PROJECT TITLE:

FAST FOOD RESTAURANT MANAGEMENT SYSTEM

Group Members:

Hania Khan, Hamna Asif, & Maha Tariq

Submitted to:

Mr. Irfan Hameed

Submitted on:

20, May 2026

Table of Contents

1 Project Overview:

HHM is a data-driven web-based restaurant management system built using ASP.NET (VB.NET) with a SQL Server backend. It allows customers to browse a menu, place orders, and make payments online, while restaurant staff manage orders, inventory, employees, and finances through role-specific dashboards.

1.1 Objectives:

- To develop an online food ordering system
- To implement secure customer and staff authentication
- To manage restaurant orders digitally
- To provide role-based access control
- To implement shopping cart functionality
- To ensure database-driven dynamic content
- To host the application online

1.2 Proposed Features

The HHM Food Management System is designed to simplify restaurant operations by providing separate functionalities for customers and staff. It automates food ordering, payment processing, employee management, and order tracking through an easy-to-use web interface.

1.2.1 Customer Features:

Customer Registration and Login: Customers can create accounts and securely log in to access ordering features and manage their activity.

Food Browsing and Categories: Users can explore food categories such as Burgers, Pizza, Broast, and Shawarma, with items displayed along with prices for easy selection.

Search Functionality: A search feature allows customers to quickly find food items using keywords. Users can access search even if they aren’t logged in.

Shopping Cart Management: Customers can add items to their cart, update quantities, and view a dynamically calculated total bill.

Secure Checkout and Payment: Multiple payment options such as Cash, Card, and Wallet are available. Customers can review and confirm orders securely.

Order Placement and Tracking: Orders are stored in the database and forwarded for processing, ensuring accurate handling and tracking.

1.2.2 Staff Features:

Staff Login Portal: Authorized employees access the system through a secure staff login.

Role-Based Access: Dashboards are assigned based on roles like Manager, Chef, Cashier, Waiter, and Inventory Staff.

Manager Dashboard: Managers can oversee operations, employee records, and financial data from a centralized dashboard.

Employee Management: Managers can add, update, and manage employee details and roles.

Finance Monitoring: The system tracks revenue and payments to support financial management.

Order Monitoring: Staff can view and process customer orders efficiently after payment confirmation.

1.2.3 System Features:

Database-Driven System: All data including users, orders, and payments are managed using SQL Server.

Session-Based Authentication: User sessions ensure secure login and access control.

Dynamic Cart Updates: UpdatePanel enables smooth, partial page updates for better user experience.

Secure Queries: Parameterized ADO.NET queries help prevent SQL injection attacks.

User-Friendly Interface: A simple and responsive design improves usability for both customers and staff.

Centralized Management: The system integrates ordering, payments, employee management, and monitoring into one platform for efficient restaurant operations.

1.3 Technologies Used

The HHM Food Management System was developed using modern web development and database technologies. These technologies helped in building a dynamic, secure, and data-driven application.21

Technology	Purpose
ASP.NET Web Forms	Developing web pages and application structure
VB.NET	Backend server-side programming
SQL Server	Database management and storage
HTML	Structure of web pages
CSS	Styling and user interface design
ADO.NET	Database connectivity and query execution
Session & ViewState	State management
IIS / Somee Hosting	Application deployment and hosting

The combination of these technologies allowed the system to handle customer orders, employee management, payment processing, and database operations efficiently.

1.4 Data-Driven Nature

The HHM Food Management System is a fully data-driven web application because all important information is stored and managed through a SQL Server database instead of hardcoded values.

The system dynamically retrieves and updates data such as:

- Customer accounts
- Employee records
- Roles and permissions

- Menu items and food categories
- Orders and order details
- Shopping cart information
- Payment records

For example, menu items are loaded directly from the database using SQL queries, and customer orders are stored in related tables such as Order and OrderLine. Employee roles are also managed using separate Role and Employee tables, allowing role-based access to different dashboards.

The application uses ADO.NET with parameterized SQL queries to communicate with the database securely and efficiently. This approach makes the system flexible, scalable, and easier to maintain because any changes in data are automatically reflected in the application.

1.5 State Management

We used State Management to maintain user data and application data between different page requests of users. We used three major state management techniques in our project:

1. Session
2. ViewState
3. Query Strings

Technique	**Usage**
Session	This stores the information of logged in user
ViewState	This maintains employee table in manager dashboard
Query String	This is used to pass OrderId to Payment page

1.5.1 Session State:

We used Session State to store information about the currently logged-in user. The system remembers the user identity and role while navigating between pages. It also identifies whether the current user is a customer or employee. Furthermore, it stores employee role for role-based redirection.

Examples of usage in our code:

Example from Staff Login:

```
Session("UserType") = "Employee"
Session("UserID") = dr("employee_id")
Session("UserName") = dr("employee_name")
Session("RoleID") = roleID
Session("IsAuthenticated") = True
```

Example from Customer Login:

```
Session("UserType") = "Customer"
Session("UserID") = dr("customer_id")
Session("UserName") = dr("name")
```

Example of Session Validation

```
If Session("RoleID") Is Nothing OrElse CInt(Session("RoleID")) <> 1 Then
    Response.Redirect("StaffLogin.aspx")
    Exit Sub
End If
```

1.5.2 ViewState:

We used ViewState in the Manager part so that we can preserve employee table data during postbacks. Since ASP.NET Web Forms reloads the page after every server event, ViewState retains the DataTable contents. It lets us maintain GridView contents without having to reload it from the database again.

Examples of usage in our code:

Example from Manager.aspx.vb:

```
Private Property EmpTable As DataTable
    Get
        If ViewState("EmpTable") Is Nothing Then
            Dim dt As New DataTable()
            dt.Columns.Add("Name")
            dt.Columns.Add("Username")
            dt.Columns.Add("Role")
            ViewState("EmpTable") = dt
        End If
        Return CType(ViewState("EmpTable"), DataTable)
    End Get
    Set(value As DataTable)
        ViewState("EmpTable") = value
    End Set
End Property
```

Adding Data into ViewState Table:

```
Dim dt = EmpTable
dt.Rows.Add(txtName.Text, txtUser.Text, ddlRole.SelectedValue)
EmpTable = dt
gvEmployees.DataSource = dt
gvEmployees.DataBind()
```

1.5.3 Query Strings:

We used Query Strings to transfer small pieces of data through the URL between pages. In our project, Query Strings were mainly used for passing the OrderId from the Cart page to the Payment page. It also helps load correct payment summary.

Example of usage in our code:

Example from Cart.aspx.vb:

```
Response.Redirect("Payment.aspx?OrderId=" & orderId)
```

Reading Query String in Payment.aspx.vb:

```
Dim orderId As String = Request.QueryString("OrderId")
```

1.6 Data Security:

Following security features were implemented:

- User authentication is implemented for both customers and staff members.
- Session management is used to prevent unauthorized access to protected pages.
- Role-based authorization restricts access according to employee roles.
- SQL Injection attacks are prevented using parameterized SQL queries.
- Validation checks are performed before database operations.

1.6.1 Authentication:

It is used to verify the identity of customers and staff members before granting access to the system. Users must login with valid credentials. Session variables are created after successful login to maintain user identity throughout the application.

Example:

Session("UserID") = dr("customer_id")

1.6.2 Authorization:

Authorization is implemented using role-based access control. Different dashboards are accessible according to roles.

Example:

If Session("RoleID") <> Cashier Then Response.Redirect("StaffLogin.aspx") End If

1.6.3 SQL Injection Prevention:

Parameterized SQL queries are used to prevent SQL Injection attacks

Example:

cmd.Parameters.AddWithValue("@email", email)

1.7 Users and Role based Functionalities:

1.7.1 Roles:

We implemented the following roles in our db:

Role ID	Role Name
1	Manager
2	Chef

3	Cashier
4	Waiter
5	Inventory Manager

These roles are stored in the database using the `Role` table and are linked with employees for role-based access control.

1.7.2 Functionalities:

The system provides functionalities according to each user's roles

User Role	Functionalities
Customer	Register account, login, browse menu, search food, manage cart, place orders, make payments
Manager	Manage employees, access finance dashboard, monitor system operations
Chef	View and process kitchen orders
Cashier	Handle order verification and payment management
Inventory Manager	Manage food inventory and item availability

2 Software Requirement Specification:

2.1 Functional Requirements

The Fast Food Management System (FFMS) shall provide the following functional requirements:

2.1.1 Customer Module

1. The system shall allow customers to register using name, email, and password.
2. The system shall allow customers to log in securely.
3. The system shall display food categories such as Burgers, Pizza, Broast, and Shawarma.
4. The system shall allow customers to search food items.

5. The system shall allow customers to add items to the shopping cart.
6. The system shall allow customers to update item quantity in the cart.
7. The system shall allow customers to remove items from the cart.
8. The system shall calculate the total bill automatically.
9. The system shall provide multiple payment methods such as Cash, Card, and Wallet.
10. The system shall generate and save customer orders.

2.1.2 Staff Module

1. The system shall allow employees to log in using staff credentials.
2. The system shall identify employee roles such as Manager, Chef, Cashier, and Inventory Manager.
3. The system shall redirect employees to their respective dashboards.
4. The manager shall be able to add new employees.
5. The manager shall be able to update employee roles and salaries.
6. The system shall maintain employee records.

2.1.3 Order Management Module

1. The system shall create a pending order when a customer adds an item to the cart.
2. The system shall update order status after payment.
3. The system shall store order details including quantity, amount, and payment type.
4. The system shall maintain transaction records for verification.

2.2 Non-Functional Requirements

2.2.1 Performance Requirements

1. The system should load pages within 2–3 seconds.
2. The system should support multiple users simultaneously.

2.2.2 Security Requirements

1. User authentication shall be required for login.
2. Session management shall be used to protect user data.
3. SQL queries shall use parameterized commands to reduce SQL injection risks.

2.2.3 Reliability Requirements

1. The system should maintain accurate order and payment records.
2. The database should provide consistent data storage.

2.2.4 Usability Requirements

1. The system shall provide a simple and user-friendly interface.
2. Navigation between pages should be easy and responsive.

2.2.5 Maintainability Requirements

1. The system should be modular for future updates.
2. New menu categories and payment methods should be easily added.

2.2.6 Availability Requirements

1. The system should be accessible 24/7 with internet connectivity.

3 Analysis and Design Modeling

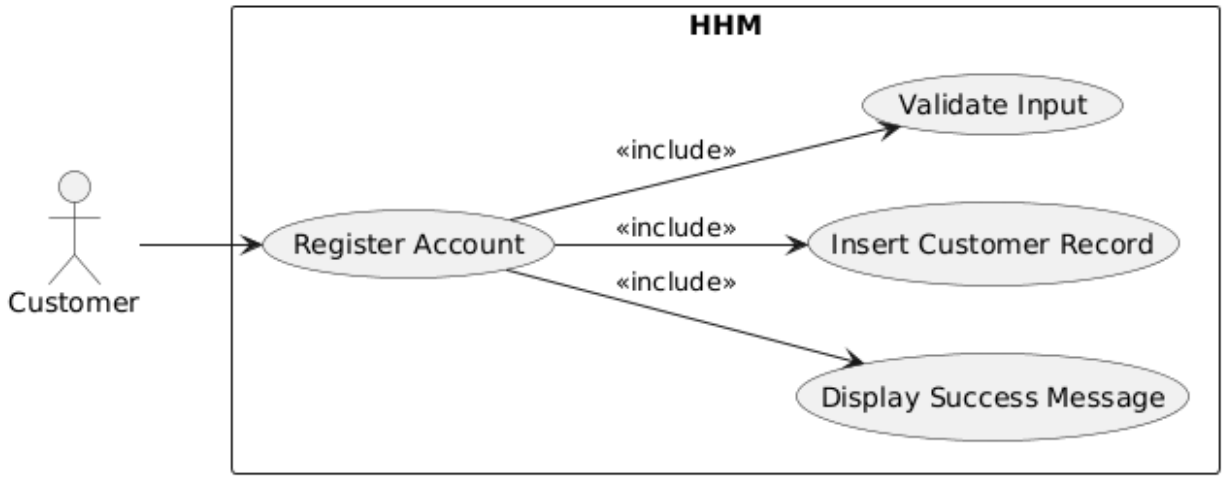
3.1 Use Cases

3.1.1 Use Case 1 - Customer Registration

Actor: Customer
Precondition: User is not registered

Steps:

1. Customer navigates to the Registration page
2. Enters name, email, mobile, password, and confirm password
3. System validates that all fields are filled → system inserts record into the Customer table → success message displayed.



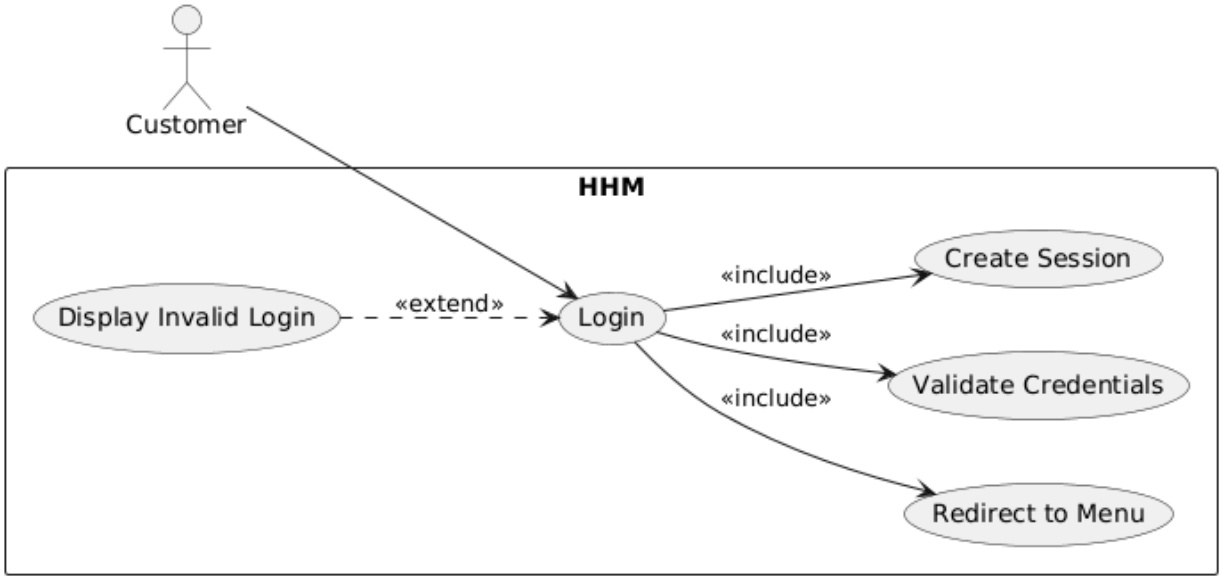
3.1.2 Use Case 2 - Customer Login

Actor: Customer

Precondition: Customer has a registered account

Steps:

1. Customer opens Login page → enters email and password
2. System queries the Customer table using parameterized SQL → on match, system sets Session("UserType"), Session("UserID"), Session("UserName") → customer is redirected to Menu.aspx.
3. Exception: Invalid credentials → "Invalid Customer email or password" shown.



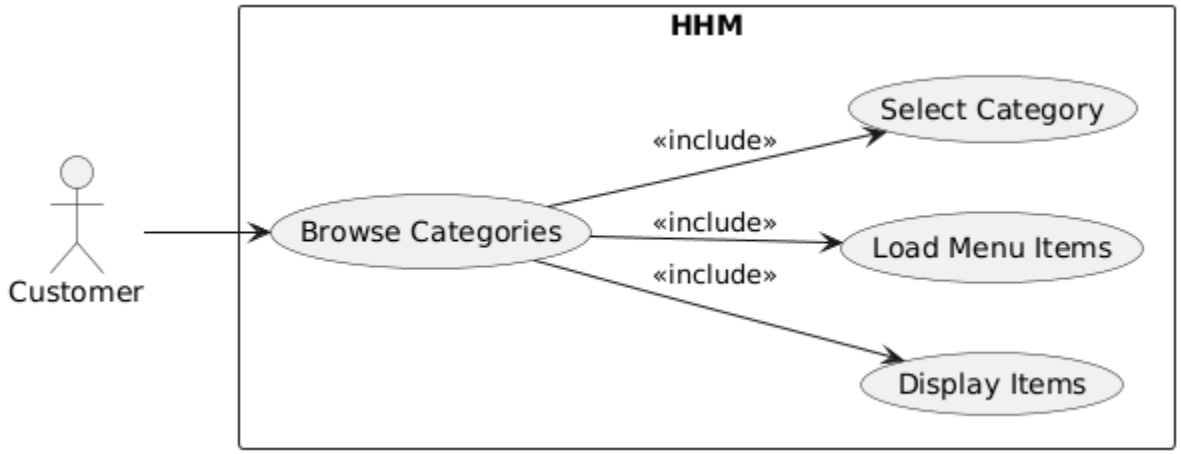
3.1.3 Use Case 3 - Browse Menu Categories

Actor: Customer

Precondition: Customer is on Menu.aspx

Steps:

1. Customer views the four category cards (Burgers, Pizza, Broast, Shawarma)
2. Selects a category → system loads MenuItem table filtered by category and availability=1
3. Items displayed via Repeater with name, price, and Add to Cart button.

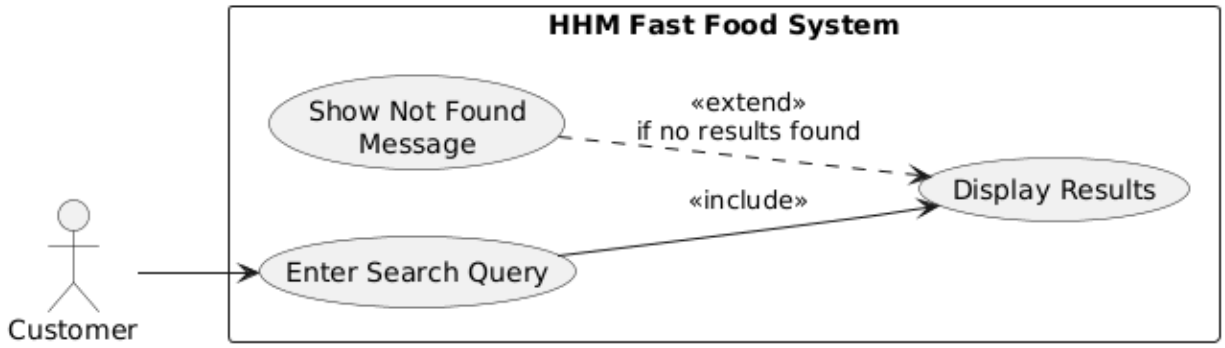


3.1.4 Use Case 4 - Search Food Items

Actor: Customer

Steps:

1. Customer enters in search box → clicks search icon
2. System redirects to SearchResults.aspx with query string → system queries MenuItem table using LIKE on product_name
3. Matching items will be displayed; if none found, "Sorry, this item was not found!" panel shown.



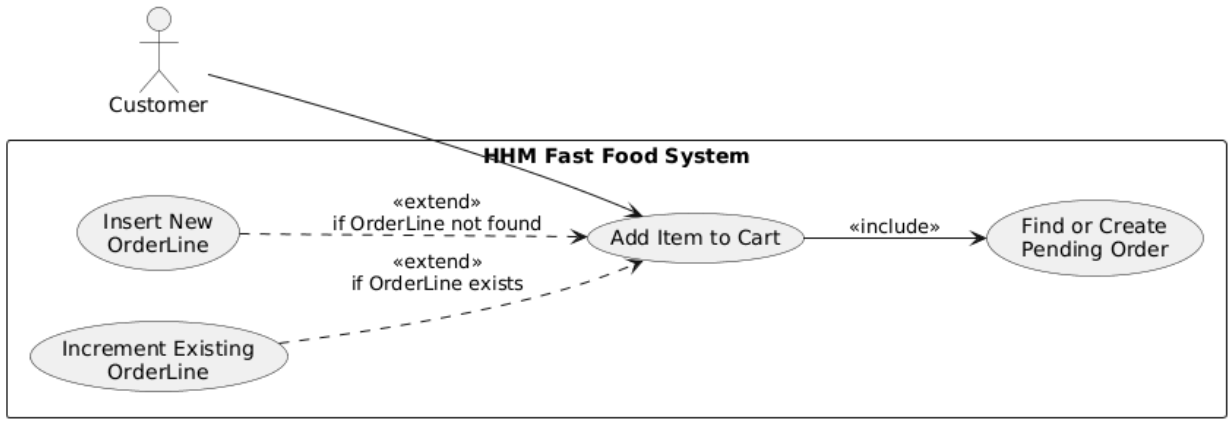
3.1.5 Use Case 5 - Add Item to Cart

Actor: Customer

Precondition: Customer is logged in

Steps:

1. Customer selects quantity and clicks Add to Cart
2. System checks for an existing Pending order for this customer → if none, system creates a new Order record with status 'Pending'
3. System checks if OrderLine already exists for this product → if yes, increments quantity; if no, inserts new OrderLine → success message shown inside UpdatePanel without full page reload.



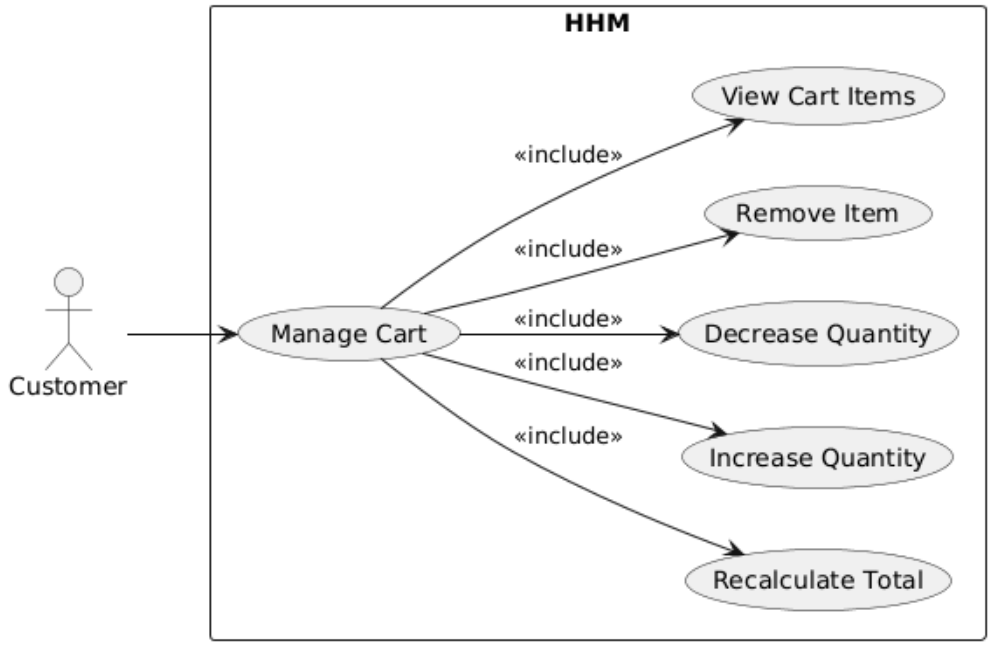
3.1.6 Use Case 6 - Manage Cart (Update / Remove)

Actor: Customer

Precondition: Customer has items in cart

Steps:

1. Customer opens Cart.aspx → views all OrderLine items joined with MenuItem
2. Clicks + to increment quantity or – to decrement
3. If quantity reaches 1 and – is clicked, item is deleted automatically
4. Customer can also click Remove to delete directly → total bill recalculates after every change.



3.1.7 Use Case 7 - Checkout and Payment

Actor: Customer

PreCondition: Cart has at least one item.

Steps:

1. Customer clicks "Proceed to Payment"
2. System retrieves Pending order_id and redirects to Payment.aspx?OrderId=X
3. System calculates total from OrderLine × MenuItem price
4. Customer selects Cash, Card, or Wallet → clicks Confirm → system begins a SQL Transaction
5. Updates OrderLine unit_price and line_total, updates Order total_amount, inserts Payment record with status 'Paid', updates Order status to 'Verification' → transaction committed → success alert shown.
6. Exception: If no payment method selected or order data missing, error message shown; on DB failure, transaction is rolled back

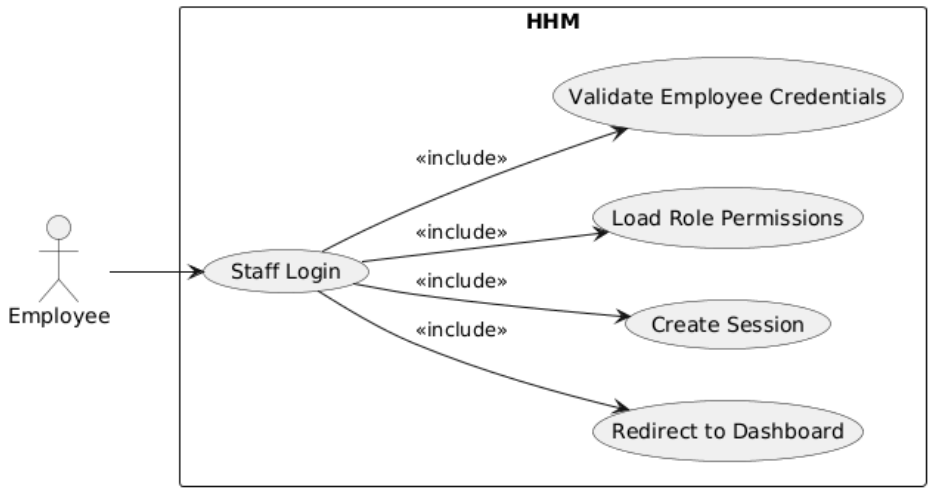
3.1.8 Use Case 8 - Staff Login

Actor: Employee

Precondition: Employee credentials exist in the Employee table

Steps:

1. Employee opens StaffLogin.aspx → enters email and password
2. System joins Employee and RolePermissions tables → on match, Session variables are set including RoleID → system redirects to role-specific dashboard (Manager=1, Chef=2, Cashier=3, else Inventory).

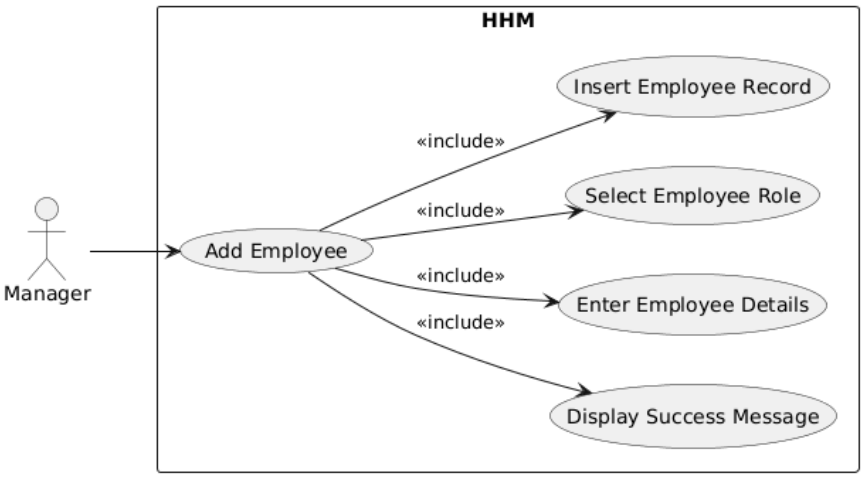


3.1.9 Use Case 9 - Manager: Add New Employee

Actor: Manager

Steps:

1. Manager navigates to ManagerAddEmployee.aspx
2. Enters employee name, email, password, salary, and selects role from dropdown
3. System inserts into Employee table with GETDATE() as hire_date → form is cleared on success.

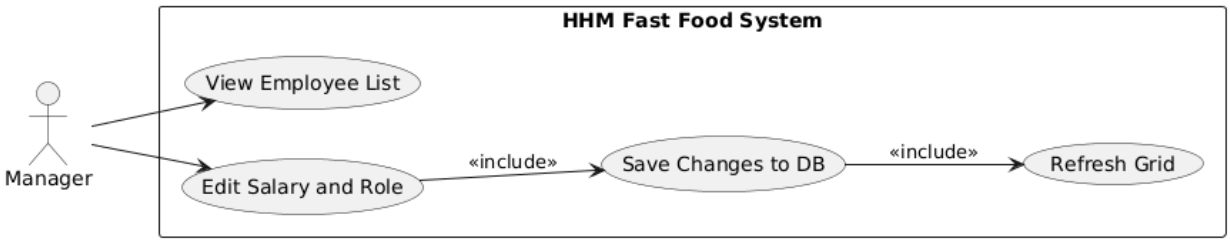


3.1.10 Use Case 10 - Manager: Edit Employee Salary and Role

Actor: Manager

Steps:

1. Manager opens ManagerEmployees.aspx
2. GridView loads all employees from DB
3. Manager enters new salary and selects new role for a specific row
4. Clicks Save → system executes UPDATE Employee SET salary, role_id WHERE employee_id
5. grid refreshes with updated values.

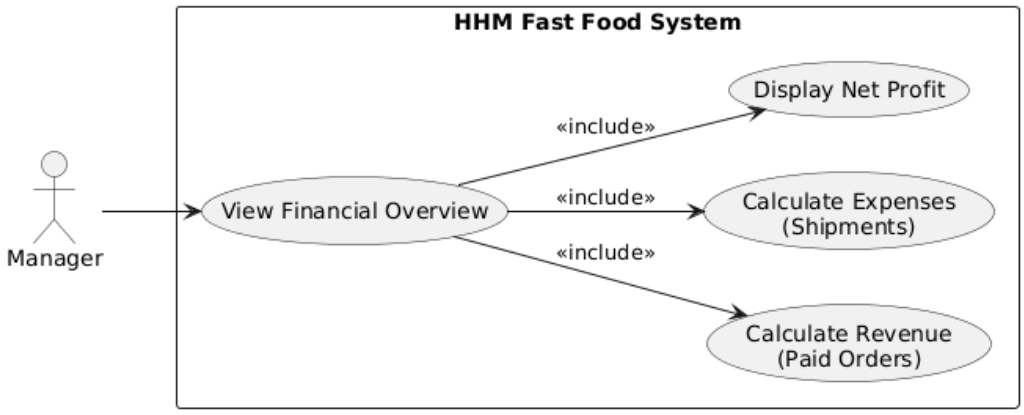


3.1.11 Use Case 11 - Manager: View Financial Overview

Actor: Manager

Steps:

1. Manager opens ManagerFinance.aspx
2. System queries total revenue (SUM of Paid orders via Payment join) and total expenses (SUM from Shipment table) → net profit calculated.
3. Two GridViews display paid orders and shipment records → profit shown in green if positive, red if negative.

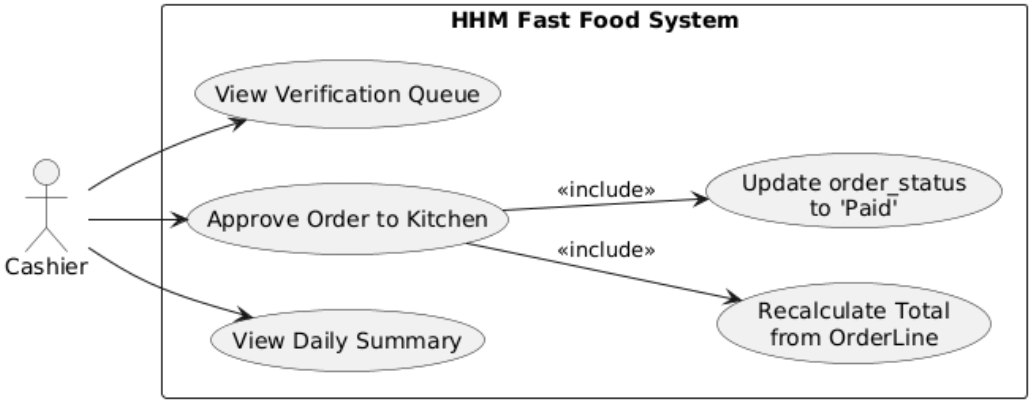


3.1.12 Use Case 12 - Cashier: Approve Order to Kitchen

Actor: Cashier

Steps:

1. CashierDashboard.aspx loads orders with status 'Verification'
2. Cashier reviews customer name and total bill → clicks "Approve to Kitchen"
3. System recalculates total from OrderLine → updates order_status to 'Paid' → order becomes visible to Chef
4. Daily Summary tab shows today's revenue and processed order count.



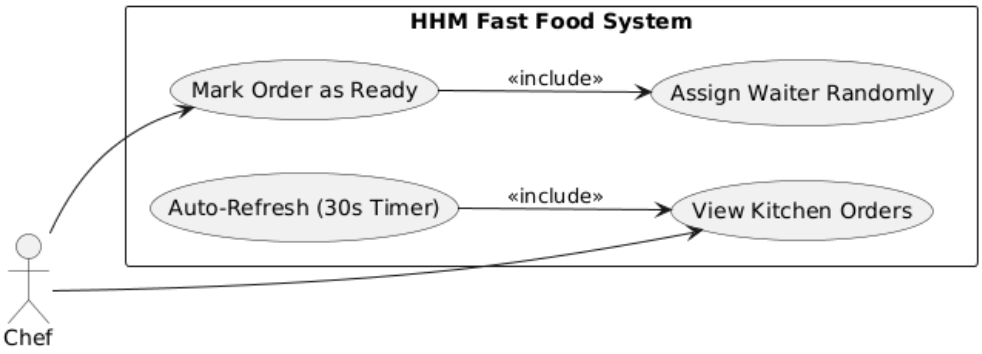
3.1.13 Use Case 13 - Chef: View and Complete Kitchen Orders

Actor: Chef

Steps:

1. ChefDashboard.aspx loads orders with status 'Verification' from the Order table, joining OrderLine and MenuItem to produce an items list using STRING_AGG

2. Chef views order details → clicks "Mark as Ready" → system updates order_status to 'Ready' and randomly assigns a Waiter (role_id=4)
3. UpdatePanel refreshes the grid and shows assignment message → a Timer auto-refreshes every 30 seconds.

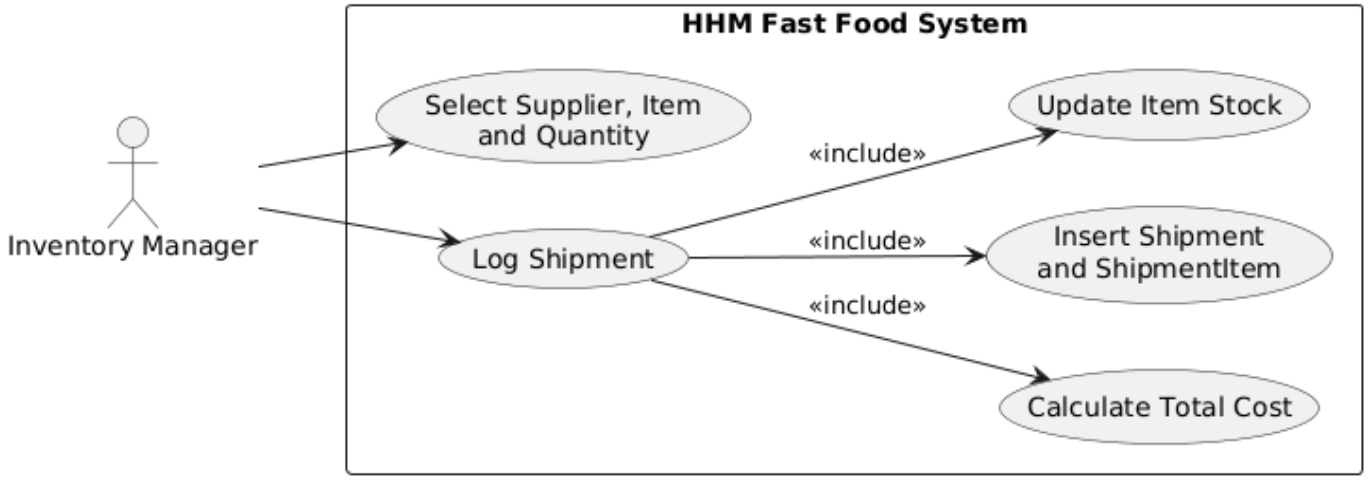


3.1.14 Use Case 14 - Inventory Manager: Restock Item

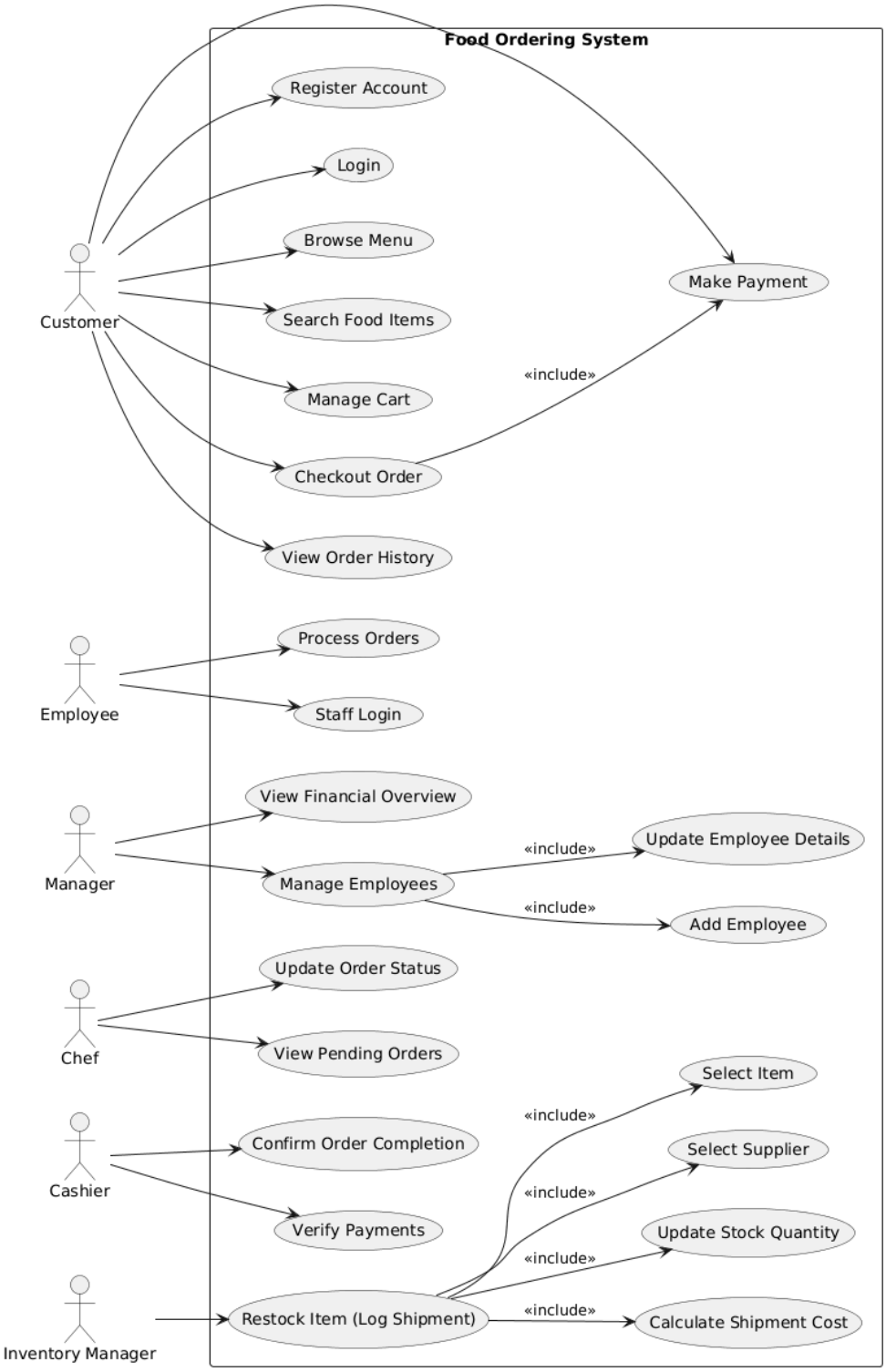
Actor: Inventory Item

Steps:

1. InventoryDashboard.aspx loads dropdown lists for Suppliers and Items
2. Inventory Manager selects supplier, item, and quantity → clicks Log Shipment
3. System fetches item unit price → calculates total cost → within a SQL Transaction: inserts Shipment record, inserts ShipmentItem record, updates Item quantity_in_stock → transaction committed → success message with quantity and cost shown.

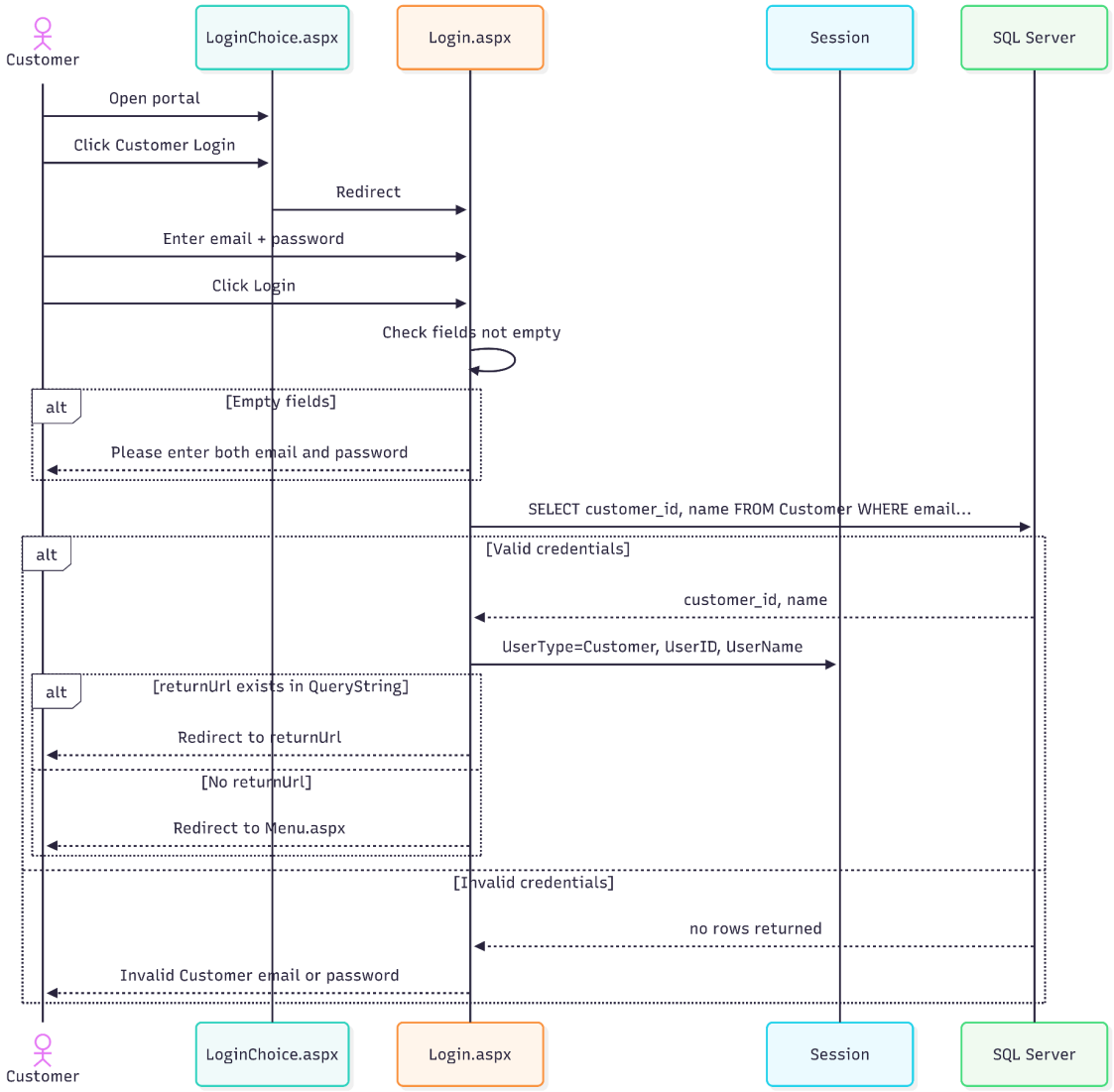


3.2 Use Case Model:

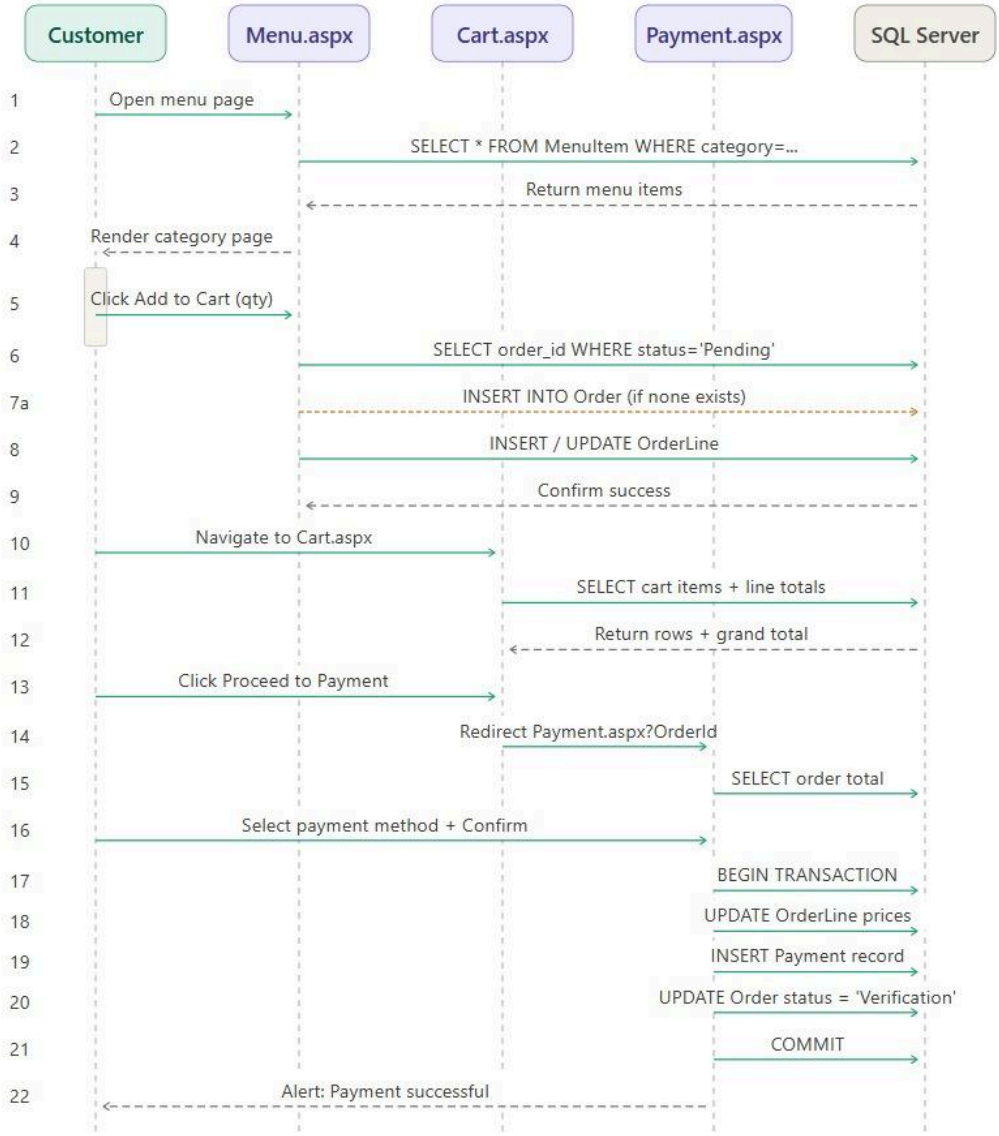


3.3 Sequence Diagram

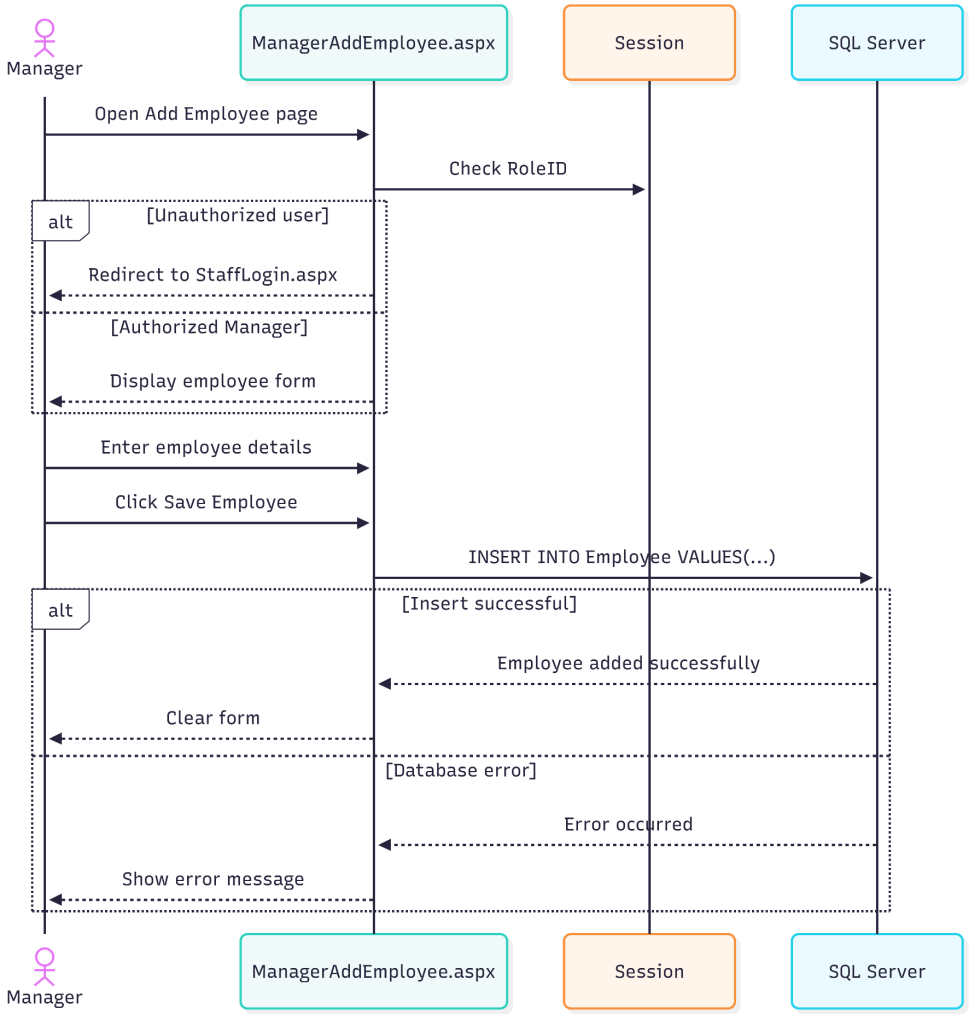
3.3.1 Customer Login Flow



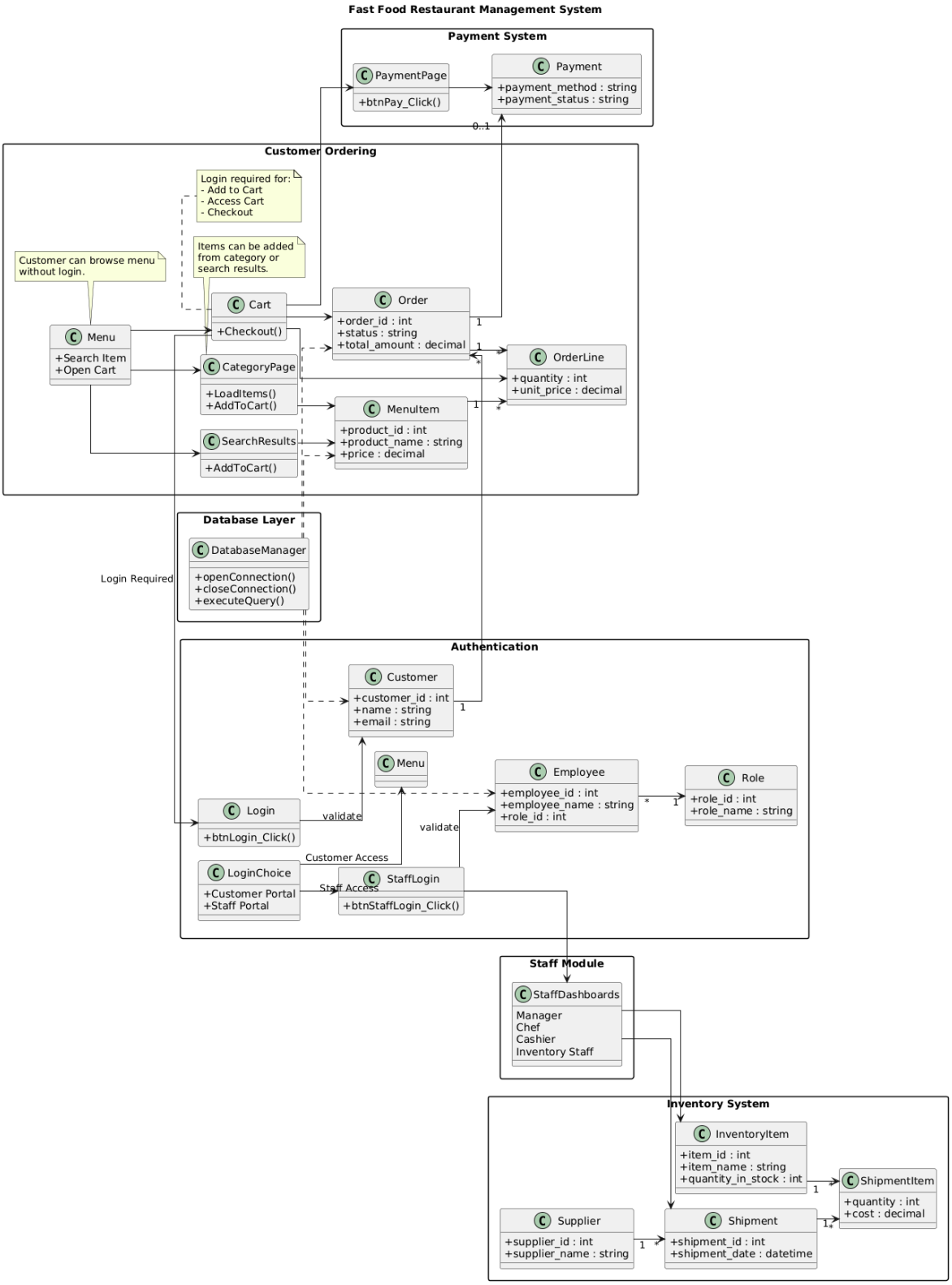
3.3.2 Customer places order



3.3.3 Manager adds New Employee

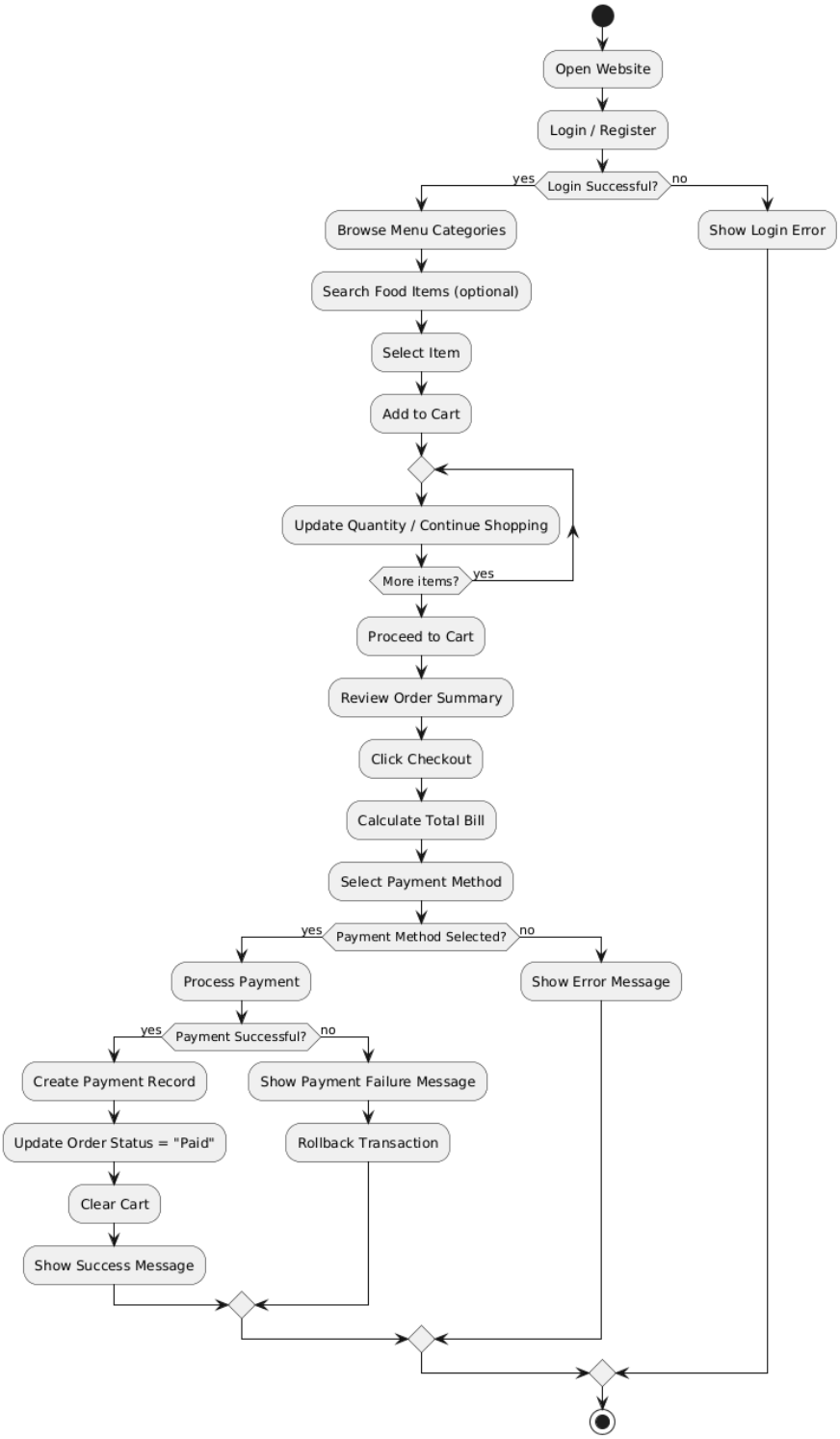


3.4 Class Diagram

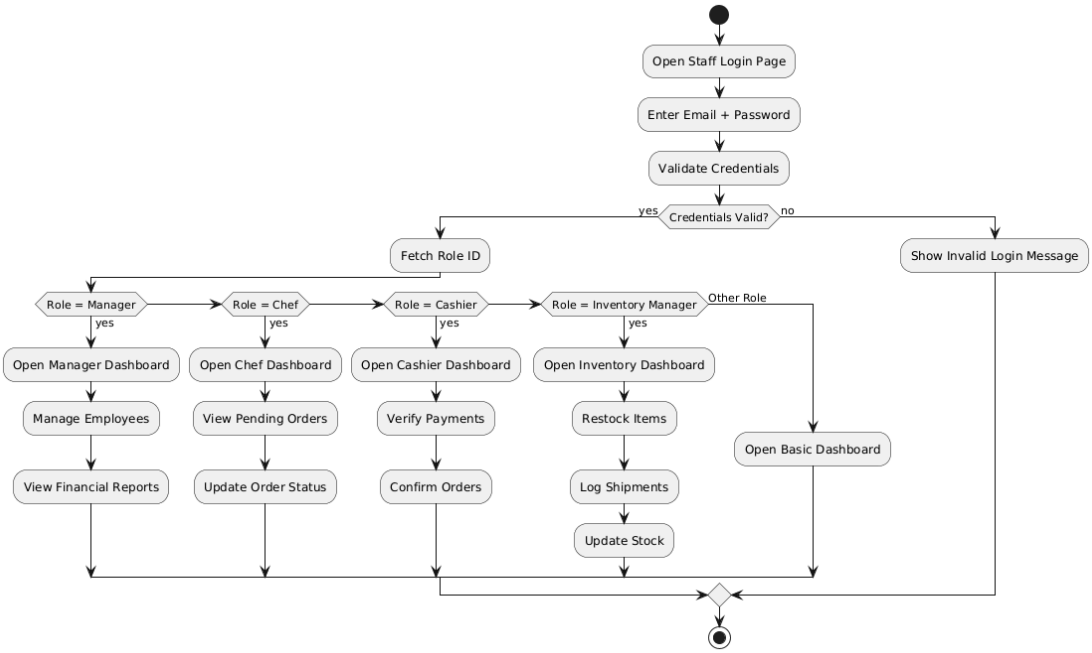


3.5 Activity Diagram

3.5.1 Customer Activity Diagram:



3.5.2 Employee Side:



3.6 Component & Deployment Model

3.6.1 Component Model:

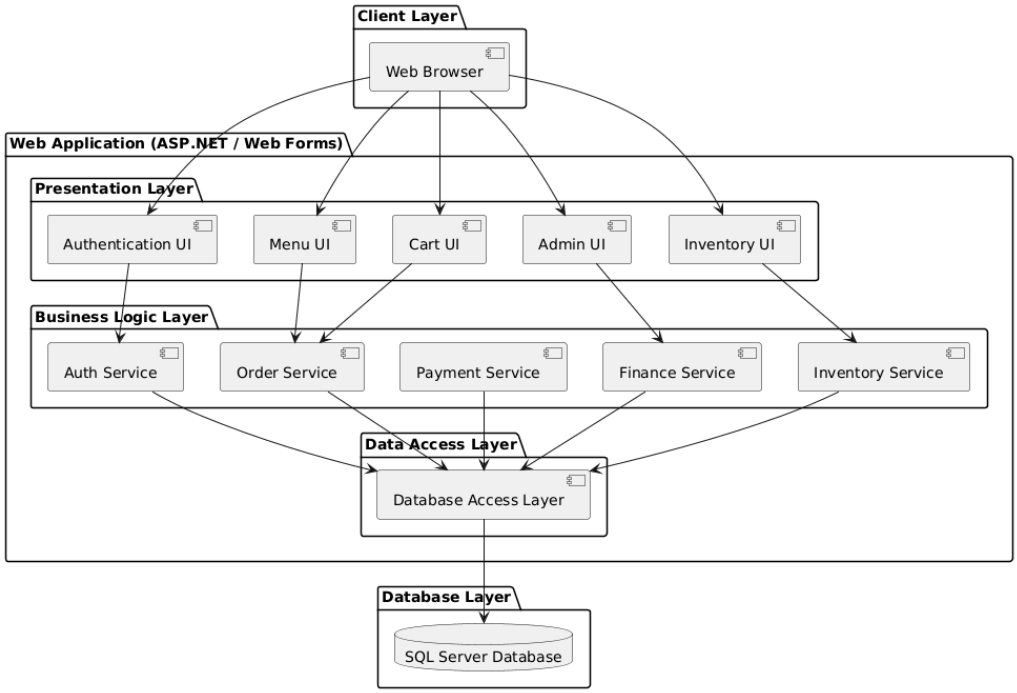
The component model of the HHM describes the high-level software structure of the application and how different parts of the system interact with each other.

The system follows a **3-tier architecture**, which includes the Presentation Layer, Business Logic Layer, and Data Access Layer.

- The **Presentation Layer** represents the user interface of the system, where users (customers and staff) interact with features such as login, menu browsing, cart management, administration, and inventory operations.
- The **Business Logic Layer** handles the core functionality of the system. It processes user requests such as authentication, order processing, payment handling, inventory updates, and financial calculations.
- The **Data Access Layer** is responsible for communicating with the database. It performs operations such as inserting, updating, deleting, and retrieving data from the SQL Server database.

All modules in the system communicate with the database through the Data Access Layer to ensure consistency, maintainability, and security.

Overall, this component model ensures separation of concerns, making the system modular, scalable, and easier to maintain.



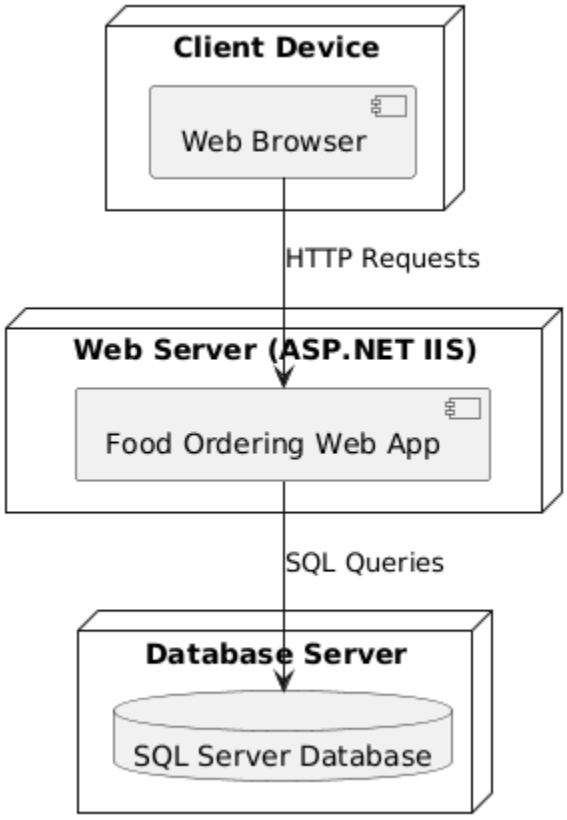
3.6.2 Deployment Model:

The deployment model of the HHM describes how the system is physically deployed and how its components are distributed across different hardware environments.

The system follows a **client–server architecture** consisting of three main nodes: the client device, the web server, and the database server.

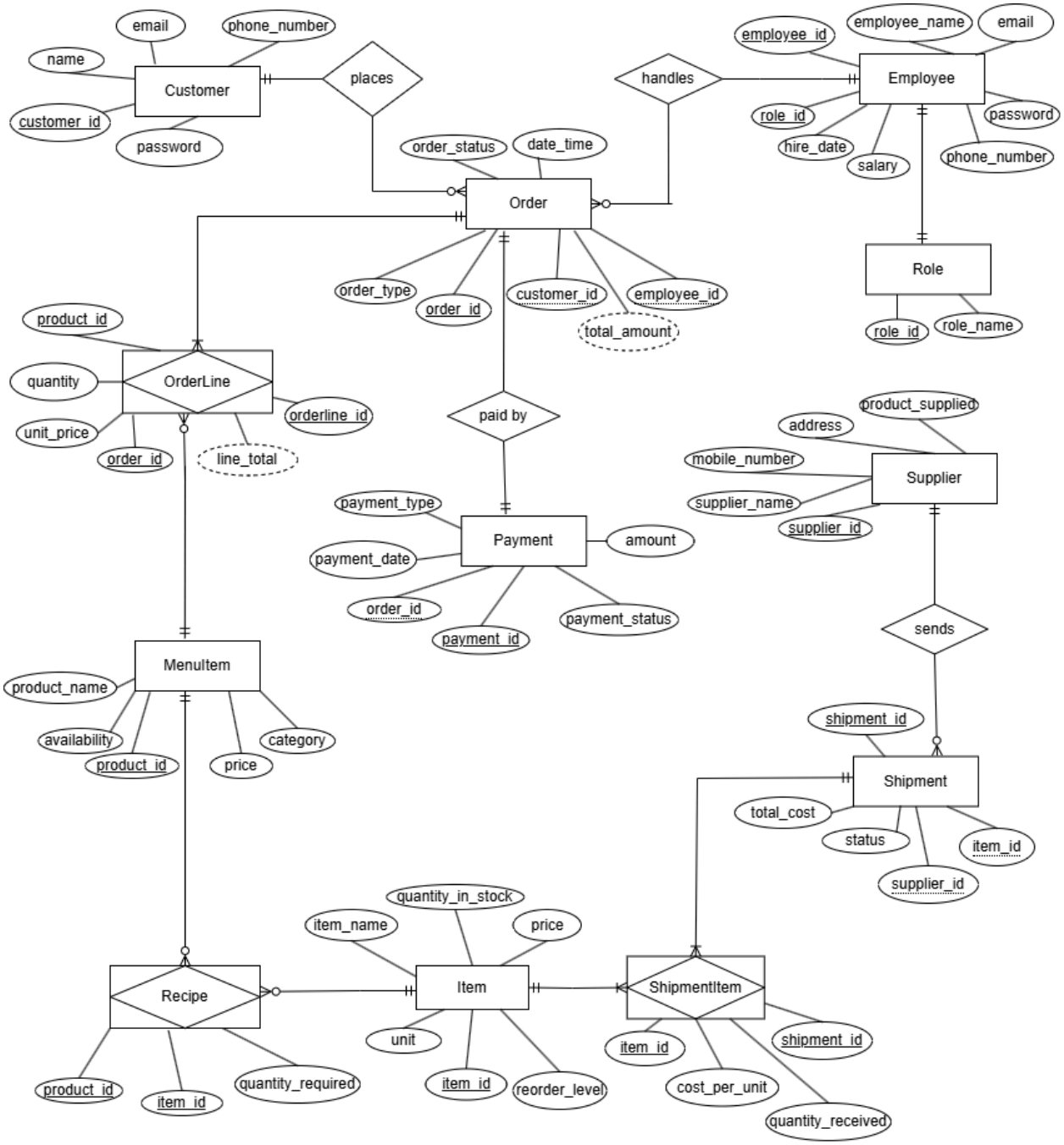
- The **Client Layer** represents the user's device (such as a laptop or mobile phone) where the application is accessed through a web browser. Customers and staff interact with the system by sending HTTP requests from the browser.
- The **Web Server Layer** hosts the ASP.NET Food Ordering Web Application. This server is responsible for processing user requests, executing business logic, managing sessions, and generating dynamic web pages. It acts as the central processing unit of the system.
- The **Database Server Layer** contains the SQL Server database, which stores all persistent data such as users, orders, menu items, payments, inventory, and employee records. The web application communicates with the database using SQL queries.

This deployment structure ensures that the system is scalable, secure, and efficient, as the presentation, processing, and data storage responsibilities are separated across different environments.

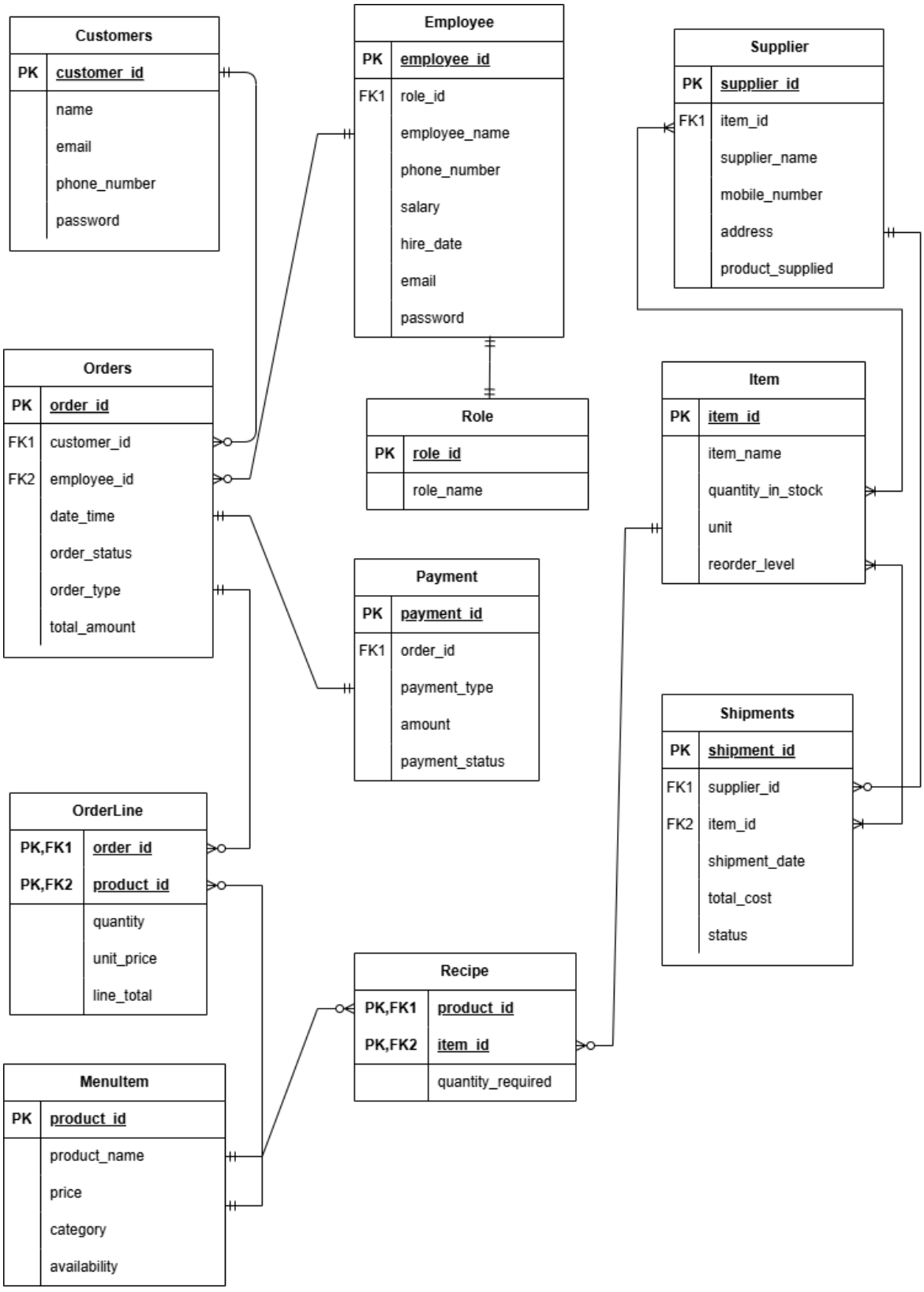


4.DataBase Implementation

4.1 ER model

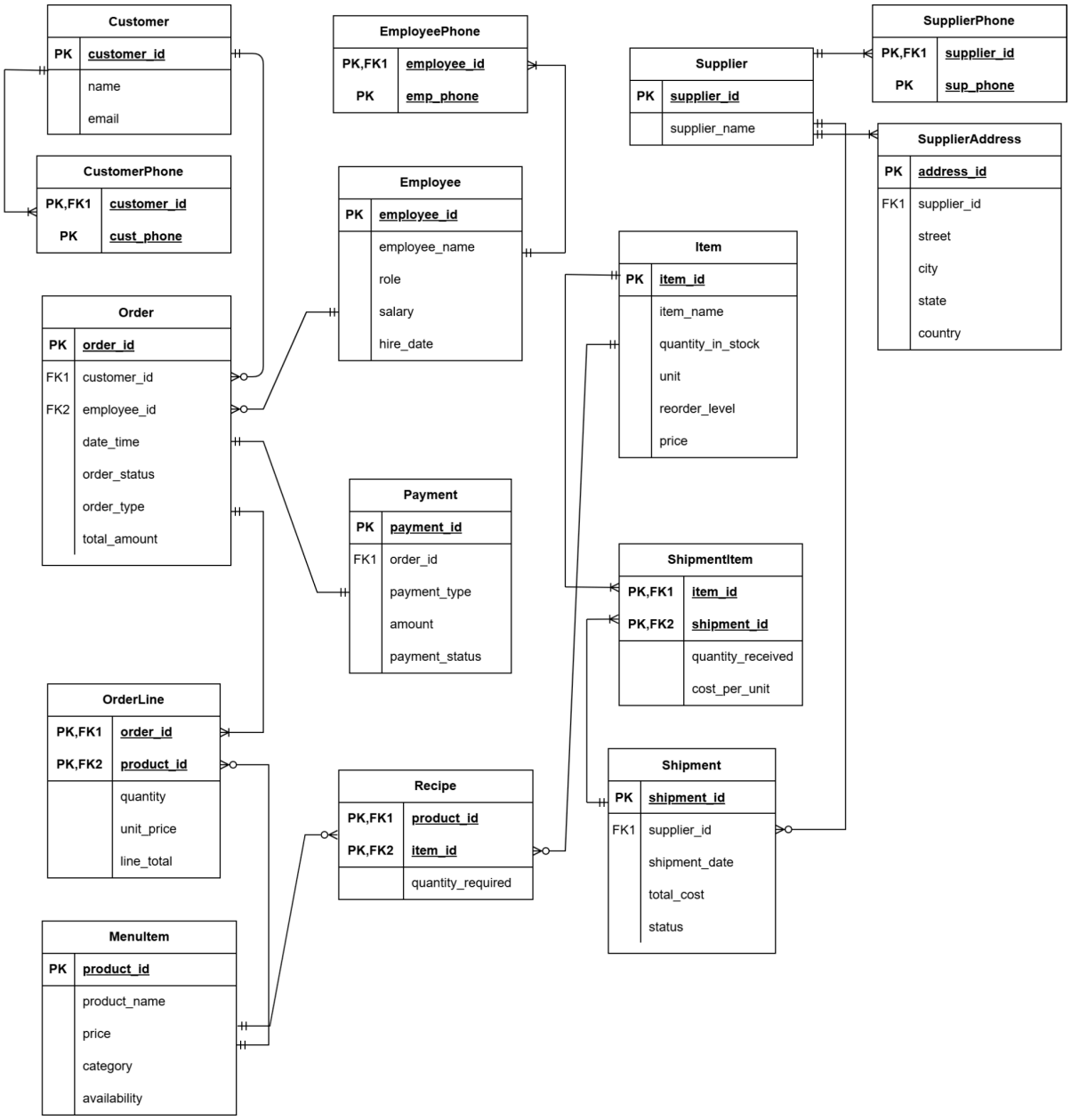


4.2 Relational Model:

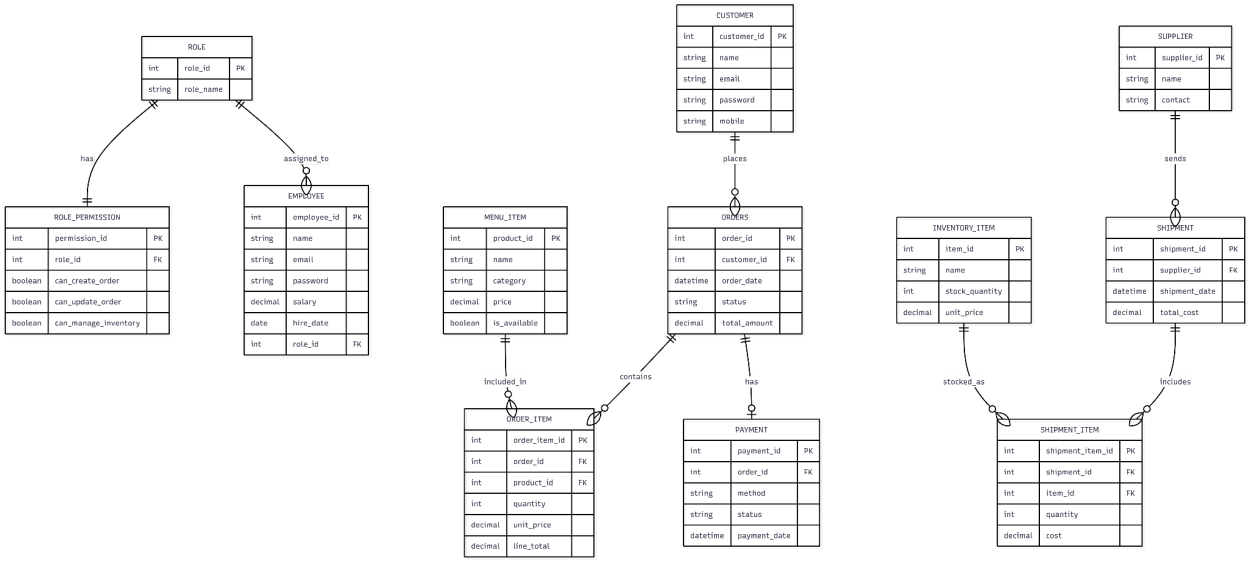


4.3 Relational Model With Normalization:

Final Normalized Form



4.4 Physical Model:



4.5 SQL Implementation:

4.5.1 Schema

1. Role:

```
CREATE TABLE Role (
 role_id INT PRIMARY KEY,
 role_name VARCHAR(50) NOT NULL
);
```

2. Role Permissions:

```
CREATE TABLE RolePermissions (
    role_id INT PRIMARY KEY,
    can_insert_order BIT,
    can_update_order BIT,
    can_insert_payment BIT,
    can_update_stock BIT,
    FOREIGN KEY (role_id) REFERENCES Role(role_id)
);
```

3. Customer

```
CREATE TABLE Customer (
    customer_id INT IDENTITY(1,1) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    password VARCHAR(100) NOT NULL
  );
```

4. CustomerPhone

```
CREATE TABLE CustomerPhone (
    customer_id INT NOT NULL,
    cust_phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (customer_id, cust_phone),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);
```

5. Employee

```
CREATE TABLE Employee (
    employee_id INT IDENTITY(1,1)  PRIMARY KEY,
    employee_name VARCHAR(100) NOT NULL,
    Role_id INT,
    salary DECIMAL(10,2),
    hire_date DATE,
    password VARCHAR(100),
    email VARCHAR(100) UNIQUE,
     FOREIGN KEY (role_id) REFERENCES Role(role_id)
);
```

6. EmployeePhone

```
CREATE TABLE EmployeePhone (
    employee_id INT NOT NULL,
    emp_phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (employee_id, emp_phone),
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);
```

7. Supplier

```
CREATE TABLE Supplier (
```

```
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(100) NOT NULL,

   );
```

8. SupplierPhone

```
CREATE TABLE SupplierPhone (
    supplier_id INT NOT NULL,
    sup_phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (supplier_id, sup_phone),
    FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id)
);
```

9. SupplierAddress

```
CREATE TABLE SupplierAddress (
    address_id INT IDENTITY(1,1) PRIMARY KEY,
    supplier_id INT NOT NULL,
    street VARCHAR(100),
    city VARCHAR(50),
    state VARCHAR(50),
    country VARCHAR(50),
    FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id)
);
```

10. Item

```
CREATE TABLE Item (
    item_id INT PRIMARY KEY,
    item_name VARCHAR(100) NOT NULL,
    quantity_in_stock DECIMAL(10,2) DEFAULT 0,
    unit VARCHAR(20),
    reorder_level DECIMAL(10,2) DEFAULT 0
    price DECIMAL(10,2) NOT NULL
);
```

11. MenuItem

```
CREATE TABLE MenuItem (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
```

```
    category VARCHAR(50),
    availability BIT DEFAULT 1
);
```

12. Recipe

```
CREATE TABLE Recipe (
    product_id INT NOT NULL,
    item_id INT NOT NULL,
    quantity_required DECIMAL(10,2),
    PRIMARY KEY (product_id, item_id),
    FOREIGN KEY (product_id) REFERENCES MenuItem(product_id),
    FOREIGN KEY (item_id) REFERENCES Item(item_id)
);
```

13. Shipment

```
CREATE TABLE Shipment (
    shipment_id INT IDENTITY(1,1)  PRIMARY KEY,
    supplier_id INT NOT NULL,
    shipment_date DATE,
    total_cost DECIMAL(10,2),
    status VARCHAR(50),
    FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id)
);
```

14. Order

```
CREATE TABLE [Order] (
    order_id INT IDENTITY(1,1)  PRIMARY KEY,
    customer_id INT NOT NULL,
    employee_id INT NOT NULL,
    date_time DATETIME,
    order_status VARCHAR(50),
    order_type VARCHAR(50),
    total_amount DECIMAL(10,2),
    delivery_address VARCHAR(150),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);
```

15. OrderLine

```
CREATE TABLE OrderLine (
    orderline_id INT IDENTITY(1,1) PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    unit_price DECIMAL(10,2),
    line_total DECIMAL(10,2),
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES [Order](order_id),
    FOREIGN KEY (product_id) REFERENCES MenuItem(product_id)
);
```

16. Payment

```
CREATE TABLE Payment (
    payment_id INT IDENTITY(1,1) PRIMARY KEY,
    order_id INT NOT NULL,
    payment_type VARCHAR(50),
    amount DECIMAL(10,2),
    payment_status VARCHAR(50),
    payment_date DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (order_id) REFERENCES [Order](order_id)
);
```

4.5.2 Data:

1. Role:

```
INSERT INTO Role VALUES
(1, 'Manager'),
(2, 'Chef'),
(3, 'Cashier'),
(4, 'Waiter'),
(5, 'Inventory');
```

2. RolePermissions:

```
INSERT INTO RolePermissions VALUES
(1, 1, 1, 1, 1), -- Manager
(2, 0, 0, 0, 0), -- Chef
```

(3, 1, 1, 1, 0), -- Cashier
(4, 0, 0, 0, 0), -- Waiter
(5, 0, 0, 0, 1); -- Inventory

3. Customer:

INSERT INTO Customer (name, email, password) VALUES
('Salman Tariq', 'salman.tariq559@yahoo.com', 'cust101'),
('Nadia Malik', 'nadia.malik800@gmail.com', 'cust102'),
('Sana Mehmood', 'sana.mehmood536@yahoo.com', 'cust103'),
('Sadia Butt', 'sadia.butt248@outlook.com', 'cust104'),
('Saima Bhatti', 'saima.bhatti479@outlook.com', 'cust105'),
('Aamir Jamil', 'aamir.jamil958@gmail.com', 'cust106'),
('Zainab Khan', 'zainab.khan792@hotmail.com', 'cust107'),
('Shakeel Javed', 'shakeel.javed431@outlook.com', 'cust108'),
('Saba Raza', 'saba.raza674@hotmail.com', 'cust109'),
('Nimra Bhatti', 'nimra.bhatti49@outlook.com', 'cust110'),

4. Employee:

INSERT INTO Employee (employee_name, email, role_id, salary, hire_date, password)
VALUES
('Ahmed Raza', 'ahmed.raza@hhm.com', 1, 70000, '2021-12-06', 'emp101'),
('Hassan Ali', 'hassan.ali@hhm.com', 2, 50000, '2023-08-07', 'emp102'),
('Omar Hussain', 'omar.hussain@hhm.com', 2, 50000, '2025-09-09', 'emp103'),
('Umer Haider', 'umer.haider@hhm.com', 3, 45000, '2024-06-03', 'emp104'),
('Yasir Bhatti', 'yasir.bhatti@hhm.com', 3, 45000, '2022-04-28', 'emp105'),
('Amna Sheikh', 'amna.sheikh@hhm.com', 4, 35000, '2020-02-26', 'emp106'),
('Nida Iqbal', 'nida.iqbal@hhm.com', 4, 35000, '2020-12-24', 'emp107'),
('Rabia Shah', 'rabia.shah@hhm.com', 4, 35000, '2022-05-01', 'emp108'),
('Khadija Hussain', 'khadija.h@hhm.com', 4, 35000, '2020-09-28', 'emp109'),
('Ahmed', 'ahmed.admin@hhm.com', 5, 40000, '2020-03-20', 'emp110'),
('Mona', 'mona.inv@hhm.com', 5, 40000, '2021-05-23', 'emp111');

5. Supplier:

INSERT INTO Supplier (supplier_id, supplier_name) VALUES
(1, 'Metro Cash & Carry'),
(2, 'Imtiaz Super Market'),
(3, 'Al-Fateh Store'),
(4, 'Sabzi Mandi Wholesalers'),

(5, 'Shaheen Traders'),

6. Supplier Address:

INSERT INTO SupplierAddress (address_id, supplier_id, street, city, state, country) VALUES
(1, 1, 'Shop 358', 'Tariq Road', 'Karachi', 'Pakistan'),
(2, 2, 'Shop 394', 'Saddar Market', 'Karachi', 'Pakistan'),
(3, 3, 'Shop 317', 'Wholesale Market', 'Lahore', 'Pakistan'),
(4, 4, 'Shop 105', 'Model Town', 'Lahore', 'Pakistan'),
(5, 5, 'Shop 206', 'Saddar Market', 'Karachi', 'Pakistan'),

7. Item:

INSERT INTO Item (item_id, item_name, quantity_in_stock, unit, reorder_level, price) VALUES
(1, 'Chicken Breast', 159, 'kg', 35, 950.00),
(2, 'Beef Mince', 78, 'kg', 31, 1400.00),
(3, 'Chicken Tikka', 44, 'kg', 26, 1100.00),
(4, 'Chicken Broast Pieces', 289, 'pieces', 35, 120.00),
(5, 'Burger Buns', 296, 'pieces', 38, 10.00),
(6, 'Pizza Dough', 50, 'kg', 27, 250.00),
(7, 'Cheese Slices', 203, 'pieces', 41, 18.00),
(8, 'Mozzarella Cheese', 59, 'kg', 46, 1750.00),
(9, 'Cheddar Cheese', 23, 'kg', 17, 1850.00),
(10, 'Tomatoes', 87, 'kg', 40, 150.00),
(11, 'Onions', 69, 'kg', 40, 120.00),
(12, 'Lettuce', 79, 'kg', 35, 300.00),
(13, 'Capsicum', 59, 'kg', 10, 200.00),
(14, 'Cabbage', 26, 'kg', 46, 80.00),
(15, 'Cucumbers', 43, 'kg', 31, 110.00),
(16, 'Potatoes', 269, 'kg', 36, 140.00),
(17, 'French Fries', 102, 'kg', 46, 550.00),
(18, 'Mayo Sauce', 64, 'liters', 42, 600.00),
(19, 'Ketchup', 88, 'liters', 27, 450.00),
(20, 'Garlic Sauce', 60, 'liters', 12, 700.00);

8. MenuItem:

INSERT INTO MenuItem(product_id, product_name, price, category, availability) VALUES
(1,'Zinger Burger',1026,'Burger',1),
(2,'Beef Burger',1075,'Burger',1),
(3,'Chicken Burger',960,'Burger',1),
(4,'Grilled Burger',979,'Burger',1),
(5,'Cheese Burger',1040,'Burger',1),

(6,'BBQ Burger',1075,'Burger',1),
(7,'Spicy Burger',910,'Burger',1),
(8,'Tandoori Burger',968,'Burger',1),
(9,'Double Patty Burger',1071,'Burger',1),
(10,'Mushroom Burger',961,'Burger',1),

(11,'Chicken Tikka Pizza',1427,'Pizza',1),
(12,'BBQ Pizza',1366,'Pizza',1),
(13,'Fajita Pizza',1264,'Pizza',1),
(14,'Cheese Pizza',1211,'Pizza',1),
(15,'Pepperoni Pizza',1489,'Pizza',1),
(16,'Veggie Pizza',1475,'Pizza',1),
(17,'Chicken Supreme Pizza',1304,'Pizza',1),
(18,'Malai Boti Pizza',1459,'Pizza',1),

(19,'Half Broast',1092,'Broast',1),
(20,'Full Broast',1208,'Broast',1),
(21,'Crispy Broast',1146,'Broast',1),
(22,'Spicy Broast',1193,'Broast',1),
(23,'Garlic Broast',1300,'Broast',1),
(24,'Broast with Fries',1063,'Broast',1),

(25,'Chicken Shawarma',742,'Shawarma',1);

9. Recipe:

INSERT INTO Recipe (product_id, item_id, quantity_required) VALUES
-- 1. Zinger Burger (product_id = 1)
(1, 1, 0.15), -- 0.15 kg Chicken Breast (for fillet)
(1, 5, 2.00), -- 2 pieces Burger Buns
(1, 12, 0.05), -- 0.05 kg Lettuce
(1, 18, 0.03), -- 0.03 liters Mayo Sauce

-- 2. Beef Burger (product_id = 2)
(2, 2, 0.18), -- 0.18 kg Beef Mince (for patty)
(2, 5, 2.00), -- 2 pieces Burger Buns
(2, 9, 0.02), -- 0.02 kg Cheddar Cheese (slices)
(2, 10, 0.05), -- 0.05 kg Tomatoes
(2, 19, 0.02), -- 0.02 liters Ketchup

-- 3. Chicken Tikka Pizza (product_id = 11) - assuming a 1kg pizza base
(11, 6, 0.40), -- 0.40 kg Pizza Dough (for one medium pizza)
(11, 3, 0.20), -- 0.20 kg Chicken Tikka
(11, 8, 0.15), -- 0.15 kg Mozzarella Cheese

(11, 13, 0.05), -- 0.05 kg Capsicum
(11, 11, 0.08), -- 0.08 kg Onions

-- 4. Half Broast (product_id = 19)
(19, 4, 4.00), -- 4 pieces Chicken Broast Pieces (for a 'Half' portion)
(19, 16, 0.25), -- 0.25 kg Potatoes (for side order of fries/potatoes)
(19, 18, 0.02), -- 0.02 liters Mayo Sauce

-- 5. Full Broast (product_id = 20)
(20, 4, 8.00), -- 8 pieces Chicken Broast Pieces (for a 'Full' portion)
(20, 16, 0.50), -- 0.50 kg Potatoes
(20, 19, 0.05), -- 0.05 liters Ketchup

-- 6. Chicken Shawarma (product_id = 25)
(25, 1, 0.12), -- 0.12 kg Chicken Breast
(25, 14, 0.03), -- 0.03 kg Cabbage
(25, 15, 0.05), -- 0.05 kg Cucumbers
(25, 20, 0.04); -- 0.04 liters Garlic Sauce

10. Shipment:

INSERT INTO Shipment (supplier_id, shipment_date, total_cost, status)
VALUES
(1, '2023-08-28', 50424, 'Delivered'),
(2, '2025-11-12', 85589, 'Delivered'),
(3, '2024-05-16', 9334, 'Delivered'),
(4, '2024-08-10', 87913, 'Delivered'),
(5, '2025-06-17', 35017, 'Delivered'),

11. ShipmentItem:

INSERT INTO ShipmentItem (shipment_id, item_id, quantity_received, unit_price)
VALUES
 (1, 1, 100, 950.00), -- 100 kg Chicken Breast from Al-Madina
 (2, 2, 78, 1400.00), -- 78 kg Beef Mince from Karachi Meat
 (3, 16, 250, 140.00), -- 250 kg Potatoes from Punjab Agro
 (3, 11, 10, 120.00), -- 10 kg Onions
 (4, 19, 50, 450.00), -- 50L Ketchup from National Foods
 (4, 18, 30, 600.00), -- 30L Mayo

5 Conclusion:

We developed the HHM Food Management System to make it easier for restaurants to manage their processes by integrating customer orders, payments, and employee management into one single system. The development process includes the concepts of web development, which include database connectivity, authentication, role-based access, and state management in a real-world scenario. With separate dashboards for customer, manager, and employees, the system became much easier to manage for all users.