
        }
        return flag;
    }

    public string GetName()
    {
        return this.name;
    }

    public string GetPrice()
    {
        return this.price;
    }
}
```

---

```
public static bool IsPresent(string name)
{
    foreach (Menu n in MenuDL.menu)
    {
        if (n.GetName().ToLower() == name.ToLower())
        {
            return true;
        }
    }
    return false;
}
```

```
public static bool NameValidation(string name)
{
    foreach (char c in name)
    {
        if (!Char.IsLetter(c) && !Char.IsWhiteSpace(c))
        {
            return false;
        }
    }
    return true;
}
```

```
public static bool PriceValidation(string price)
{
    bool flag = true;
    foreach (char c in price)
    {
        if (!Char.IsDigit(c))
        {
            flag = false;
            break;
        }
    }
    return flag;
}
```

```
public static Menu IsItemExist(string name)
{
    foreach (Menu m in MenuDL.menu)
    {
        if (m.GetName().ToLower() == name.ToLower())
        {

```

```

        return m;
    }
}
return null;
}

public static int ChangeName(Menu menu)
{
    int count = 0;
    foreach (Menu m in MenuDL.menu)
    {
        if (m.Name.ToUpper() == menu.Name.ToUpper())
        {
            m.Price = menu.Price;
            break;
        }
        count++;
    }
    MenuDL.MenuDataUpdate();
    return count;
}
}

```

## 5.9. Order Class:

### BL Layer:

#### Responsibility:

It contain the attributes related to the order. It has order name, order item name, order item quantity. It have getter setter for these attributes.

#### Code:

```

class Order
{
    private string name;
    private string orderedItem;
    private int orderQuantity;

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

    public string OrderedItem

```

```
{
    get { return orderedItem; }
    set { orderedItem = value; }
}

public int OrderQuantity
{
    get { return orderQuantity; }
    set { orderQuantity = value; }
}

public Order(string name)
{
    this.name = name;
}

public Order(string name, string orderName, int quantity)
{
    this.name = name;
    this.orderedItem = orderName;
    this.orderQuantity = quantity;
}

public bool SetName(string name)
{
    bool flag = true;
    foreach (char c in name)
    {
        if (!Char.IsLetter(c))
        {
            flag = false;
            break;
        }
    }
    if (flag)
    {
        this.name = name;
        return flag;
    }
    else
    {
        return flag;
    }
}

public string GetName()
{
    return this.name;
}
```

---

```
}
```

## **DL Layer:**

### **Responsibility:**

It have all the functions related to the list of orders as list of orders is present in this layer. This layer or Order Class have function to remove the order from the list.

### **Code:**

```
class OrderDL
{
    public static List<Order> orders = new List<Order>();

    public static void AddToList(Order o)
    {
        orders.Add(o);
    }

    public static bool RemoveOrder(string name)
    {
        bool flag = false;

        for (int i = 0; i < orders.Count; i++)
        {
            if (name == orders[i].GetName())
            {
                flag = true;

                orders.RemoveAt(i);
                break;
            }
        }

        return flag;
    }
}
```

## 5.10. Customer Class:

### BL Layer:

#### Responsibility:

The Customer class manages the customer-related operations. It includes attributes of rating like service rating, environment rating and food rating. It have getter setter for these attributes. It also have methods like Bill Divider, and Tip Calculator to handle the calculation of these functions.

#### Code:

```
class Customer:Users
{
    private string serviceRating;
    private string foodRating;
    private string environmentRating;

    public string Service { get { return serviceRating; } set { serviceRating
= value; } }
    public string Food { get { return foodRating; } set { foodRating =
value; } }
    public string Environment { get { return environmentRating; } set {
environmentRating = value; } }

    public Customer(string serviceRating, string foodRating, string
environmentRating)
    {
        this.serviceRating = serviceRating;
        this.foodRating = foodRating;
        this.environmentRating = environmentRating;
    }

    public string GetServiceRating()
    {
        return this.serviceRating;
    }

    public string GetFoodRating()
    {
        return this.foodRating;
    }
}
```

```
    }

    public string GetEnvironmentRating()
    {
        return this.environmentRating;
    }

    public static float BillDivider(float totalbill, int peoplecount)
    {
        float divided_bill;
        divided_bill = totalbill / peoplecount;
        return divided_bill;
    }

    public static float TipCalculator( float totalBill, float totalMoney, float
percentTip)
    {
        float remaining = totalMoney - totalBill;

        if (remaining >= 0 && percentTip <= 100)
        {
            float remainingAfter = remaining - ((percentTip / 100) *
remaining);
            return remaining - remainingAfter;
        }

        return 0;
    }

    public static bool DeleteReview(int number)
    {
        if(number > 0 && number <= CustomerDL.customers.Count)
        {
            CustomerDL.customers.RemoveAt(number - 1);
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

---

**DL Layer:****Responsibility:**

It controls all the data storing and list related operations it have list of reviews that customer entered. It remove reviews from list and add them to list. It store reviews into the files and read those reviews from the files.

**Code:**

```
class CustomerDL
{
    public static List<Customer> customers = new List<Customer>();

    public static void AddToList(Customer c)
    {
        customers.Add(c);
    }

    public static bool DeleteReview(int index)
    {
        if (index < customers.Count)
        {
            customers.RemoveAt(index);
            return true;
        }
        else
        {
            return false;
        }
    }

    public static void ReviewFile()
    {
        string path = "Reviews.txt";

        StreamWriter myFile = new StreamWriter(path, true);
        Customer info;
        info = customers[customers.Count - 1];
        myFile.WriteLine(info.GetServiceRating() + "," +
info.GetFoodRating() + "," + info.GetEnvironmentRating());

        myFile.Flush();
        myFile.Close();
    }
    public static void ReviewDatasRead()
```

---



```

        {
            string path = "Reviews.txt";
            string record;
            if (File.Exists(path))
            {
                StreamReader myFile = new StreamReader(path);
                while (!myFile.EndOfStream)
                {
                    record = myFile.ReadLine();
                    string[] splitedData = record.Split(',');

                    string serviceRating = splitedData[0].Trim();
                    string foodRating = splitedData[1].Trim();
                    string environmentRating = splitedData[2].Trim();

                    Customer c = new Customer(serviceRating, foodRating,
environmentRating);
                    AddToList(c);
                }

                myFile.Close();
            }
        }

        public static void UpdateReviewFile()
        {
            StreamWriter reviewData = new StreamWriter("Reviews.txt");

            foreach (Customer c in customers)
            {
                reviewData.WriteLine(c.GetServiceRating() + "," +
c.GetFoodRating() + "," + c.GetEnvironmentRating());
            }
            reviewData.Flush();
            reviewData.Close();
        }
    }

```

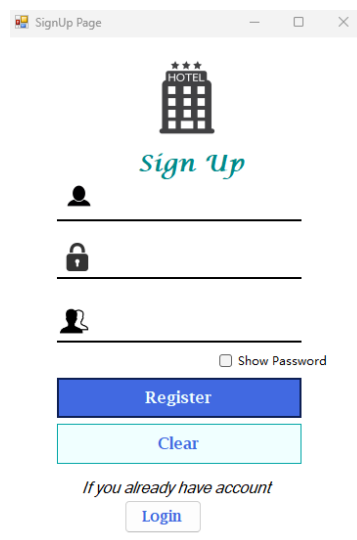
## 6. Wire Frames:

### 6.1. Login Page:



A wireframe of a login page titled "Login Page". At the top center is a hotel icon with three stars and the word "HOTEL" above it. Below the icon is the word "LOGIN" in a blue, italicized font. There are two input fields: the first has a person icon and the second has a padlock icon. To the right of the password field is a checkbox labeled "Show Password". Below the input fields are two buttons: a blue "Login" button and a light blue "Clear" button. At the bottom, there is a link that says "If you don't have account" followed by a "Sign Up" button.

### 6.2. Sign-Up Page:



A wireframe of a sign-up page titled "SignUp Page". At the top center is a hotel icon with three stars and the word "HOTEL" above it. Below the icon is the word "Sign Up" in a green, italicized font. There are three input fields: the first has a person icon, the second has a padlock icon, and the third has a person icon. To the right of the second field is a checkbox labeled "Show Password". Below the input fields are two buttons: a blue "Register" button and a light blue "Clear" button. At the bottom, there is a link that says "If you already have account" followed by a "Login" button.

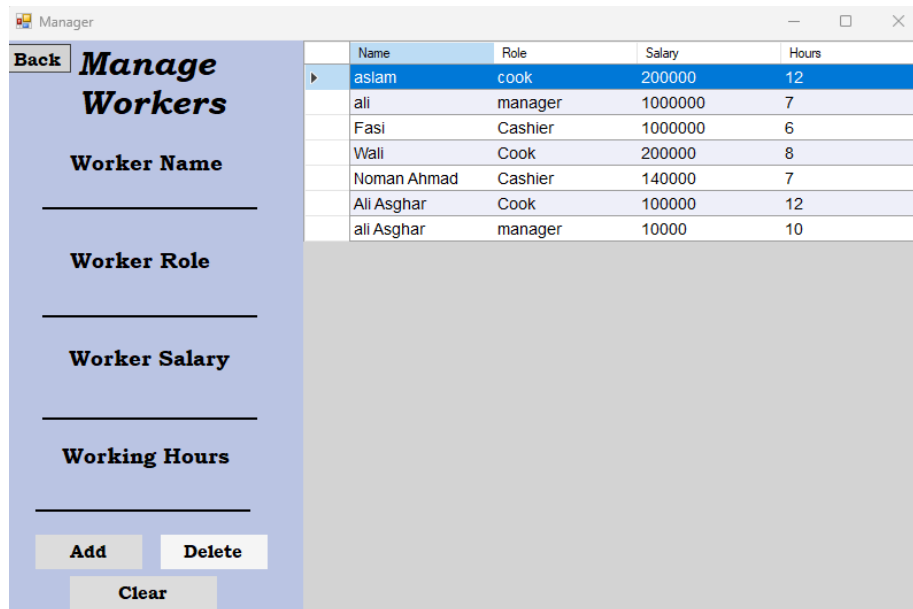
### 6.3. Manager Page:



### 6.4 Manage Product Page:

Name	Quantity	PricePerUnit
rice	100	120
oil	80	340
salt	1000	70
flour	300	160
Cheese	1000	400
Ghee	100	250
rice	100	120
oil	80	340
salt	1000	70
flour	300	160
Cheese	1000	400
Ghee	100	250

## 6.5 Manage Worker Page:



Name	Role	Salary	Hours
aslam	cook	200000	12
ali	manager	1000000	7
Fasi	Cashier	1000000	6
Wali	Cook	200000	8
Noman Ahmad	Cashier	140000	7
Ali Asghar	Cook	100000	12
ali Asghar	manager	10000	10

## 6.6 Manage Expenses Sub-Menu:



### 6.7 Manage Expenses - Product:

Expenses On Products		
Name	Quantity	PricePerUnit
rice	100	120
oil	80	340
salt	1000	70
flour	300	160
Cheese	1000	400
Ghee	100	250
Sum Of Expenses		
Total Expenses are: 582200		

### 6.8 Manage Expenses - Worker:

Expenses On Workers				
Name	Role	Salary	Hours	
aslam	cook	200000	12	
ali	manager	1000000	7	
Fasi	Cashier	1000000	6	
Wali	Cook	200000	8	
Noman Ahmad	Cashier	140000	7	
Ali Asghar	Cook	100000	12	
ali Asghar	manager	10000	10	
Sum Of Expenses				
Total Expenses are: 2650000				

## 6.9 Manage Expenses - Total:

The screenshot shows a web application window titled 'Manager'. The main heading is 'Total Expenses'. On the left, there is a sidebar with a 'Back' button and a section titled 'Other Expenses' containing 'Add' and 'Clear' buttons. The main content area displays the following summary:

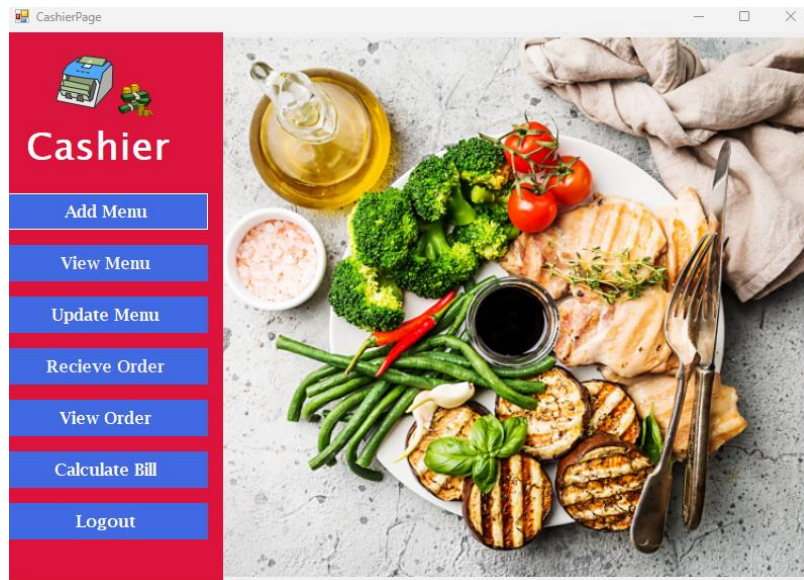
Total Expenses on Products: 582200  
Total Expenses on Workers: 2650000  
Total Other Small Expenses: 121212  
Overall Total Expenses: 3353412

## 6.10 Manage Reviews:

The screenshot shows a web application window titled 'Manager'. The main heading is 'Manage Reviews'. On the left, there is a sidebar with a 'Back' button and three sections: 'Service Review', 'Food Review', and 'Environment Review', each with a 'Delete' and 'Clear' button. The main content area displays a table with the following data:

	Service	Food	Environment
▶	Good	Good	Good
	Excellent	Excellent	Excellent
	Bad	Bad	Bad
	Excellent	Excellent	Excellent
	Good	Good	Good
	Excellent	Excellent	Excellent
	Excellent	Excellent	Excellent

### 6.11 Cashier Page:



### 6.12 Cashier Page – Add Menu:

The screenshot shows the "Add Menu Item" form within the "CashierPage" window. The sidebar on the left is identical to the previous screenshot. The main content area has a light gray background and contains the following elements: a blue header "Add Menu Item", a label "Enter Name of the item:" followed by a text input field, a label "Enter Price of the item:" followed by a text input field, and two buttons at the bottom: a blue "Add Item" button and a white "Clear" button with a blue border.

### 6.13 Cashier Page – View Menu:



Back

## Menu

Name	Price
BIRYANI	200
PALAO	250
BOTTLE	120
PIZZA	1200
JUICE	90
BURGER	200
SALAD	100
SOUP	150
LASAGNIA	1000
PIZZA BURGER	999
OMELETE	50

### 6.14 Cashier Page – Update Menu:



Update Menu

Name: \_\_\_\_\_

New Price: \_\_\_\_\_

Update Item

Delete

Clear

Name	Price
BIRYANI	200
PALAO	250
BOTTLE	120
PIZZA	1200
JUICE	90
BURGER	200
SALAD	100
SOUP	150



### 6.15 Cashier Page – Receive Order:

CashierPage

**Cashier**

Add Menu

View Menu

Update Menu

Recieve Order

View Order

Calculate Bill

Logout

**Recieve Order**

Enter Name Customer:

Enter Name of the item:

Enter Quantity of the item:

Received

Clear

### 6.16 Cashier Page – View Order:

CashierPage

**Cashier**

Add Menu

View Menu

Update Menu

Recieve Order

View Order

Calculate Bill

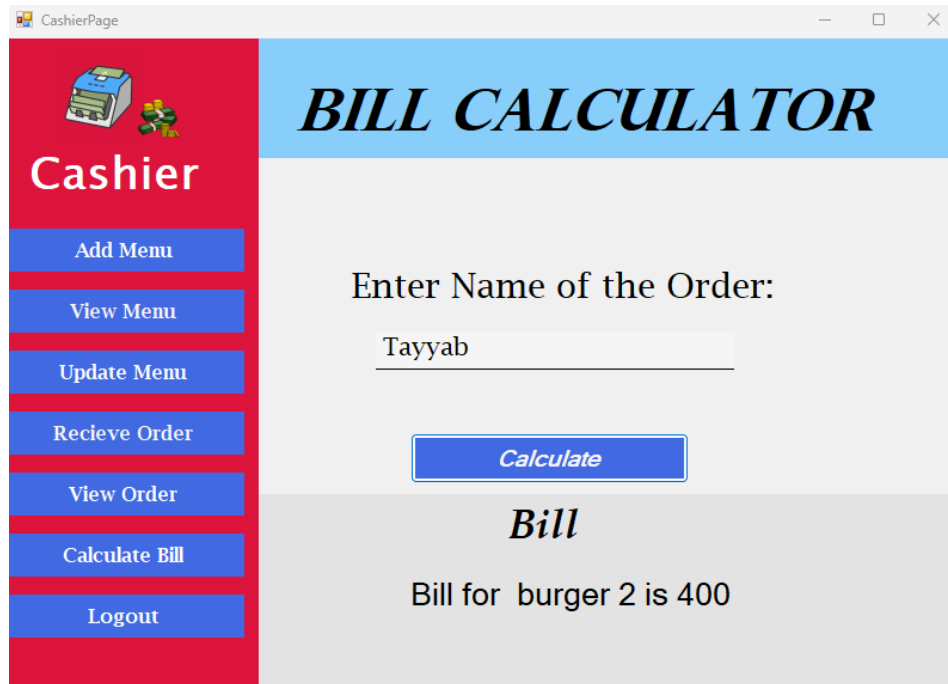
Logout

**Orders**

Name	OrderedItem	OrderQuantity
tayyab	burger	2

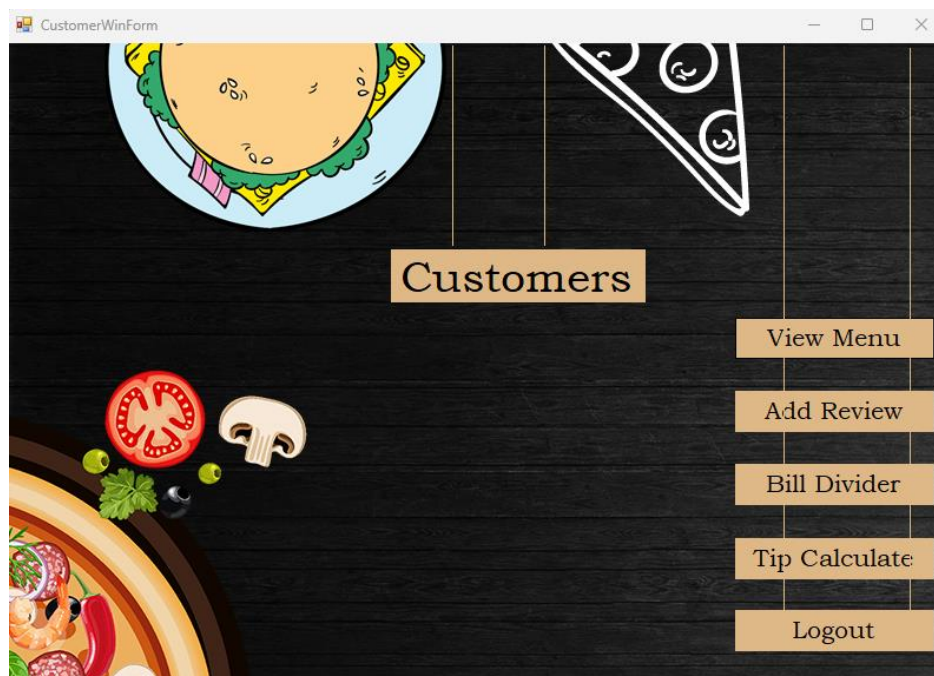
Delete

### 6.17 Cashier Page – Calculate Bill:



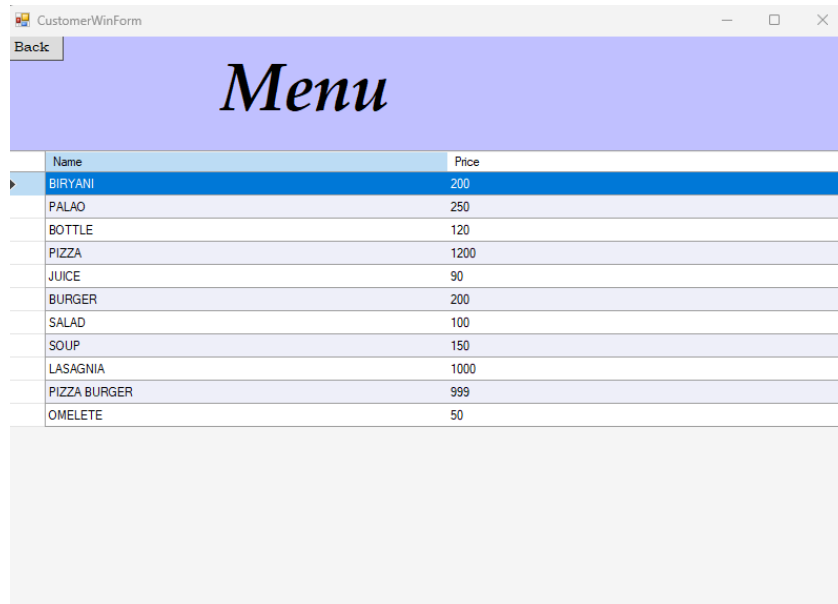
The screenshot shows a window titled "CashierPage". On the left is a red sidebar with a cash register icon and the word "Cashier" in white. Below it are several blue buttons: "Add Menu", "View Menu", "Update Menu", "Recieve Order", "View Order", "Calculate Bill", and "Logout". The main area has a light blue header with the text "BILL CALCULATOR" in a large, bold, italicized font. Below this, it says "Enter Name of the Order:" followed by a text input field containing "Tayyab". A blue button labeled "Calculate" is positioned below the input field. At the bottom, the word "Bill" is displayed in a large, bold, italicized font, followed by the text "Bill for burger 2 is 400".

### 6.18 Customer Page:



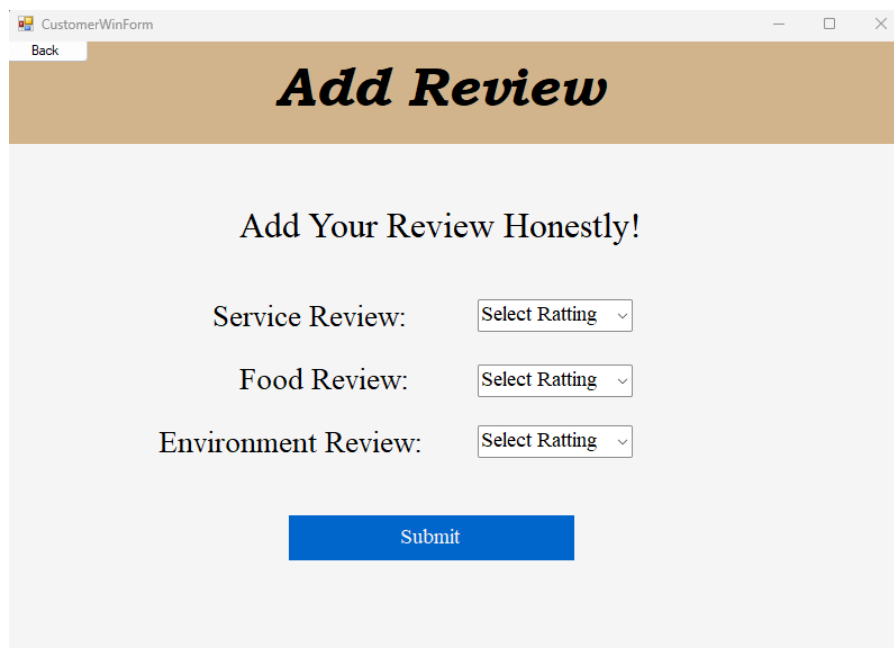
The screenshot shows a window titled "CustomerWinForm". The background is a dark, textured surface with food-related illustrations: a burger at the top, a pizza slice on the right, and a bowl of salad at the bottom left. In the center, the word "Customers" is written in a large, bold, serif font. On the right side, there is a vertical stack of five orange buttons: "View Menu", "Add Review", "Bill Divider", "Tip Calculate", and "Logout".

### 6.19 Customer Page – View Menu:



Name	Price
BIRYANI	200
PALAO	250
BOTTLE	120
PIZZA	1200
JUICE	90
BURGER	200
SALAD	100
SOUP	150
LASAGNIA	1000
PIZZA BURGER	999
OMELETE	50

### 6.20 Customer Page – Add Review:



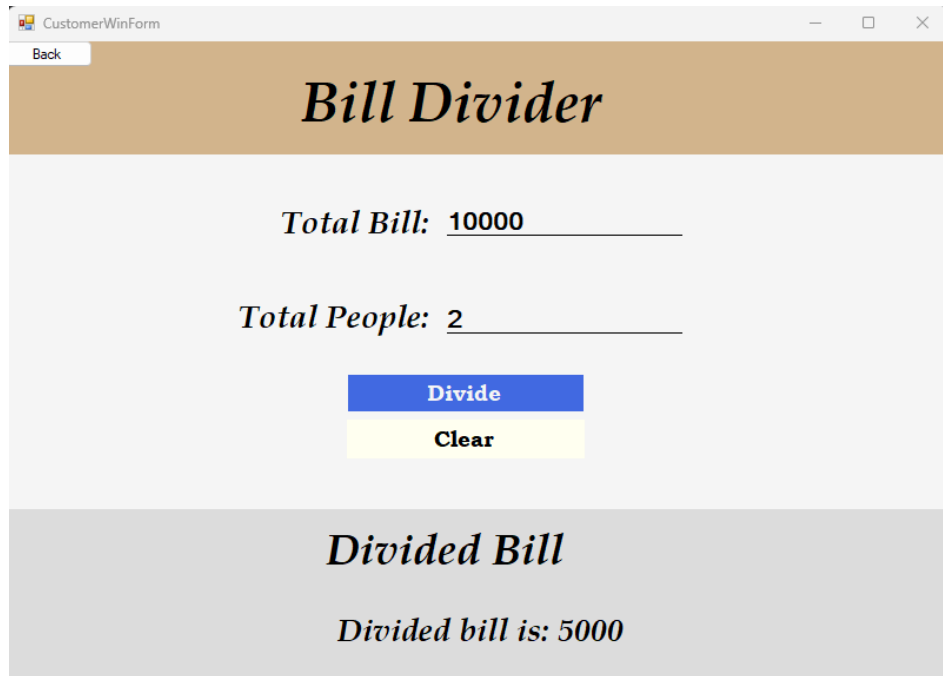
Add Your Review Honestly!

Service Review:

Food Review:

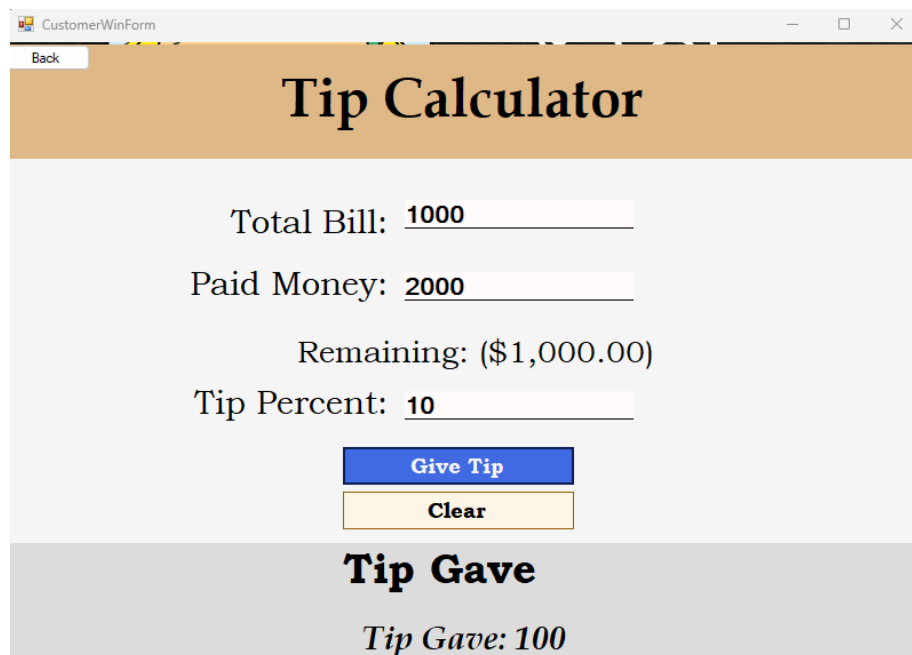
Environment Review:

### 6.21 Customer Page – Bill Divider:



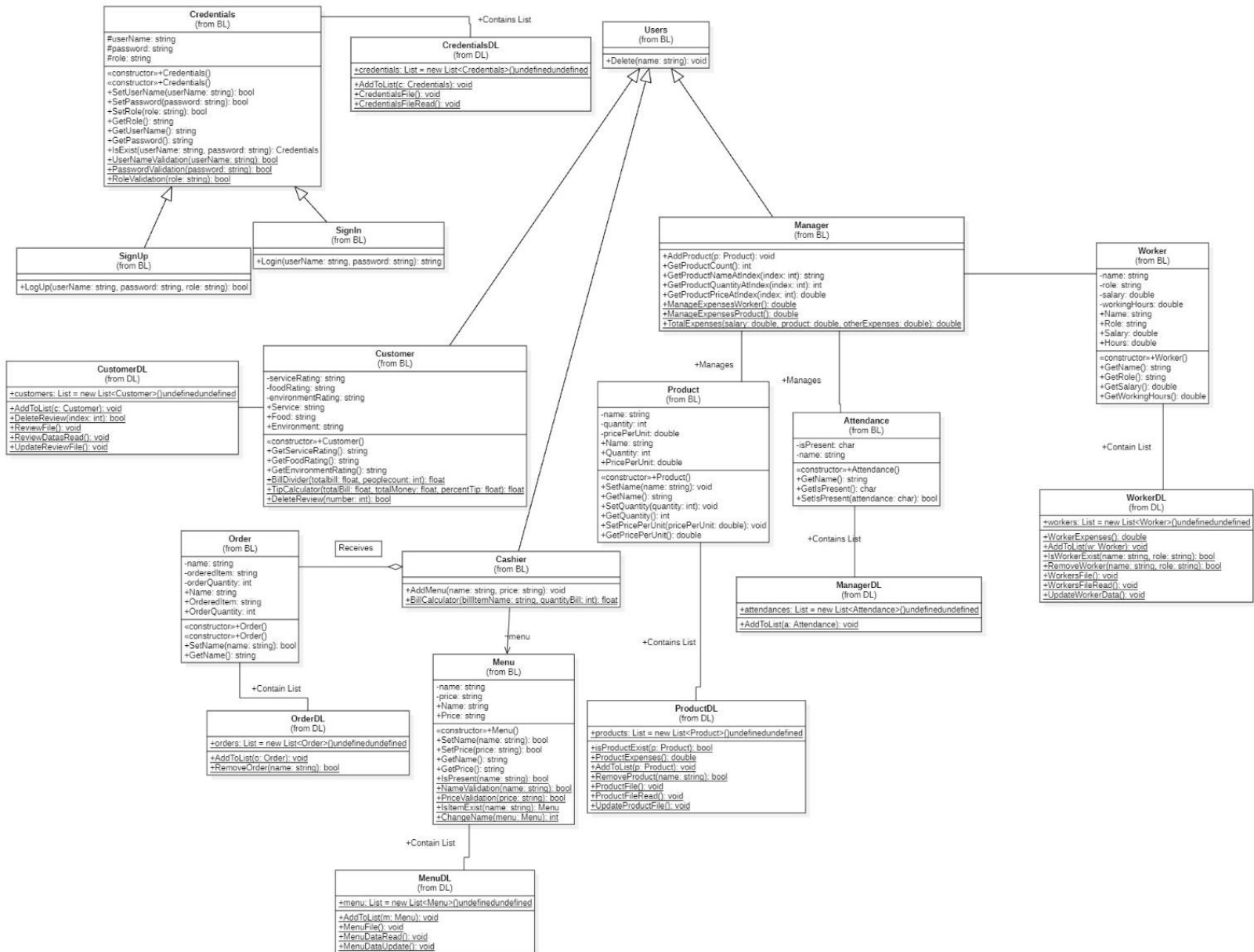
The screenshot shows a Windows application window titled "CustomerWinForm". It features a "Back" button in the top-left corner. The main content area has a light blue header with the title "Bill Divider" in a large, bold, black serif font. Below the header, the text "Total Bill: 10000" is displayed with "10000" in a red font and an underline. This is followed by "Total People: 2" with "2" in a red font and an underline. There are two buttons: a blue "Divide" button and a yellow "Clear" button. The bottom section has a light gray background with the text "Divided Bill" in a large, bold, black serif font, and "Divided bill is: 5000" in a red font below it.

### 6.22 Customer Page –Tip Calculator:



The screenshot shows a Windows application window titled "CustomerWinForm". It features a "Back" button in the top-left corner. The main content area has a light blue header with the title "Tip Calculator" in a large, bold, black serif font. Below the header, the text "Total Bill: 1000" is displayed with "1000" in a red font and an underline. This is followed by "Paid Money: 2000" with "2000" in a red font and an underline. The text "Remaining: (\$1,000.00)" is displayed in a black font. Below this is "Tip Percent: 10" with "10" in a red font and an underline. There are two buttons: a blue "Give Tip" button and a yellow "Clear" button. The bottom section has a light gray background with the text "Tip Gave" in a large, bold, black serif font, and "Tip Gave: 100" in a red font below it.

## 7. Class, Responsibility and Collaboration Diagram (CRC):



## 8. Conclusion:

### 8.1. Summary:

The Hotel Management System aims to automate various tasks and operations within a

hotel, including user authentication, managing products and workers, handling customer interactions, and facilitating cashier operations.

Throughout the project, several key functionalities have been implemented.

Users can sign up and log in with different roles (manager, cashier, or customer) to access relevant features.

Managers can add and view products, manage workers, track expenses and view and remove reviews.

Cashiers can manage the menu, receive orders, calculate bills, and view orders.

Customers can view the menu, leave reviews, divide bills, and calculate tips.

The project demonstrates the implementation of object-oriented programming principles such as association, inheritance, and polymorphism.

## **8.2. Challenges and lessons:**

The basic logic of functions were already available as we did it in procedural programming. But the main challenge was about the implementation of code. It was very difficult to think where this piece of code will go and how we can improve the reusability of code. In short, we can say that the designing of code was the main challenge I faced during this project.

During conversion from CLI to GUI I faces difficulty in use of data grid view. Data binding in Data Grid View gave me a tough time.

Designing and finding the appropriate images for the application was another difficult thing that I faced during the conversion from CLI to GUI.

Learned many valuable lessons during this project. Learned a lot about designing of the code. Although this code might have many designing issues but I learned a lot about it and will improve with more and more projects.