HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INFORMATION TECHNOLOGY

COURSE: OBJECT-ORIENTED PROGRAMMING

GROUP REPORT
COFFEE SHOP MANAGEMENT

ID CLASS: OOPR230279E_25_1_01FIE

INSTRUCTOR: TS. Lê Văn Vinh

GROUP: 3

List of members

Student ID	**Student name**	**Contribution (%)**
24110107	Nguyễn Đăk Lộc	100%
24110128	Lê Tuấn Thành	100%

ACADEMIC YEAR: 2025–2026

Ho Chi Minh City, October 2025

TABLE OF CONTENTS

1. Introduction 1

1.1. Problem Statement 1

1.2. Objectives and Requirements..1

1.3. Scope and Target Users...2

1.4. Methodology and Report Structure...3

2. System Analysis and Design 4

2.1. Project Structure..4

2.2. Class Design & OOP Diagram..5

2.3. System Architecture..9

3. Implementation and Demonstration 10

3.1. Environment and Technology Used..10

3.2. Functional analysis and application of object-oriented features...............10

3.3. Core Business Services 16

4. Results 20

4.1 Login and Authorization Functions 20

4.2 Employee Management Function (For Admin)..........................21

4.3 Menu Management Function (For Admin)................................22

4.4 Business Management (for Manager).......................................23

4.5 Menu Management (for Manager)..24

4.6 Report for product (for Manager) 25

4.7 Cashier Business Process: Sales (for Cashier)...........................26

4.8 Bill Generation and Receipt Printing 29

5. Simulate how the app works 31

6. Conclusion..32

7. Job assignment board 33

8.References 34

1. Introduction

1.1. Problem Statement

In modern coffee shop operations, manual management often leads to inefficiencies such as misplaced orders, slow service, inaccurate billing, and difficulties in tracking employee performance or business statistics. As customer expectations increase, coffee shops require a reliable system that automates order handling, employee management, and reporting while maintaining ease of use and flexibility.

Our project, Coffee Shop Management System, aims to provide a digital solution that supports both the administrative and operational aspects of a coffee shop. The system allows different user roles such as Admin, Manager, Employee, and Cashier to perform various tasks efficiently within an integrated environment. This includes table reservation, menu creation, order processing, billing, and statistical reporting—all managed through a user-friendly interface built in C# using Windows Forms.

At the same time, the project serves as a practical application of Object-Oriented Programming (OOP) concepts learned during the course. It helps us understand how fundamental principles such as encapsulation, inheritance, polymorphism, and abstraction can be used to build a structured, reusable, and scalable software system. During the development of the application, the team upgraded the app so that it could be used not only by the coffee shop but also by other eateries, restaurants, etc.

1.2. Objectives and Requirements

Objectives

Design and implement an object-oriented coffee shop management application.

Enable multiple user roles with appropriate permissions (Admin, Manager, Cashier).

Support the process of menu management, order creation, table booking, and billing.

Generate detailed reports and analytics to assist managers in monitoring performance and making business decisions.

Ensure smooth and intuitive user experience through a modern Windows Forms interface.

Functional Requirements

User Management: Add, edit, delete, and update employee information with role-based access control and salary adjustment features.

Menu Management: Create, edit, and remove menu items, including name, price, and category.

Order and Table Management: Allow customers to place and modify orders, reserve tables, and generate corresponding invoices.

Billing System: Automatically calculate total amounts, apply discounts if any, and print.

Reports: Generate summary reports including daily sales, popular items.

Non-Functional Requirements

Developed using C# (.NET 8) with a Windows Forms graphical user interface.

Provides efficient performance and smooth interaction for small to medium-sized coffee shops.

Includes a user-friendly interface emphasizing usability, clarity, and data consistency.

The team uses SQLite to create the database first, and then upgrade to SQL server if there is time or investment.

1.3. Scope and Target Users

This application is designed for cafe owners, eateries, restaurants, etc., managers and employees who need an integrated system for daily operations and performance monitoring. The system is suitable for small and medium sized cafes looking to transition from manual management to a computerized solution.

In the current version, the system focuses on order processing, staff and menu management, and statistical reporting. Future expansions may include online ordering, inventory management and loyalty discount integration to enhance business scalability and customer experience.

By applying object-oriented programming principles, this project not only serves as a practical management tool but also demonstrates important software engineering concepts such as encapsulation, inheritance, polymorphism, and abstraction.

1.4. Methodology and Report Structure

Before starting development, our team conducted an analysis of real-world coffee shop operations to identify common management challenges such as order tracking, staff coordination, and billing accuracy. From these findings, we established a structured development plan to ensure the system met both functional and practical requirements.

The development process followed an **object-oriented programming (OOP)** approach to ensure modularity, scalability, and code reusability. Each major feature—such as employee management, menu creation, order processing, and reporting—was designed and implemented as an independent class interacting with others through well-defined relationships. The project was built using C# (.NET 8) and Windows Forms, with a relational database for secure and efficient data storage.

During the design stage, class diagrams, workflows, and database schemas were created to define the logical structure of the system. The GUI was then developed to provide an intuitive and user-friendly interface suitable for all user roles (Admin, Manager, Employee, Cashier). After implementation, the system was tested using multiple real-life scenarios to verify its correctness, performance, and usability.

2. System Analysis and Design

2.1. Project Structure

```
source
├── Models              // (business models)
│   └── Catalog
│   │       ├── Dessert.cs
│   │       ├── Drink.cs
│   │       ├── Food.cs
│   │       └── Product.cs
│   └── OrderModel
│   │   ├── OrderItem.cs
│   │   └── Order.cs
│   └── PersonModel
│      ├── Administrator.cs
│      ├── Cashier.cs
│      ├── Customer.cs
│      ├── Employee.cs
│      ├── Manager.cs
│      └── Person.cs
├── Bill.cs
├── EntityBase.cs
├── Report.cs
├── Services            // Business Logic Layer
│   ├── AdminService.cs
│   ├── AuthService.cs
│   ├── EmployeeService.cs
│   ├── OrderService.cs
│   ├── ProductService.cs
│   ├── ReportService.cs
│   └── TableService.cs
├── Common              // class helper and interface
│       ├── EmployeeCompare.cs
│       ├── ISalaryCompute.cs
│       └── MoneyHelper.cs
├── Data                  // Data Access Layer
│       ├──DatabaseSeeder.cs
│       ├──EmployeeAccount.cs
│       ├──OrderRepository.cs
│       └──ProductRepository.cs
├── UI           // User Interface(Winforms)
│   ├── Controls
│    │    ├── FormDragger.cs
│    │    └── Smooth.cs
│    ├── Login
│    │    ├── AddEmployee.cs
│    │    ├── AdminManage.cs
│    │    └── LoginForm.cs
│    ├──BillForm.cs
│    ├── EmployeeDetailForm.cs
│    ├── HomeForm.cs
│    ├── MainForm.cs
│    ├── ManagerForm.cs
│    ├── MenuForm.cs
│    ├── MenuFormForManager.cs
│    ├── OrderForm.cs
│    ├── ProductForm.cs
│    ├── ProductFormForManager.cs
│    ├── QuantityForm.cs
│    └── Report.cs
├──Resources // tài nguyên(icon, ảnh nền,…)
└── Program.cs
```

Figure 2.1: Project folder structure.

2.2. Class Design & OOP Diagram

Meaning: The heart of an object-oriented application lies in the design of classes and the relationships between them. The class diagram below details how the objects in the system interact with each other, thereby applying the core properties of OOP.

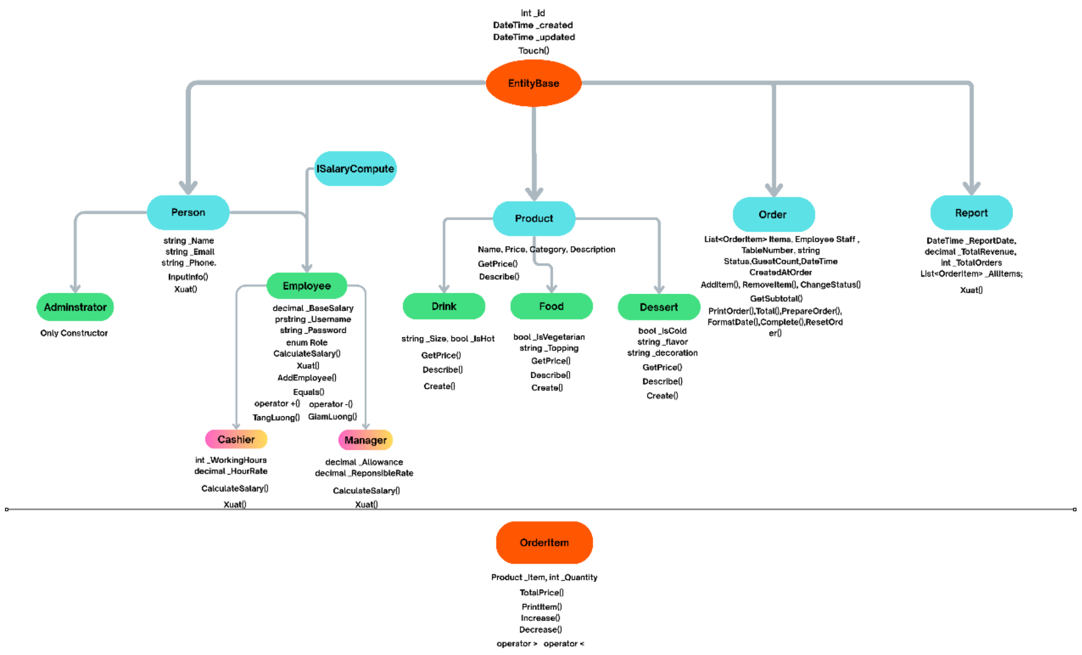


Figure 2.2: Overview class diagram of the system.

OOP Diagram

2.1.1. Encapsulation

Description: This is a fundamental principle that ensures that an object's data is hidden and can only be accessed through public methods.

Evidence: In all Model classes (Person, Employee, Product, …), data is declared as private (private string _Name;…) and accessed through public Properties (public string Name { get; set; },…). This helps control how data is retrieved and changed, ensuring integrity.

```
//Trong class Person
// Data is hidden, not directly accessible from outside.
private string _Name;
private string _Email;
private string _Phone;
public string Name { get { return _Name; } set { _Name = value; } }
public string Email { get { return _Email; } set { _Email = value; } }
public string Phone { get { return _Phone; } set { _Phone = value; } }
      ………
//Trong class Product.cs
private string _Name;
private decimal _Price;
private string _Category;
private string _Description;
// public to securely access data.
public string Name { get { return _Name; } set { _Name = value; } }
public decimal Price { get { return _Price; } set { _Price = value; } }
public string Category { get { return _Category; } set { _Category = value; } }
public string Description { get { return _Description; } set { _Description =
value; } }
// Similar to the class EntityBase.cs, Bill.cs, Food.cs, ….
```

2.1.2. Inheritance

Description: Allows a class (child class) to inherit properties and methods from another class (parent class).

Evidence: The system has 2 main inheritance flows:

EntityBase -> Person -> Employee -> (Manager, Cashier, Administrator).

EntityBase -> Product -> (Food, Drink, Dessert).

Benefits: Effective code reuse, building a logical and easy-to-understand hierarchical structure.

```
// Abstract parent class, defines common properties.
public abstract class EntityBase
{
      …//another
}
public abstract class Product:EntityBase
{
      …//another
}
public class Food:Product// Subclass inherits from Product
{
      …/another
}
```

```
public class Drink:Product
{
      …//another
}//another class inherited.
```

Explanation: The above code shows that the Food class inherits from Product, thereby automatically getting the Name and Price properties without having to redefine them. This is code reuse in practice.

2.1.3. Polymorphism

Description: Allows different objects to react differently to the same message (function call) (virtual -override).

Evidence: Best demonstrated by the virtual decimal CalculateSalary() method in Employee and overridden in Manager and Cashier. The same call to employee.CalculateSalary() will execute different salary calculation formulas depending on the actual object.

```
public abstract class Product:EntityBase
{
      ….
        public abstract decimal GetPrice();
        public virtual string Describe()
        {
            return $"{Name} ({Category}) - {Price}đ";
}
public class Food:Product
{
        public override decimal GetPrice()
        {
            decimal toppingFee = !string.IsNullOrEmpty(Topping) ? 5000m : 0m;
            return Price + toppingFee;
        }
        public override string Describe()
        {
            return $"{Name} {(IsVegetarian ? "(Vegan)" : "")}
{(!string.IsNullOrEmpty(Topping) ? $"[Topping: {Topping}]" : "")} -
{GetPrice():N0}đ";
        }
}
// Similar to the other class polymorphism at class Product(Drink,Dessert)
```

public abstract decimal GetPrice();

Meaning: This is a "command". The parent class Product says: "Anyone who inherits from me must redefine their own price calculation method".

How to create Polymorphism: The Drink.cs class will redefine (override) GetPrice() to calculate the price by Size.

The Food.cs class will redefine (override) GetPrice() to calculate the price by Topping.

The Dessert.cs class will redefine (override) GetPrice() to calculate the price by Decoration.

public virtual string Describe()

Meaning: This is a "suggestion". The parent class Product says: "This is my default description. Subclasses can use it, or if they want, they can redefine (override) a better description".

How to create Polymorphism: Drink.cs will override Describe() to display both Size and IsHot information.

Food.cs will override Describe() to also display information about IsVegetarian

2.1.4. Abstraction

Description: Hides complexity and exposes only the necessary functionality.

Evidence:

Abstract Class: Person and Product classes are declared abstract, they define a common "concept" but cannot be instantiated directly.

Interface: The ISalaryCompute interface defines a "contract" that the Employee class must adhere to, ensuring that every employee must be able to calculate salaries.

```
public abstract class Person:EntityBase
{
}
public interface ISalaryCompute
{
    decimal CalculateSalary();
}
public class Employee : Person, ISalaryCompute
{
}
```

2.1.5. Advanced OOP Mechanisms

Operator Overloading

Description: Allows redefining the behavior of operators (like +, -, >, <) for custom data types.

Example: Operator > is overloaded in the OrderItem class to compare quantities, making the code in ReportService more natural (if (item > best)).

```
public static Employee operator +(Employee a, decimal TangLuong)
{
    a.BaseSalary += TangLuong;
    return a;
}
public static Employee operator -(Employee a, decimal GiamLuong)
{
    a.BaseSalary = Math.Max(0, a.BaseSalary - GiamLuong);
    return a;
}
```

Static Members:

Description: Members that belong to the class itself, instead of a specific object.

Example: The static int pIdCounter variable in the Bill.cs class is used to automatically generate a unique invoice ID for all invoices.

```
public static void AddEmployee(List<Employee> list, Employee emp)
{
    list.Add(emp);
}
```

2.3. System Architecture

To ensure the project had a clear structure, was easy to maintain and extend, the team organized the source code based on the principle of separation of responsibilities. This process naturally led to a model very similar to the 3-Layer Architecture, a standard model in software development.

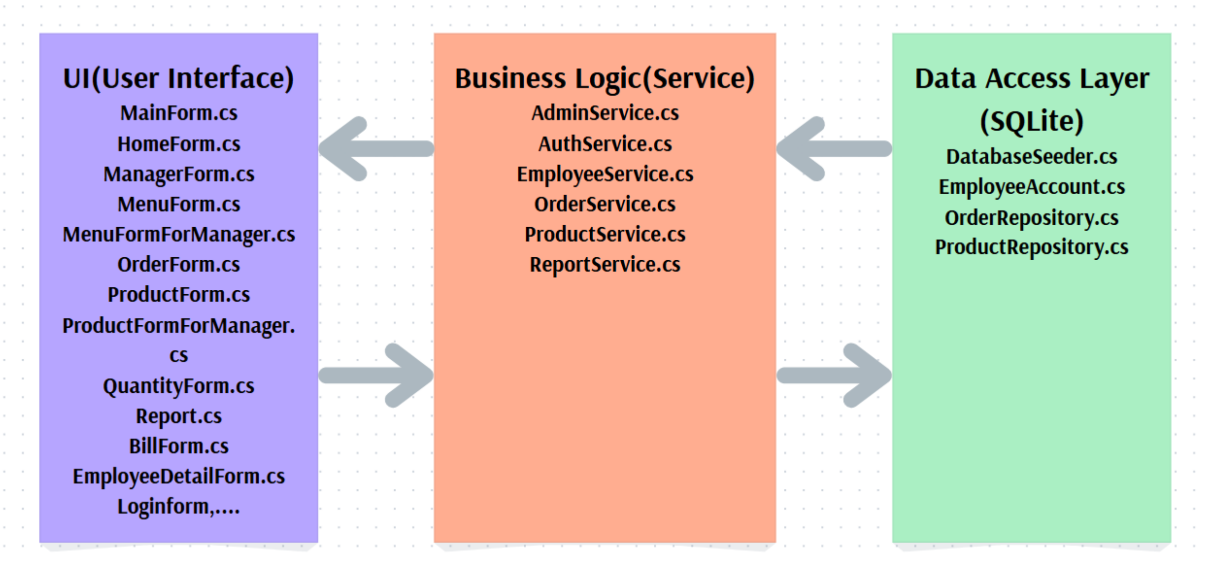


Figure 2.1: 3-layer Architectural Diagram of the Project

3. Implementation and Demonstration

3.1. Environment and Technology Used

Development Environment (IDE): Visual Studio 2022

Platform & Language: .NET 8, C#

User Interface (UI): Windows Forms (WinForms)

Database: SQLite

3.2. Functional analysis and application of object-oriented features

3.2.1. Human Resource Management

Main function: Build and manage a system of employees with different roles (Administrator, Manager, Cashier) and salaries. For users who are Managers, they can increase or decrease the salary for cashiers, sort by name and salary level.

Typical functions performed

`CalculateSalary()` (in `Employee`, `Manager`, `Cashier`): Calculate salary for employees.

```
//in class (parent: Employee.cs)
public virtual decimal CalculateSalary()
{
    return BaseSalary;
}
//In class Manager.cs
public override decimal CalculateSalary()
{
    return BaseSalary + Allowance + (BaseSalary * ReponsibleRate*10);
}
//in class Cashier.cs
public override decimal CalculateSalary()
 {
     return BaseSalary +(_WorkingHours * _HourRate);
 }
```

`AddEmploy()`, `UpdateEmploy()`, `DeleteEmploy()` (in `AdminService`): Employee management operations.

```
//In class AdminService.cs
//xóa 1 nhân viên
public void DeleteEmploy(int eid)
{
    EmployeeAccount.DeleteEmployee(eid);
}
//cập nhật, edit nhân viên
public void UpdateEmploy(Employee e)
```

```
{
    EmployeeAccount.UpdateEmployee(e);
}
//check và thêm
public void AddEmploy(Employee e)
{
    if (EmployeeAccount.UsernameExists(e.Username))
    {
        throw new Exception("Tên đăng nhập đã tồn tại!");
    }
    EmployeeAccount.AddEmployee(e);
}
```

`IncreaseSalary()`,`DecreaseSalary()`, `SortEmployees()` (in `EmployeeService`):

Employee management operations.

```
//in Employee.cs
public void TangLuong(decimal amount)
{
    this.BaseSalary += amount;
}
public void GiamLuong(decimal amount)
{
    this.BaseSalary -= amount;
}
//in EmployeeCompare.cs
public class ByName : IComparer<Employee>
{
    public int Compare(Employee x, Employee y)
    {
        return string.Compare(x.Name, y.Name);
    }
}

public class BySalaryDescending : IComparer<Employee>
{
    public int Compare(Employee x, Employee y)
    {
        return y.CalculateSalary().CompareTo(x.CalculateSalary());
    }
}
//in EmployeeService.cs
//tăng lương
public void IncreaseSalary(Employee employee, decimal amount)
{
    if (employee == null || amount <= 0) return;

    employee.TangLuong(amount);
    EmployeeAccount.UpdateEmployee(employee);
}
//sắp xếp
public List<Employee> SortEmployees(List<Employee> employees, string criteria)
{
    if (criteria == "Name")
    {
        employees.Sort(new EmployeeCompare.ByName());
    }
    else if (criteria == "Salary")
    {
        employees.Sort(new EmployeeCompare.BySalaryDescending());
    }
    return employees;}
```

Meaning and Properties of OOP applied

Inheritance: Used to build the Person -> Employee -> Manager/Cashier family tree, helping to reuse code for common information and create a logical structure.

Polymorphism: Is the "soul" of the CalculateSalary() function. By using virtual and override, the system can automatically call the correct salary formula for each role, making the code extremely flexible and easy to extend.

Encapsulation: Basic salary information is protected, and adding/editing/deleting employees must go through the AdminService, ensuring that business rules are followed.

Abstraction: The SortEmployees function uses the IComparer<Employee> "contract". This allows it to sort by multiple criteria (ByName, BySalary) flexibly.

3.2.2. Sales & Billing &Order Management

Main function: Allows staff to create orders, add various products (Food, Drinks...) and manage menus.

Typical functions performed

`AddItem()` (in `Order`): Add a product to the current order.

`Total()` (in `Order`): Calculate the total amount of an order.

`GetPrice()` (in `Product`, `Food`, `Drink`): Get the price of the product, taking into account additional factors (size, toppings).

```
//In Order.cs
public void AddItem(Product product, int quantity)
{
    if (product == null || quantity <= 0) return;
    var exist = Items.FirstOrDefault(i => i.Item.Name == product.Name);
    if (exist != null)
    {
        exist.Quantity += quantity;
    }
    else
    {
        Items.Add(new OrderItem(product, quantity));
    }
    Touch();
}
public decimal Total()
{
    decimal total = 0;
    foreach (var i in Items)
    {
        total += i.TotalPrice();
    }
    return total;
}
```

```
//In Product.cs
public abstract decimal GetPrice();
//In Food.cs
public override decimal GetPrice()
{
    decimal toppingFee = !string.IsNullOrEmpty(Topping) ? 5000m : 0m;
    return Price + toppingFee;
}
//in Drink.cs
public override decimal GetPrice()
{
    return Size.ToLower() switch
    {
        "Nhỏ" => Price,
        "To" => Price + 5000m,
        _ => Price
    };
}
//In Dessert.cs
public override decimal GetPrice()
{
    decimal extra = 0m;
    if (IsCold) extra += 3000m;
    if (!string.IsNullOrWhiteSpace(Decoration)) extra += 2000m;
    return Price + extra;
}
```

Meaning and Properties of OOP applied

Inheritance & Abstraction: The abstract Product class acts as a general "template". This allows an Order to contain any type of product (Food, Drink...) without having to process them individually.

Polymorphism: The GetPrice() function is overridden in each product type to apply different price calculation formulas, demonstrating the flexibility of the system.

3.2.3. Reporting & Analytics

Main function: Aggregate data from completed orders to provide an overview of business situation.

Typical functions performed

TotalComprehensiveReport() (in ReportService): Collects and processes all data to create a complete report.

operator > (in OrderItem): Compares the quantity of two items.

```
public Report TotalComprehensiveReport()
{
    var allOrders = orderRepo.GetAllOrders();
    var report = new Report();

    if (allOrders == null || !allOrders.Any())
    {
        return report;
    }

    report.TotalOrders = allOrders.Count;
    var allItems = allOrders.SelectMany(order => order.Items).ToList();
    report.AllItems = allItems;
    report.TotalRevenue = allItems.Sum(item => item.TotalPrice());

    if (allItems.Any())
    {
        var groupedItems = new Dictionary<string, OrderItem>();
        foreach (var item in allItems)
        {
            string name = item.Item.Name;
            if (groupedItems.ContainsKey(name))
            {
                groupedItems[name].Quantity += item.Quantity;
            }
            else
            {
                groupedItems[name] = new OrderItem(item.Item, item.Quantity);
            }
        }
        // Logic tìm món bán chạy nhất (GetBestSeller)
        var best = groupedItems.Values.First();
        foreach (var item in groupedItems.Values)
        {
            if (item > best)
            {
                best = item;
            }
        }
        report.BestSeller = best;

        // Logic sắp xếp và lấy top 3 (GetTopSellingItems)
        var sortedList = groupedItems.Values.ToList();
        // Dùng vòng lặp sort
        sortedList.Sort((item1, item2) =>
item2.Quantity.CompareTo(item1.Quantity));
        report.TopSelling = sortedList.Take(3).ToList();
    }

    return report;
}
public static bool operator >(OrderItem a, OrderItem b)
{
    return a.Quantity > b.Quantity;
}

public static bool operator <(OrderItem a, OrderItem b)
{
    return a.Quantity < b.Quantity;
}
```

Meaning and Properties of OOP applied

Encapsulation (architecture level): This is a perfect example of a 3-tier architecture. All the complex computation logic is neatly encapsulated inside the ReportService. The interface (ReportForm) does not need to know this complexity, it just calls a function and gets the result.

Operator Overloading: Overloading the > operator makes the code for finding the best-selling item in the ReportService more natural and readable (if (item > best)).

3.2.4. Menu Management

Main function: Allows managers to perform full CRUD operations (Add, Edit, Delete) on products in the menu.

Typical functions performed

`Add()`, `Update()`, `Delete()` (in `ProductService`): add, update (edit), delete items from the menu.

```
private ProductRepository _repo = new ProductRepository();
private List<Product> _products;
public void Add(Product p)
{
    _repo.Add(p);
    _products.Add(p);
}

public void Update(Product old, Product updated)
{
    _repo.Update(old, updated);
    int index = _products.IndexOf(old);
    if (index >= 0) _products[index] = updated;
}

public void Delete(Product p)
{
    _repo.Delete(p);
    _products.Remove(p);

}
```

Meaning and Properties of OOP applied

Inheritance & Abstraction: The abstract Product class acts as a common "template", allowing ProductService to manage a List<Product> without caring whether it is Food or Drink.

Encapsulation (at the architectural level): The entire menu management logic is neatly encapsulated inside the ProductService. The interface (MenuForm) just issues commands without knowing the details of how the data is stored.

3.3. Core Business Services

3.3.1 AuthService – Authentication & Role-Based Authorization

Business Role: AuthService is responsible for verifying user credentials and assigning the correct role (Admin, Manager, Cashier) upon login. It encapsulates all authentication and authorization logic, ensuring that the UI remains clean and role-agnostic.

Workflow Summary

Input Validation: Checks for empty username or password before querying the database.

```
if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(password))
    throw new Exception("Username and password cannot be blank.");
```

Credential Verification: Calls down to `EmployeeAccount.Authenticate()` to verify credentials.

Error Handling: If authentication fails, throws a clear exception:

```
if (user == null)
    throw new Exception ("Wrong account or password!!!");
```

Role Assignment: Returns a fully populated Employee object with the _Role property, which determines what the user can access.

Encapsulated Method

```
public Employee Login(string username, string password)
{
    if (string.IsNullOrWhiteSpace(username) ||
string.IsNullOrWhiteSpace(password))
        throw new Exception ("Username and password cannot be blank.");

    var user = Authenticate(username, password);

    if (user == null)
        throw new Exception ("Wrong account or password!!!");

    return user;
}
```

Architectural Strengths

Encapsulation of role logic: The UI doesn't need to know how many roles exist—it simply reacts to the returned _Role.

Separation of concerns: The UI focuses on display; AuthService handles all access control rules.

3.3.2 EmployeeService – Human Resource Operations

Business Objective: EmployeeService supports operations such as adjusting salaries, sorting employees, and filtering staff by role. Unlike AdminService, which handles basic CRUD, this service encapsulates business rules that reflect real-world HR workflows, allowing Managers to manage their teams effectively without needing to understand internal system structures.

Workflow Summary

Role-Based Filtering: Managers can only view and manage employees who are not Admins or other Managers. This is enforced via:

```
return EmployeeAccount.GetAllEmployees()
    .Where(emp => emp._Role != Employee.Role.Admin && emp._Role != 
Employee.Role.Manager)
    .ToList();
```

Salary Adjustment: Salary changes are processed in two steps: update the in-memory object (TangLuong() or GiamLuong()), then persist the change to the database.

```
public void IncreaseSalary(Employee employee, decimal amount)
{
    if (employee == null || amount <= 0) return;
    employee.TangLuong(amount);
    EmployeeAccount.UpdateEmployee(employee);
}
```

Sorting Employees: Sorting is implemented using the Strategy Pattern. Depending on the selected criteria, the service applies the appropriate comparer:

```
if (criteria == "Name")
    employees.Sort(new EmployeeCompare.ByName());
else if (criteria == "Salary")
    employees.Sort(new EmployeeCompare.BySalaryDescending());
```

Architectural Strengths

Encapsulation of Salary Logic: The UI only triggers actions; all validation and updates are handled within the service.

Role-Based Access Control: Managers are restricted to managing only authorized staff, protecting sensitive data.

Strategy Pattern: Sorting logic is abstracted into separate comparer classes, making it easy to extend.

Polymorphism: Each employee type overrides CalculateSalary() with its own formula, allowing the service to compute salaries without knowing the specific role.

3.3.3 OrderService – Order Lifecycle Management

Business Objective: OrderService manages the full lifecycle of a customer order—from table selection to item management and final payment. It ensures that all operations follow business rules such as merging duplicate items, validating order state, and cleaning up completed orders.

Workflow Summary

Table Initialization: When a cashier selects a table, the service checks for an existing active order. If found, it returns it; otherwise, it creates a new one:

```
var existingOrder = _orderRepo.GetFullOrderByTable(tableNumber);
return new Order().PrepareOrder(existingOrder, tableNumber);
```

Item Management: When adding a product, the service checks if it already exists in the order. If so, it increases the quantity; otherwise, it adds a new OrderItem.

Finalize Payment: Validates the order, generates a Bill, and deletes the active order:

```
if (orderToPay == null || !orderToPay.Items.Any() || orderToPay.Id <= 0)
    throw new InvalidOperationException("Invalid order.");
var bill = new Bill(orderToPay);
_orderRepo.Delete(orderToPay.Id);
return bill;
```

Architectural Strengths

Encapsulation: All order logic is handled within the service; the UI simply triggers actions.

Data Integrity: Prevents invalid or empty orders from being processed.

Scalability: Easily adaptable for dine-in, takeout, or delivery workflows.

3.3.4 ReportService – Business Analytics & Insights

Business Objective: ReportService transforms raw order data into actionable insights, helping managers monitor performance through metrics like total revenue, best-selling products, and top-selling items.

Workflow Summary

Data Collection: Retrieves all completed orders and flattens them into a list of OrderItems.

Aggregation & Analysis: Groups items by name, sums quantities, and calculates total revenue:

```
report.TotalRevenue = allItems.Sum(item => item.TotalPrice());
```

Best-Seller & Ranking: Uses a dictionary to group items and determine the best-seller. Then sorts and selects the top 3:

```
sortedList.Sort((item1, item2) => item2.Quantity.CompareTo(item1.Quantity));
report.TopSelling = sortedList.Take(3).ToList();
```

Architectural Strengths

Encapsulation: The UI only calls TotalComprehensiveReport() and receives a ready-to-use Report object.

Operator Overloading: The > operator in OrderItem simplifies best-seller comparison.

Extensibility: New metrics (e.g., average order value, hourly trends) can be added without changing the UI.

4. Results

4.1 Login and Authorization Functions

Description: The system starts with a single login screen. Based on the role of the successfully logged in account (Admin, Manager, or Cashier), the main interface will automatically display the allowed functions, ensuring security and correct role.

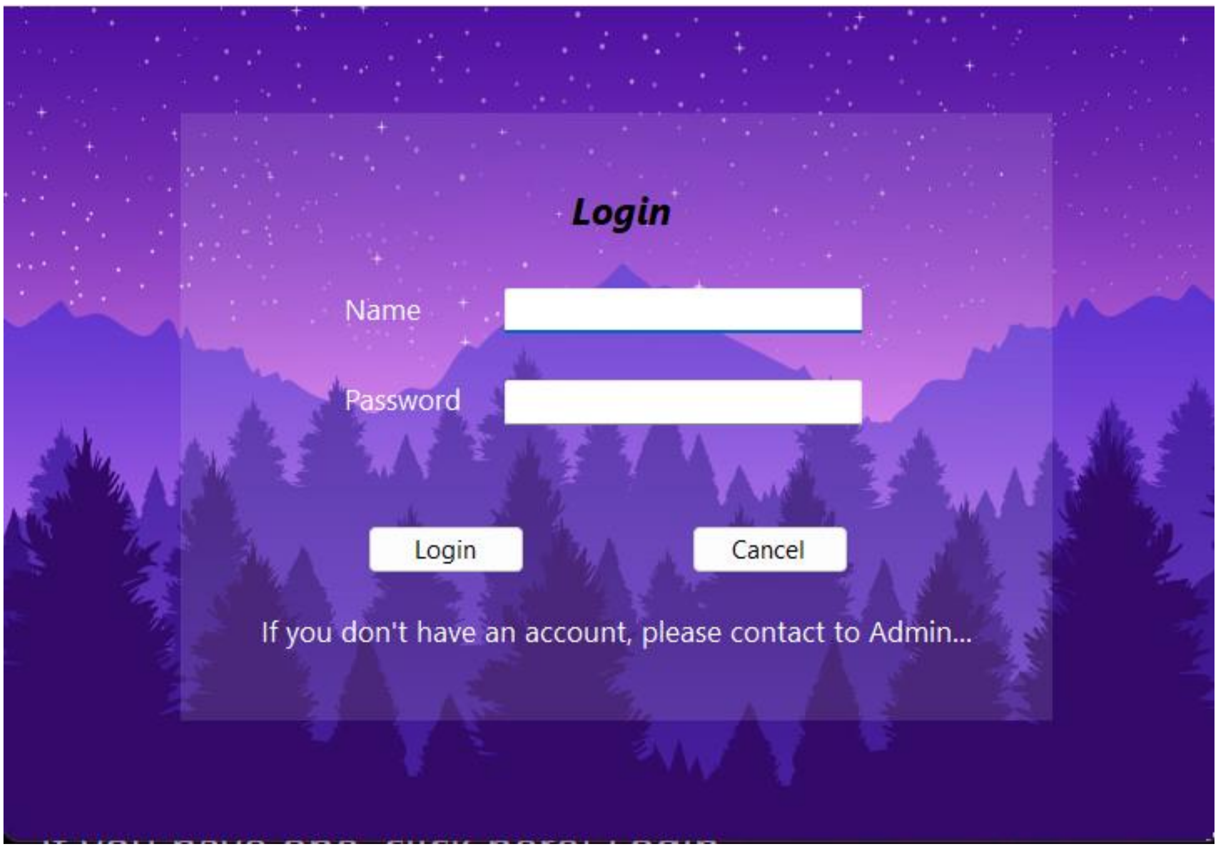


Figure 4.1: System Login interface.

Explanation

After the user enters information and clicks "Login", LoginForm will call AuthService (BLL layer) for authentication.

AuthService then calls down to EmployeeAccount (DAL layer) to check with the database.

If successful, the system will open the working interface corresponding to that user's role. The example below is the interface for Administrator with full management buttons.

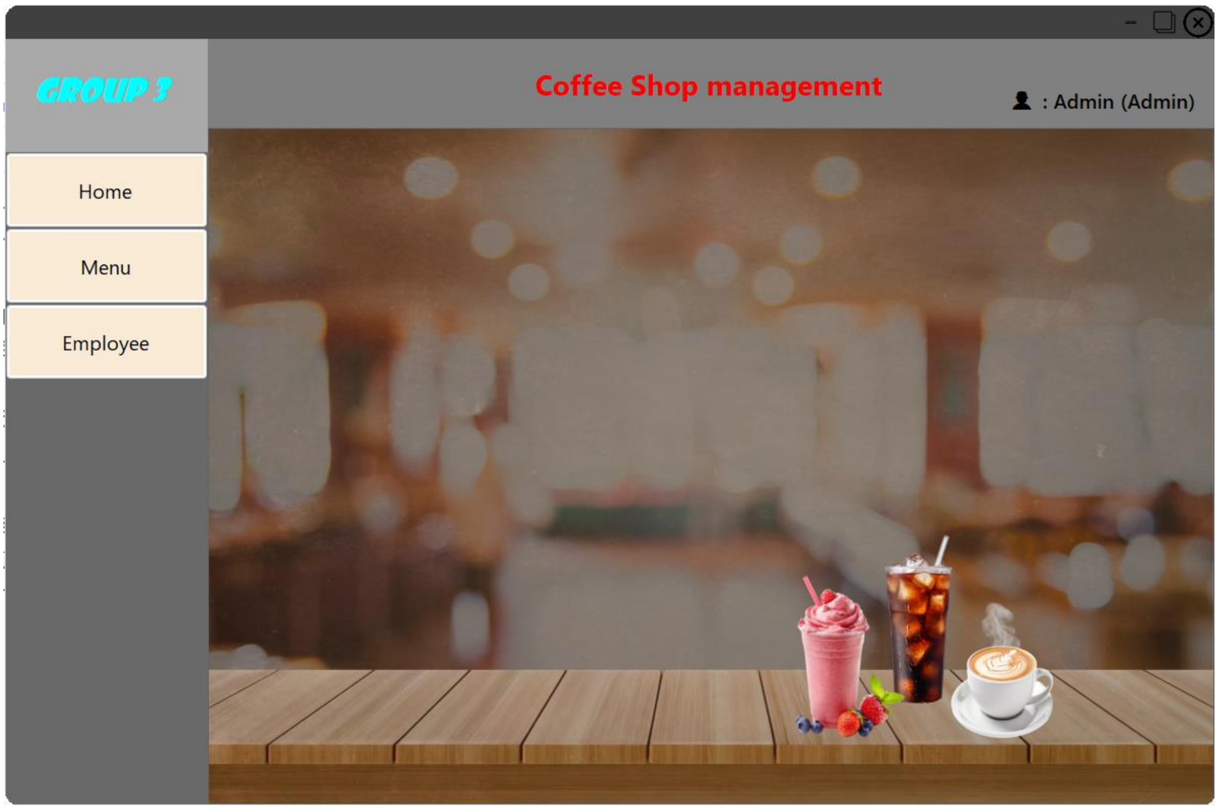
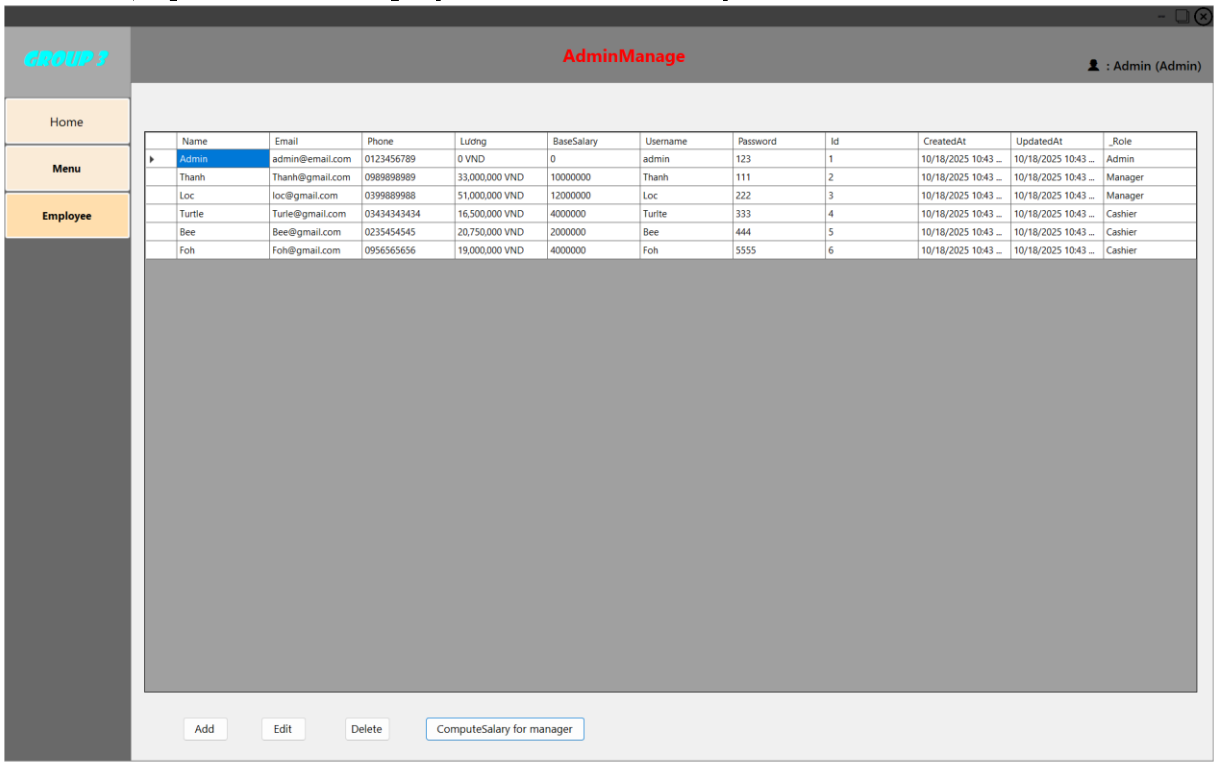


Figure 4.1.1.: The working interface of Administrator after successful login, showing full management function buttons.

4.2 Employee Management Function (For Admin)

Description: This function allows Administrator to perform full CRUD (Add, Edit, Delete) operations on employee accounts in the system.



Name	Email	Phone	Lương	BaseSalary	Username	Password	Id	CreatedAt	UpdatedAt	_Role
Admin	admin@email.com	0123456789	0 VND	0	admin	123	1	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Admin
Thanh	Thanh@gmail.com	0989898989	33,000,000 VND	10000000	Thanh	111	2	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Manager
Loc	loc@gmail.com	0399889988	51,000,000 VND	12000000	Loc	222	3	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Manager
Turtle	Turle@gmail.com	03434343434	16,500,000 VND	4000000	Turlte	333	4	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Cashier
Bee	Bee@gmail.com	0235454545	20,750,000 VND	2000000	Bee	444	5	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Cashier
Foh	Foh@gmail.com	0956565656	19,000,000 VND	4000000	Foh	5555	6	10/18/2025 10:43 ...	10/18/2025 10:43 ...	Cashier

Figure 4.2: Employee Management Interface.

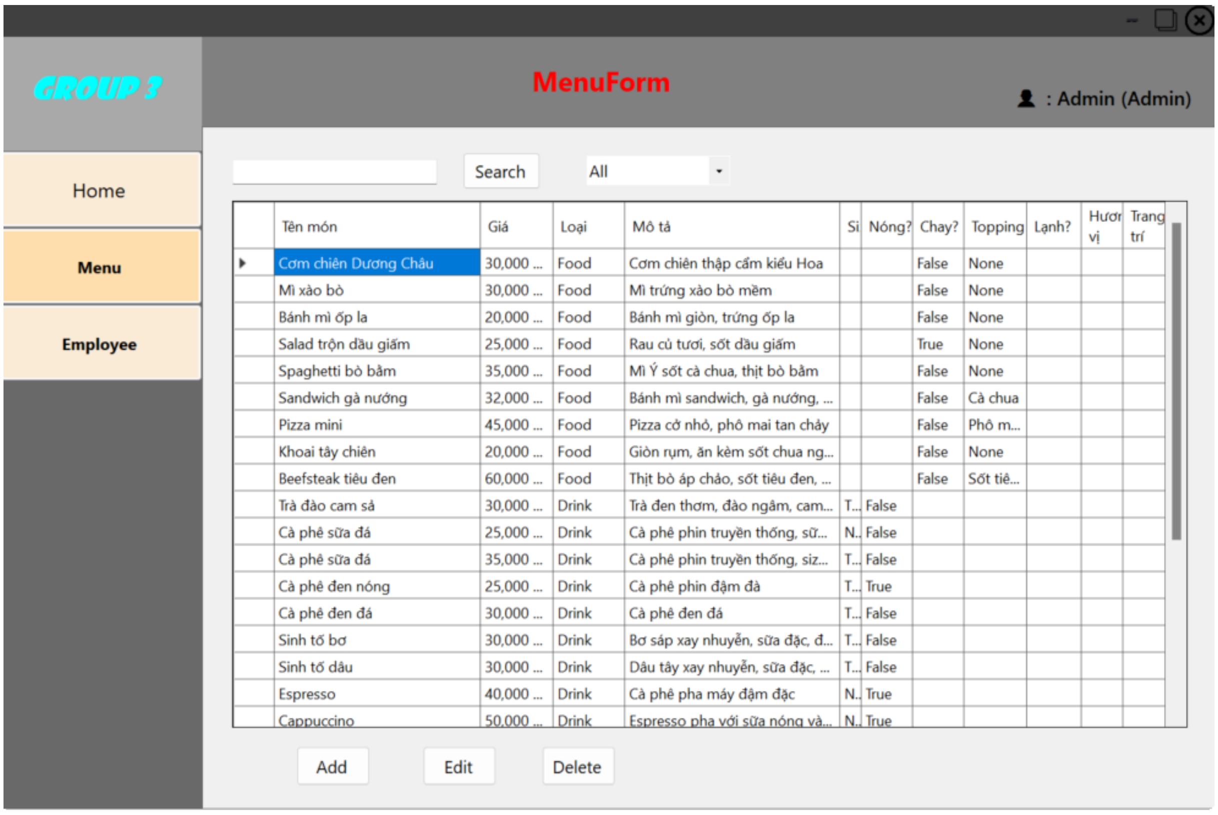
Explanation

(1) Display: Employee data is retrieved from EmployeeAccount by AdminService and displayed on DataGridView.

(2) Add/Edit/Delete: When Admin performs an operation (e.g., press "Delete" button), AdminManageForm will call the corresponding function in AdminService (e.g., _adminService.DeleteEmploy(employeeId)). All business logic and calling down to DAL layer to update database are handled by Service, Form only has the task of giving command and updating interface.

4.3 Menu Management Function (For Admin)

Description: This function allows Administrator to perform full CRUD operations (Add, Edit, Delete) for product types (food, drink, dessert) in the system.



Tên món	Giá	Loại	Mô tả	Si	Nóng?	Chay?	Topping	Lạnh?	Hương vị	Trang trí
Cơm chiên Dương Châu	30,000 ...	Food	Cơm chiên thập cẩm kiểu Hoa			False	None			
Mì xào bò	30,000 ...	Food	Mì trứng xào bò mềm			False	None			
Bánh mì ốp la	20,000 ...	Food	Bánh mì giòn, trứng ốp la			False	None			
Salad trộn dầu giấm	25,000 ...	Food	Rau củ tươi, sốt dầu giấm			True	None			
Spaghetti bò bằm	35,000 ...	Food	Mì Ý sốt cà chua, thịt bò bằm			False	None			
Sandwich gà nướng	32,000 ...	Food	Bánh mì sandwich, gà nướng, ...			False	Cà chua			
Pizza mini	45,000 ...	Food	Pizza cỡ nhỏ, phô mai tan chảy			False	Phô m...			
Khoai tây chiên	20,000 ...	Food	Giòn rụm, ăn kèm sốt chua ng...			False	None			
Beefsteak tiêu đen	60,000 ...	Food	Thịt bò áp chảo, sốt tiêu đen, ...			False	Sốt tiê...			
Trà đào cam sả	30,000 ...	Drink	Trà đen thơm, đào ngâm, cam...	T...	False					
Cà phê sữa đá	25,000 ...	Drink	Cà phê phin truyền thống, sữ...	N...	False					
Cà phê sữa đá	35,000 ...	Drink	Cà phê phin truyền thống, siz...	T...	False					
Cà phê đen nóng	25,000 ...	Drink	Cà phê phin đậm đà	T...	True					
Cà phê đen đá	30,000 ...	Drink	Cà phê đen đá	T...	False					
Sinh tố bơ	30,000 ...	Drink	Bơ sáp xay nhuyễn, sữa đặc, đ...	T...	False					
Sinh tố dâu	30,000 ...	Drink	Dâu tây xay nhuyễn, sữa đặc, ...	T...	False					
Espresso	40,000 ...	Drink	Cà phê pha máy đậm đặc	N...	True					
Cappuccino	50,000 ...	Drink	Espresso pha với sữa nóng và...	N...	True					

Figure 4.3: Menu Management interface.

Explanation

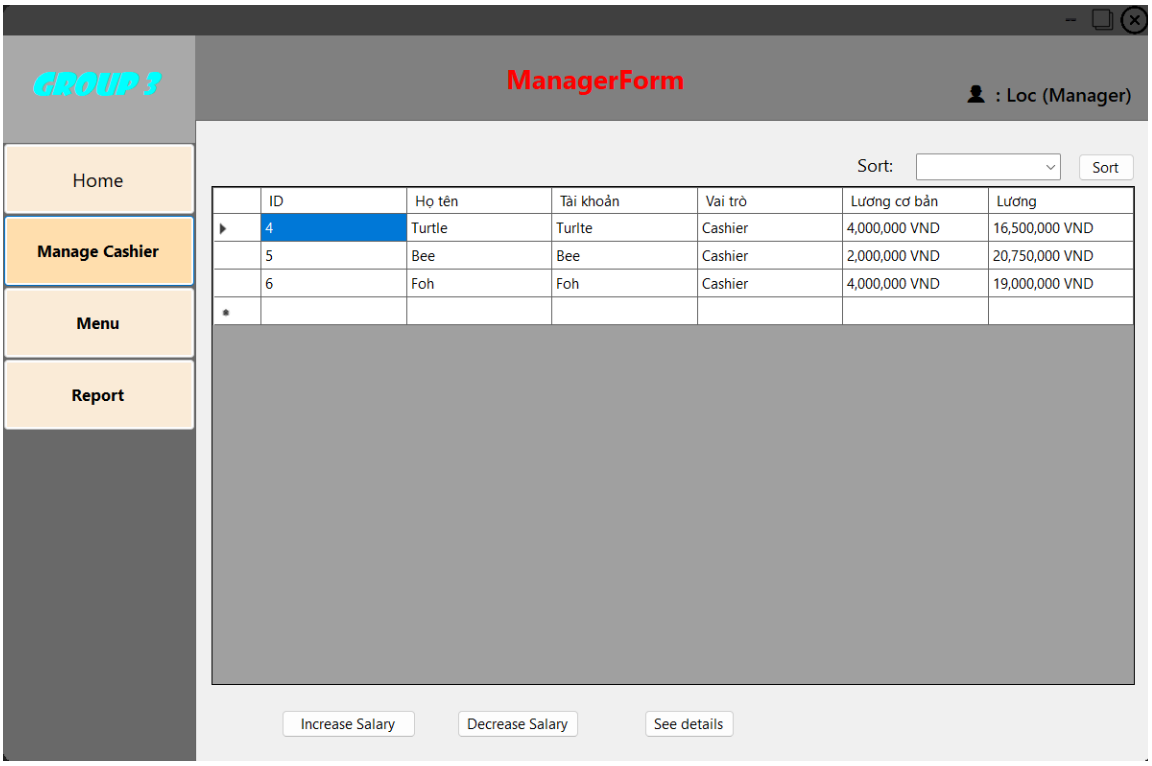
(1) Display: Product data is retrieved from ProductRepository by ProductService and displayed on DataGridView.

(2) Add/Edit/Delete: When Admin performs an operation (for example: press "Delete" button), ProductFormForManager appears and can be edited such as adding or

changing. Then the data will be transferred to the database and displayed on DataGridView.

4.4 Business Management (for Manager)

Description: This function allows Manager to perform all operations of increasing salary, decreasing salary, adding working hours,... in the See detail section for cashiers in the shop. Can be sorted by name or salary.



ID	Họ tên	Tài khoản	Vai trò	Lương cơ bản	Lương
4	Turtle	Turlte	Cashier	4,000,000 VND	16,500,000 VND
5	Bee	Bee	Cashier	2,000,000 VND	20,750,000 VND
6	Foh	Foh	Cashier	4,000,000 VND	19,000,000 VND

Figure 4.4: Manager Cashier interface.

Explanation

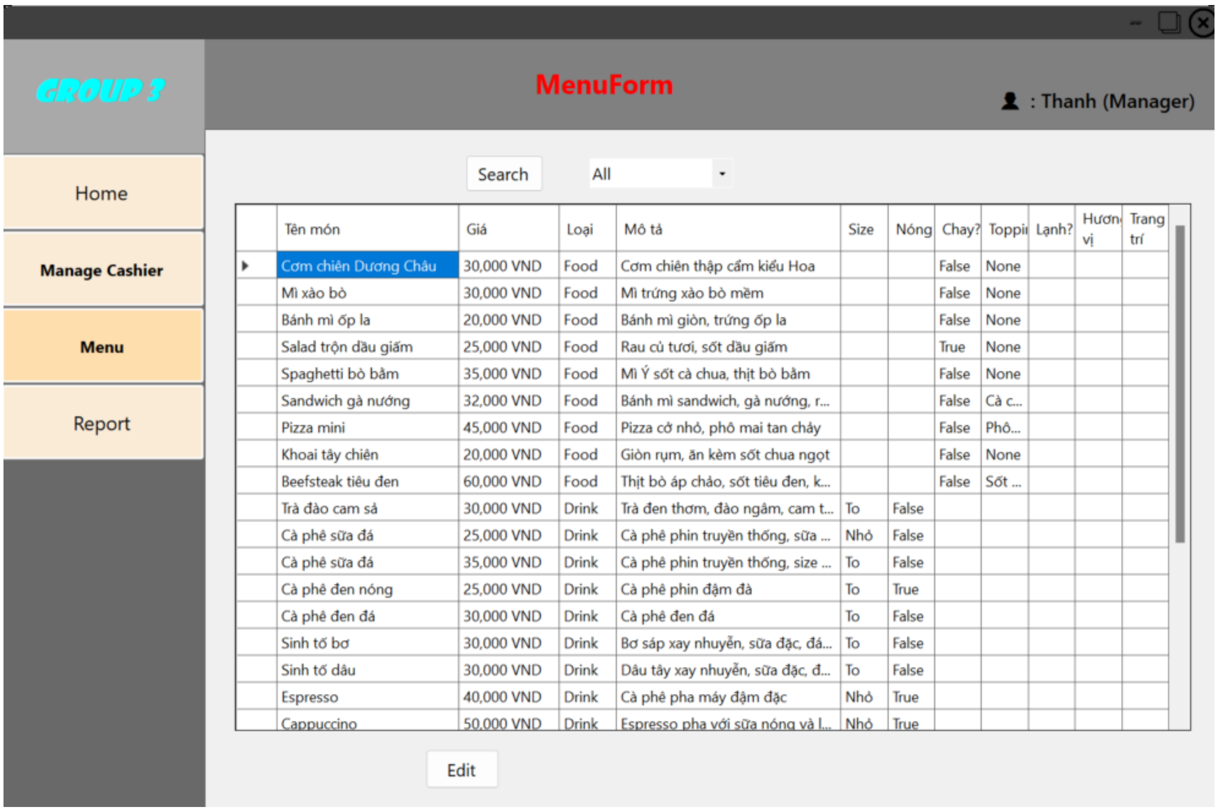
(1) Display: Cashier data is retrieved from EmployeeAccount via EmployeeService.GetManagableEmployees() and displayed in a DataGridView within the ManagerForm interface. Each row shows employee ID, name, username, role, base salary, and calculated salary. The salary is computed differently depending on the employee type (e.g. cashier includes working hours and hourly rate).

(2) Manage Salary and Details: The Manager can perform operations such as increasing or decreasing base salary by selecting an employee and entering an amount. These actions call IncreaseSalary() or DecreaseSalary() and update the database. Additionally, clicking the "View Details" button opens EmployeeDetailForm, where the

Manager can view and edit cashier-specific data like working hours and hourly rate. After confirmation, the updated data is saved and reflected in the main grid.

4.5 Menu Management (for Manager)

Description: This function allows users with the **Manager** role to **view and edit** existing menu items. Managers can update details of various product types such as food, drinks, and desserts, including attributes like size, toppings, flavor, and decoration. The interface is designed to be role-specific, ensuring that Managers can maintain menu accuracy without full administrative permissions.



Tên món	Giá	Loại	Mô tả	Size	Nóng	Chay?	Toppi	Lạnh?	Hương vị	Trang trí
Cơm chiên Dương Châu	30,000 VND	Food	Cơm chiên thập cẩm kiểu Hoa			False	None			
Mì xào bò	30,000 VND	Food	Mì trứng xào bò mềm			False	None			
Bánh mì ốp la	20,000 VND	Food	Bánh mì giòn, trứng ốp la			False	None			
Salad trộn dầu giấm	25,000 VND	Food	Rau củ tươi, sốt dầu giấm			True	None			
Spaghetti bò bằm	35,000 VND	Food	Mì Ý sốt cà chua, thịt bò bằm			False	None			
Sandwich gà nướng	32,000 VND	Food	Bánh mì sandwich, gà nướng, r...			False	Cà c...			
Pizza mini	45,000 VND	Food	Pizza cỡ nhỏ, phô mai tan chảy			False	Phô...			
Khoai tây chiên	20,000 VND	Food	Giòn rụm, ăn kèm sốt chua ngọt			False	None			
Beefsteak tiêu đen	60,000 VND	Food	Thịt bò áp chảo, sốt tiêu đen, k...			False	Sốt ...			
Trà đào cam sả	30,000 VND	Drink	Trà đen thơm, đào ngâm, cam t...	To	False					
Cà phê sữa đá	25,000 VND	Drink	Cà phê phin truyền thống, sữa ...	Nhỏ	False					
Cà phê sữa đá	35,000 VND	Drink	Cà phê phin truyền thống, size ...	To	False					
Cà phê đen nóng	25,000 VND	Drink	Cà phê phin đậm đà	To	True					
Cà phê đen đá	30,000 VND	Drink	Cà phê đen đá	To	False					
Sinh tố bơ	30,000 VND	Drink	Bơ sáp xay nhuyễn, sữa đặc, đá...	To	False					
Sinh tố dâu	30,000 VND	Drink	Dâu tây xay nhuyễn, sữa đặc, đ...	To	False					
Espresso	40,000 VND	Drink	Cà phê pha máy đậm đặc	Nhỏ	True					
Cappuccino	50,000 VND	Drink	Espresso pha với sữa nóng và l...	Nhỏ	True					

Figure 4.5: Menu management for Manager interface.

Explanation

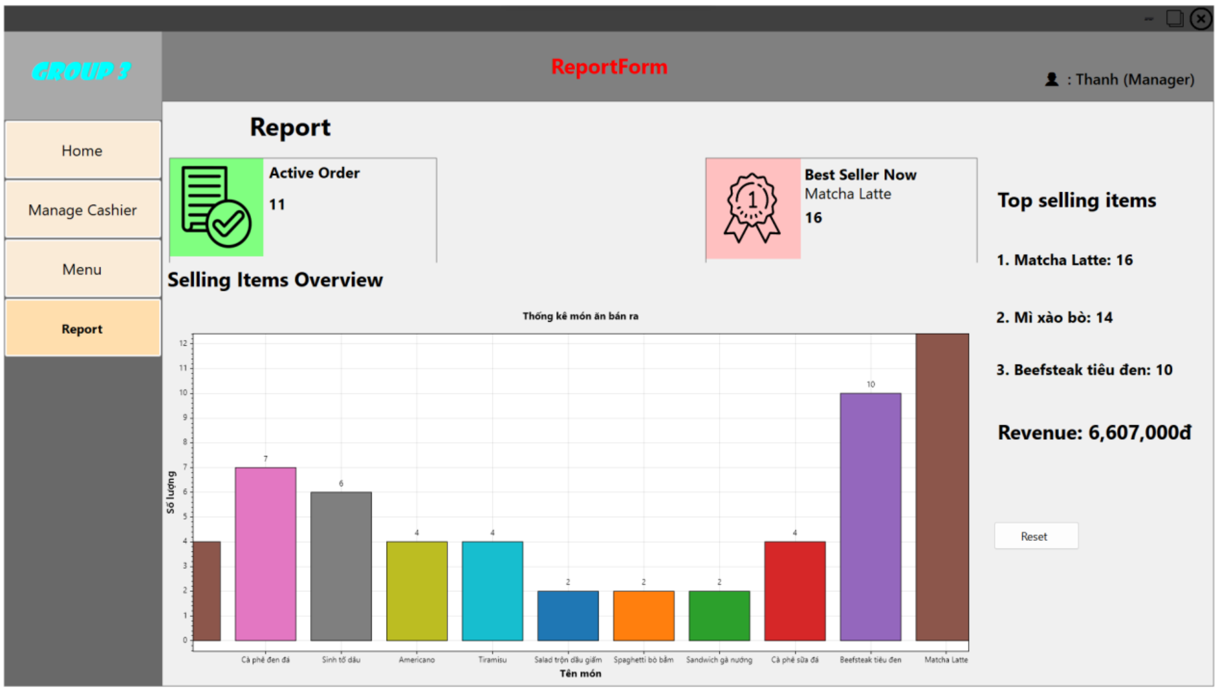
(1) Display: Menu data is retrieved from ProductRepository through ProductService and displayed in a DataGridView within the MenuFormForManager interface. The grid includes detailed columns such as item name, price, category, description, size, hot/cold status, vegetarian status, toppings, flavor, and decoration.

(2) Edit Only: Unlike the Admin interface, the Manager is restricted to **editing** existing menu items. When the Manager selects a row and clicks the "Edit" button, the system opens ProductFormForManager, pre-filled with the selected product's data. After

editing, the updated product is passed to ProductService.Update(), which updates both the database and the in-memory list. The grid is then refreshed to reflect the changes.

4.6 Report for product (for Manager)

Description: This function enables the **Manager** to generate a comprehensive business report based on completed orders. The report provides key performance indicators such as total number of orders, total revenue, best-selling product, and the top 3 most sold items. Additionally, a bar chart is rendered to visualize product sales distribution, helping managers make data-driven decisions.



Report

Active Order
11

Best Seller Now
Matcha Latte
16

Selling Items Overview

Top selling items

1. Matcha Latte: 16

2. Mì xào bò: 14

3. Beefsteak tiêu đen: 10

Revenue: 6,607,000đ

Figure 4.6: The report for Manager interface.

Explanation

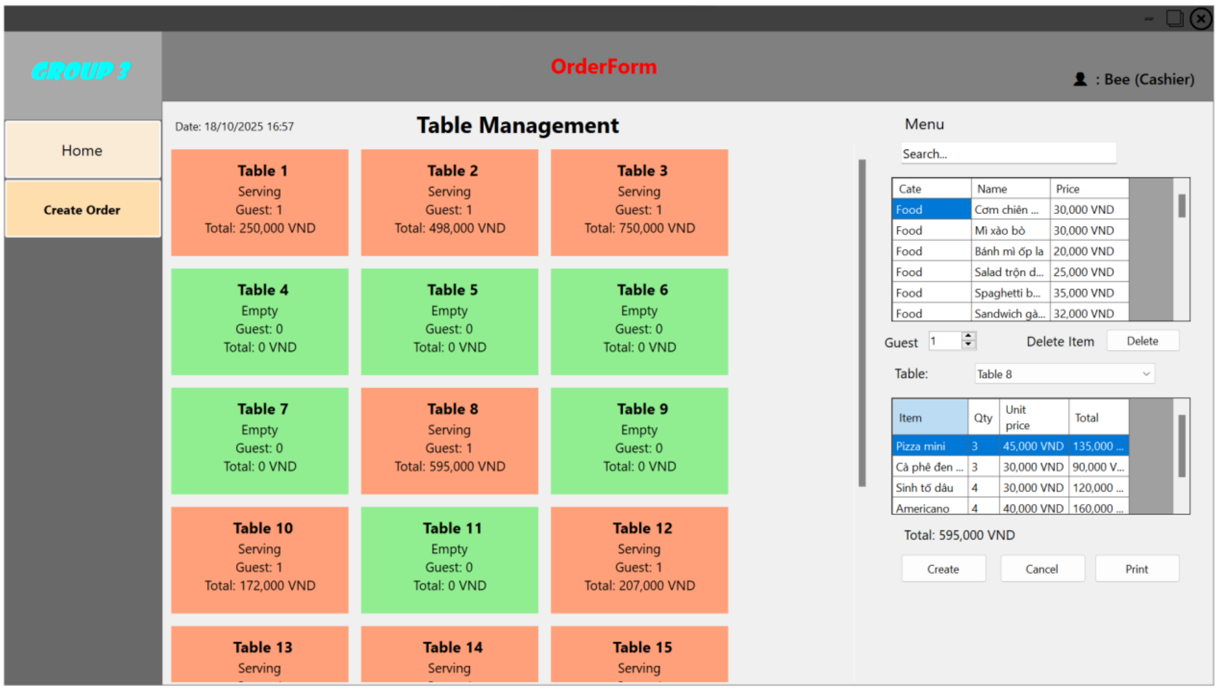
(1) Data: The ReportService class retrieves all completed orders from the OrderRepository and processes them to generate a Report object. This object includes: TotalOrders(Total number of order), TotalRevenue (Sum of all item prices across orders), BestSeller (The product with the highest quantity sold), TopSelling(A list of the top 3 best-selling items).

(2) Chart Visualization: The ChartData() method in ReportService transforms the report data into a dictionary of product names and their corresponding quantities. This data is passed to the RenderChart() method in ReportForm, which uses the ScottPlot

library to render a bar chart. Each bar represents a product, with height indicating quantity sold.

(3) Reset button: using The btnResetReport_Click() method in ReportForm allows the Manager to clear the current report and chart, resetting all fields to default values.

4.7 Cashier Business Process: Sales (for Cashier)


GROUP 7

OrderForm

: Bee (Cashier)

Home

Create Order

Date: 18/10/2025 16:57

Table Management

Table 1
Serving
Guest: 1
Total: 250,000 VND

Table 2
Serving
Guest: 1
Total: 498,000 VND

Table 3
Serving
Guest: 1
Total: 750,000 VND

Table 4
Empty
Guest: 0
Total: 0 VND

Table 5
Empty
Guest: 0
Total: 0 VND

Table 6
Empty
Guest: 0
Total: 0 VND

Table 7
Empty
Guest: 0
Total: 0 VND

Table 8
Serving
Guest: 1
Total: 595,000 VND

Table 9
Empty
Guest: 0
Total: 0 VND

Table 10
Serving
Guest: 1
Total: 172,000 VND

Table 11
Empty
Guest: 0
Total: 0 VND

Table 12
Serving
Guest: 1
Total: 207,000 VND

Table 13
Serving

Table 14
Serving

Table 15
Serving

Menu

Search...

Cate	Name	Price
Food	Cơm chiên ...	30,000 VND
Food	Mì xào bò	30,000 VND
Food	Bánh mì ốp la	20,000 VND
Food	Salad trộn d...	25,000 VND
Food	Spaghetti b...	35,000 VND
Food	Sandwich gà...	32,000 VND

Guest 1

Delete Item | Delete

Table: Table 8

Item	Qty	Unit price	Total
Pizza mini	3	45,000 VND	135,000 ...
Cà phê đen ...	3	30,000 VND	90,000 V...
Sinh tố dâu	4	30,000 VND	120,000 ...
Americano	4	40,000 VND	160,000 ...

Total: 595,000 VND

Create | Cancel | Print

Figure 4.7: The Order interface for Cashier.

Description: The Cashier Business Process is the core operational workflow of the Coffee Shop Management System. It enables cashiers to handle real-time customer orders, manage table assignments, add or remove items from orders, calculate totals, and finalize payments. This module is designed to be intuitive, responsive, and efficient.

The cashier begins by selecting a table from the visual layout in the OrderForm interface. Each table show its number, status (Empty or Serving), guest count, and current total.

When a table is clicked, the system either: Retrieves an existing active order for that table using OrderService.GetOrCreateOrder(), and then initializes a new Order object if none exists.

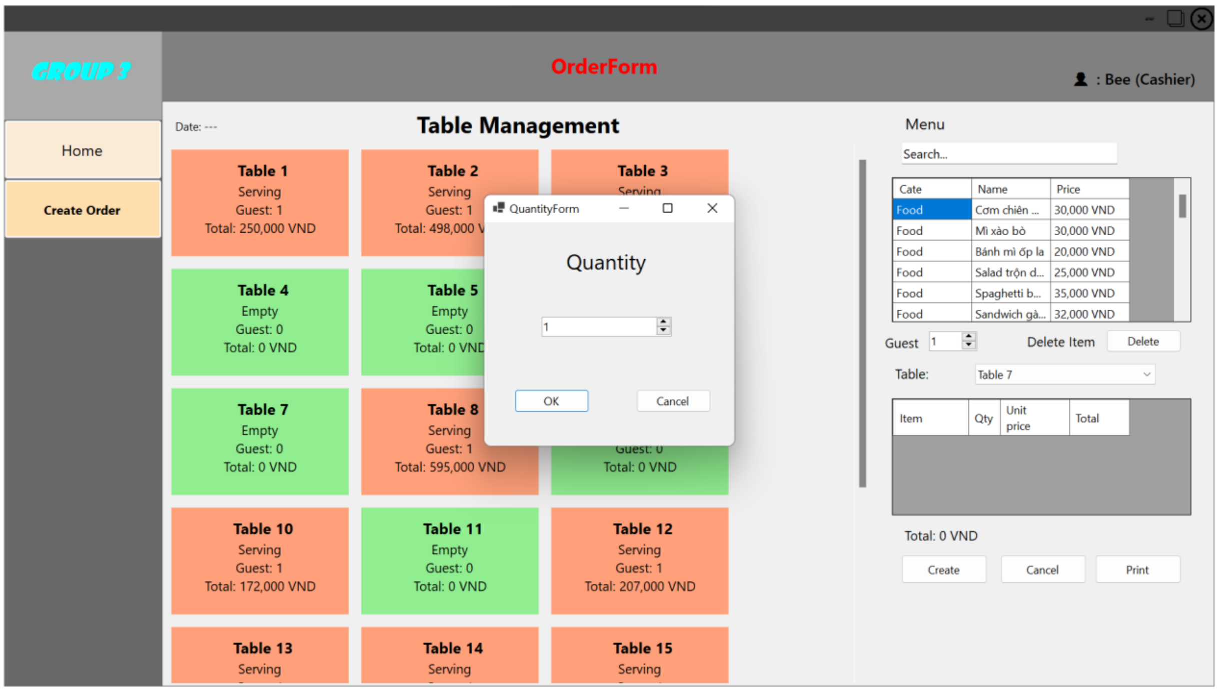


Table 1
Serving
Guest: 1
Total: 250,000 VND

Table 2
Serving
Guest: 1
Total: 498,000 V

Table 3
Serving

Table 4
Empty
Guest: 0
Total: 0 VND

Table 5
Empty
Guest: 0
Total: 0 VND

Table 7
Empty
Guest: 0
Total: 0 VND

Table 8
Serving
Guest: 1
Total: 595,000 VND

Guest: 0
Total: 0 VND

Table 10
Serving
Guest: 1
Total: 172,000 VND

Table 11
Empty
Guest: 0
Total: 0 VND

Table 12
Serving
Guest: 1
Total: 207,000 VND

Table 13
Serving

Table 14
Serving

Table 15
Serving

Cate	Name	Price
Food	Cơm chiên ...	30,000 VND
Food	Mì xào bò	30,000 VND
Food	Bánh mì ốp la	20,000 VND
Food	Salad trộn d...	25,000 VND
Food	Spaghetti b...	35,000 VND
Food	Sandwich gà...	32,000 VND

Guest 1 Delete Item Delete

Table: Table 7

Item	Qty	Unit price	Total

Total: 0 VND

Create Cancel Print

Figure 4.7.1: The quantity form when clicked choose item from Menu.

The full menu is displayed in a DataGridView, populated from ProductRepository via OrderService.GetMenu().

When the cashier clicks on a menu item: A QuantityForm popup appears, allowing the cashier to input the desired quantity and then the selected product and quantity are added to the current order using Order.AddItem().

Besides, the cashier can perform all basic tasks in managing tables, these actions represent the daily workflow of cashier in small to medium-sized food and beverage businesses: update guest count using a numeric input field, remove items using the Delete button and which call the function OrderService.Remove ItemFromOrder(), view total amount in real-time and calculated through Order.Total().

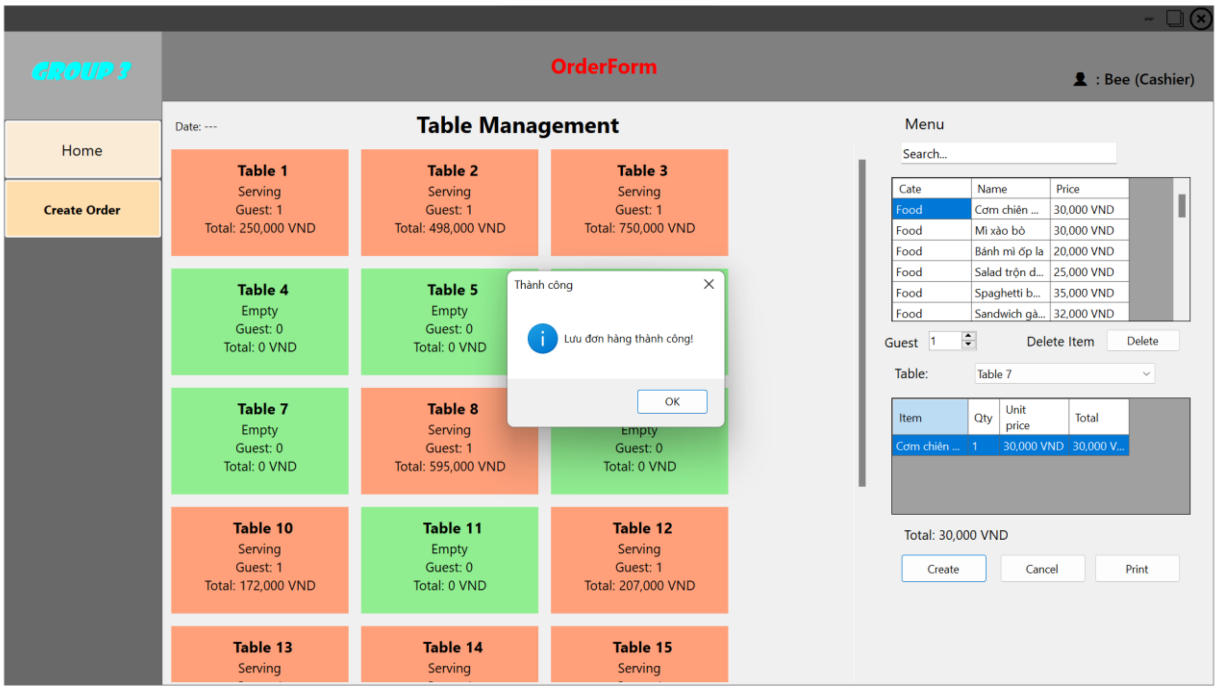


Table 1 Serving Guest: 1 Total: 250,000 VND

Table 2 Serving Guest: 1 Total: 498,000 VND

Table 3 Serving Guest: 1 Total: 750,000 VND

Table 4 Empty Guest: 0 Total: 0 VND

Table 5 Empty Guest: 0 Total: 0 VND

Table 7 Empty Guest: 0 Total: 0 VND

Table 8 Serving Guest: 1 Total: 595,000 VND

Empty Guest: 0 Total: 0 VND

Table 10 Serving Guest: 1 Total: 172,000 VND

Table 11 Empty Guest: 0 Total: 0 VND

Table 12 Serving Guest: 1 Total: 207,000 VND

Table 13 Serving

Table 14 Serving

Table 15 Serving

Thành công

Lưu đơn hàng thành công!

Cate	Name	Price
Food	Cơm chiên ...	30,000 VND
Food	Mì xào bò	30,000 VND
Food	Bánh mì ốp la	20,000 VND
Food	Salad trộn d...	25,000 VND
Food	Spaghetti b...	35,000 VND
Food	Sandwich gà...	32,000 VND

Guest 1

Table: Table 7

Item	Qty	Unit price	Total
Cơm chiên ...	1	30,000 VND	30,000 V...

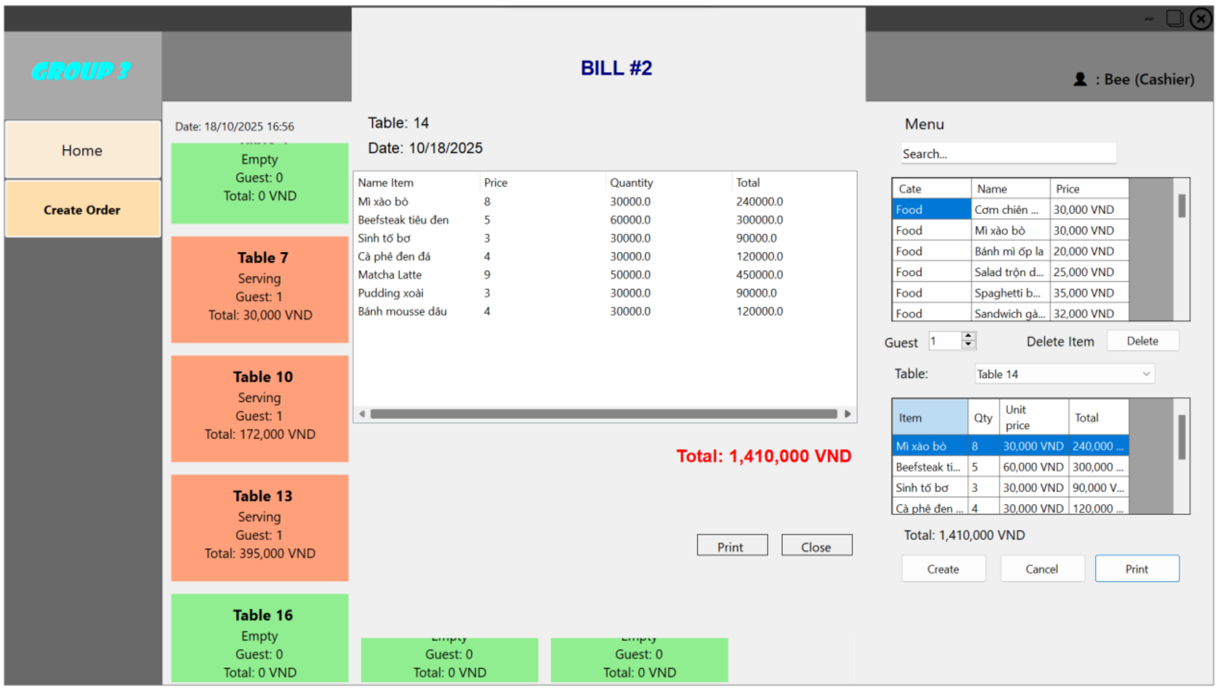
Total: 30,000 VND

Figure 4.7.2: The quantity form when clicked choose item from Menu.

Saving and Implement the Order: When the cashier finishes adding items and setting the guest count, they click the "Create Order" button (btncreate). This action saves the order to the database via OrderRepository.Save(), updates the table status to "Occupied", and displays a confirmation message to indicate the order has been successfully recorded.

Finalizing Payment and Printing Bill: Once the customer is ready to pay, the cashier clicks the "Print Bill" button (btnprint). The system then calls OrderService.FinalizePaymentForOrder() to generate a Bill object, opens a BillForm to display the receipt, and deletes the order from the database using OrderRepository.Delete(). Finally, the table status is reset to "Empty", and the order panel is cleared to prepare for the next customer.

4.8 Bill Generation and Receipt Printing



BILL #2

Table: 14

Date: 10/18/2025

Name Item	Price	Quantity	Total
Mì xào bò	8	30000.0	240000.0
Beefsteak tiêu đen	5	60000.0	300000.0
Sinh tố bơ	3	30000.0	90000.0
Cà phê đen đá	4	30000.0	120000.0
Matcha Latte	9	50000.0	450000.0
Pudding xoài	3	30000.0	90000.0
Bánh mousse dâu	4	30000.0	120000.0

Total: 1,410,000 VND

Menu

Search...

Cate	Name	Price
Food	Cơm chiên ...	30,000 VND
Food	Mì xào bò	30,000 VND
Food	Bánh mì ốp la	20,000 VND
Food	Salad trộn d...	25,000 VND
Food	Spaghetti b...	35,000 VND
Food	Sandwich gà...	32,000 VND

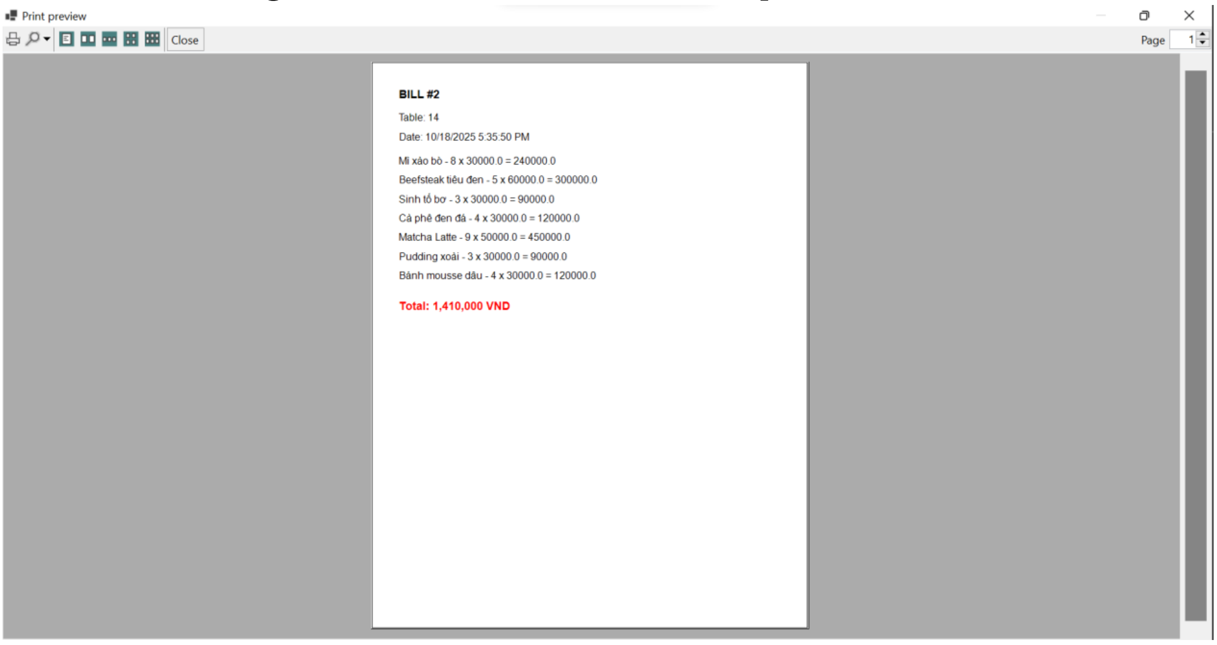
Guest 1 Delete Item

Table: Table 14

Item	Qty	Unit price	Total
Mì xào bò	8	30,000 VND	240,000 ...
Beefsteak ti...	5	60,000 VND	300,000 ...
Sinh tố bơ	3	30,000 VND	90,000 V...
Cà phê đen ...	4	30,000 VND	120,000 ...

Total: 1,410,000 VND

Figure 4.8: Bill Generation after completed Order.



BILL #2

Table: 14

Date: 10/18/2025 5:35:50 PM

Mì xào bò - 8 x 30000.0 = 240000.0

Beefsteak tiêu đen - 5 x 60000.0 = 300000.0

Sinh tố bơ - 3 x 30000.0 = 90000.0

Cà phê đen đá - 4 x 30000.0 = 120000.0

Matcha Latte - 9 x 50000.0 = 450000.0

Pudding xoài - 3 x 30000.0 = 90000.0

Bánh mousse dâu - 4 x 30000.0 = 120000.0

Total: 1,410,000 VND

Figure 4.8.1: Print Bill preview to finish.

Description: This module handles the final step of the customer transaction: generating a bill and printing a receipt. It transforms a completed order into a structured invoice, displays it in a dedicated form, and allows the cashier to preview and print the receipt. The bill includes essential details such as table number, date, itemized list of products, quantities, unit prices, and total amount.

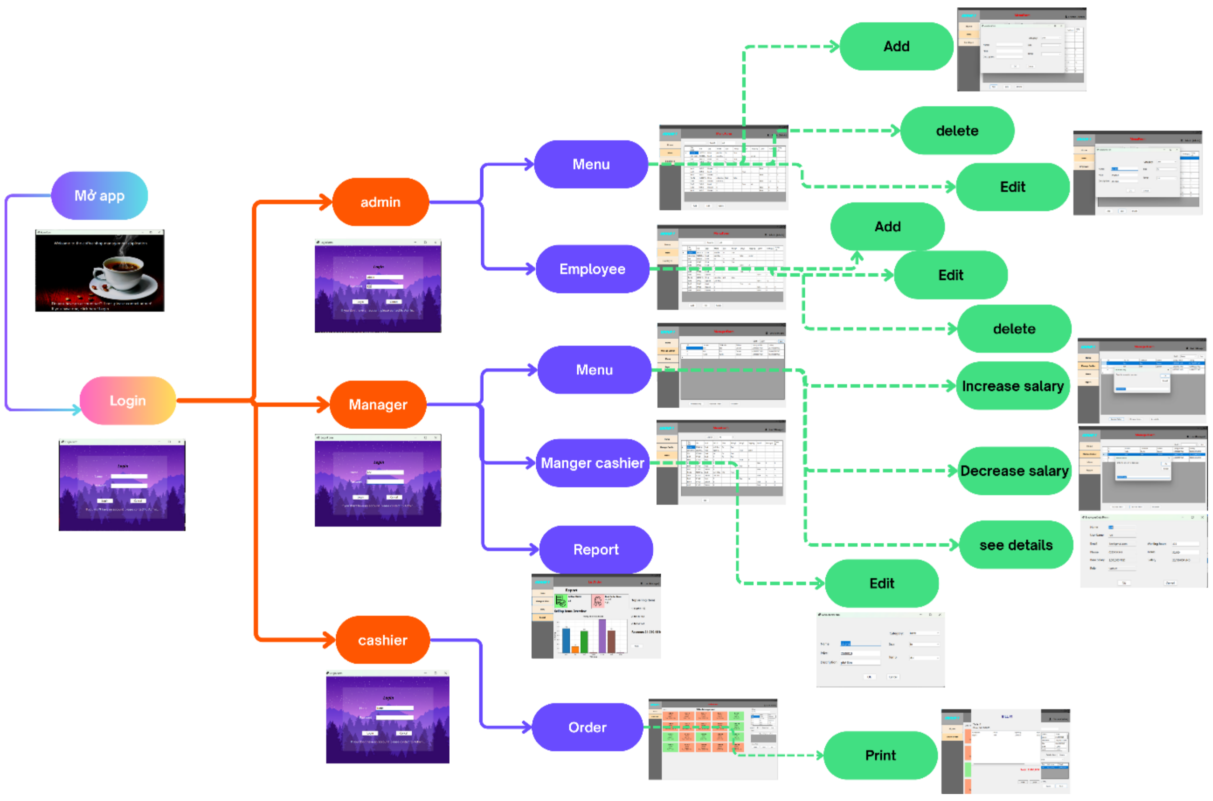
Explanation

(1) Bill Creation After the cashier confirms payment, the system calls OrderService.FinalizePaymentForOrder() to generate a Bill object. This object captures the order's table number, datetime, item list, and total amount. The bill ID is auto-incremented using a static counter in the Bill class.

(2) Printing the Receipt When the cashier clicks the Print button, the system triggers printDocument1_PrintPage() to render the receipt using GDI+ graphics. The receipt is formatted with headers, item rows, and total amount, and shown in a print preview dialog (printPreviewDialog1). The cashier can then confirm and send the receipt to the printer.

Post-Payment Cleanup After printing, the order is removed from the database using OrderRepository.Delete(), and the table status is reset to "Empty" for the next customer.

5. Simulate how the app works



If you want to see all picture, you get link https://www.canva.com/design/

If you want to repo project in github or to see implementation process, you can access link https://github.com/Diablo-Loc/coffee-management

6. Conclusion

The project has successfully built a Coffee Shop Management System using C# language and Windows Forms platform, fully completing the proposed business goals. The core success of the project lies not only in completing functions such as sales, authorization or reporting, but also in thoroughly applying modern software design principles. By strictly adhering to the 3-layer architecture (UI-BLL-DAL), the system has achieved a clear separation of responsibilities, making the source code scientific and easy to maintain. Object Oriented Principles (OOP) have been practically applied: Inheritance helps reuse code through logical family trees (such as Person and Product), Polymorphism brings flexibility to core operations (such as CalculateSalary(),..), and Encapsulation (through Service classes) effectively hides complexity, ensuring data integrity. Although the system is currently limited to the WinForms platform, this solid architectural foundation has laid a very good foundation, allowing the project to be easily upgraded and expanded in the future. Later, when we learn more new subjects, we will add more and the app will be more complete.

7. Job assignment board

No.	Task	Description(include class.cs)	Reponsible	Complete(%)
1	Catalog	Include: Product, Food, Dessert, Drink	Lê Tuấn Thành	100%
2	OrderModel	Include: Order	Lê Tuấn Thành	100%
3	OrderModel	Include: OrderItem	Nguyễn Đăk Lộc	100%
4	PersonModel	Include: Person, Employee, Adminstrator, Manager, Cashier	Nguyễn Đăk Lộc	100%
5	Other Class	Bill, Report	Lê Tuấn Thành	100%
6	Other Class	EntityBase	Nguyễn Đăk Lộc	100%
7	Common (Interface,helper)	Include: IsalaryCompute, MoneyHelper	Nguyễn Đăk Lộc	100%
8	Common (Interface,helper)	EmployeeCompare	Lê Tuấn Thành	100%
9	UI	Include: HomeForm, MenuForm, MenuFormForManager, BillForm, QuantityForm, Report, OrderForm,	Lê Tuấn Thành*	100%
10	UI for Login	Include: AddEmployeeForm, AdminManage, LoginForm	Nguyễn Đăk Lộc*	100%
11	UI	Include: MainForm, ManagerForm, ProductForm, ProductFormForManager, EmployeeDetailForm,	Nguyễn Đăk Lộc*	100%
12	Services foulder	Include: AdminService, AuthService, EmployeeService, OrderService, ProductService, ReportService	Both	100%
13	Data foulder	Database SQLite	Both*	100%
13	OOP design	Design class diagram of the system	Nguyễn Đăk Lộc	100%

Note: *Items marked with (*) are completed with consultation from documents, Youtube and AI tools to learn and optimize design solutions.*

8.References

[1]. Modern Flat UI, Random MultiColor, Highlight button-Active Form, WinForm, C#, V-0.1. Truy cập từ https://youtu.be/

[2]. Lập Trình VB. (không rõ ngày). *Lập trình thêm, xóa, sửa, tìm kiếm với Sqlite in C#*. Truy cập từ https://laptrinhvb.net/...

[3]. Microsoft. (2024). *Sử dụng SQLite với .NET (Use SQLite with .NET)*. Microsoft Docs. Truy cập từ https://learn.microsoft.com/en-us/