Project Report

Course Instructor

Sir Bilal Khalid Dar

Submitted By

Amna Hassan 22i-8759

M Shuja Uddin 22i-2553

Date

May 5,2024

Fall 2024

Department of Software Engineering

FAST – National University of Computer and Emerging Sciences

Cafe management System

Detailed Functional Requirements of each module

1.Cashier Module:

1. User Authentication:

Cashiers can login and log out successfully using their credentials. If a cashier's credentials are not added in the database, then he/she cannot login.

2. Process Payments:

Process the payments of the current orders.

Calculate and display the total cost of orders.

Receive the payments (cash, credit card, etc.) and confirm the payment.

3. Change Calculation:

Calculate and display the remaining amount/return cash to be given when the customer pays with cash.

4. Payment Confirmation:

Confirm successful payment and update the order status accordingly. The order is completed successfully after payment confirmation. The record is successfully updated in database.

5. Refunds:

Process refunds in case of order cancellations or returns.

Ensure that refunds are accurately reflected in both the transaction history and inventory.

2. Inventory Manager Module

1. User Authentication:

Inventory managers can login and log out using their own credentials. Non authorized personals cannot login to the manager's account.

2. Inventory Management:

View Inventory:

Display a complete list of all products in the inventory.

Include information such as the item name, category, quantity, and expiration date.

Add New Item:

Allows Inventory Managers to add new items to their inventory.

Record important information such as the item's name, category, quantity, price, and expiration date.

Update Item Information:

Allow for a change of item information, such as quantity, pricing, and other important details.

Remove Item:

Allow the ability to remove things that are no longer stocked or sold by the cafe.

Maintain a complete record of all inventory-related transactions, including additions, updates, and removals.

3. Low Stock Alerts:

Set up automatic alerts for Inventory Managers when the stock of an item falls below a specified threshold. Provide suggestions for reordering or restocking low-inventory items.

4. Reporting:

Inventory Reports:

Generate reports laying out the inventory's present condition, including item amounts.

Provide details regarding things that are low or out of stock.

.

5. Supplier Management:

Maintain a database of suppliers' contact information.

Integrate with the ordering system to make stock replenishment more efficient.

3. Cafe Manager Module:

The Cafe Manager module regulates the cafe's general management.

1. Staff Management:

Add a Cashier

Enables the Cafe Manager to enter new cashier into the system.

Collect important details about cashiers , including name, contact information, position, and employment status.

Remove a Cashier

Allow for the deactivation or removal of former cafe cashiers.

Add an Inventory manager

Enables the Cafe Manager to enter new inventory manager into the system.

Collect important details about inventory manager, including name, contact information, position, and employment status.

Remove an Inventory manager

Allow for the deactivation or removal of former cafe inventory manager.

2. Managing Inventory:

ADD items in Menu

By seeing the inventory cafe manager can regulate the menu by adding new items in specific categories.

Provides details for new item to add the item

Remove Item in Menu

By seeing the inventory cafe manager can regulate the menu by removing items from specific categories.

Provides details for the item to remove the item

3. Financial Management and Reporting

Create a financial report of expenses and budget.

Maintain a record of sales through the database for reporting.

Create a report of each cafe activity, generating trends on sales .

Create detailed reports for daily, weekly, and monthly sales.

Analyze sales by category, time of day, and individual items.

4. Menu Planning

Modify the cafe menu by adding new items and modifying prices.

Plan the menu and prices.

Organize promotions, deals and discounts for specific items, dates or events.

5. Dealing Rating and Reviews

Enables to Deal with the rating and reviews given by the customer

4.Customer Module:

1. Explore Food Section:

User-Friendly UI:

Create an easy-to-use interface for customers to navigate the café menu.

Organize products properly to make it easier for customer browsing and finding wanted dishes.

Visual Representation:

Improve customer experience by using visually attractive images and descriptions for menu items.

Include ingredients, nutritional information, and pricing.

2. Inventory Interaction:

Real-Time Inventory Updates:

Make sure customers can see the real-time availability of items.

Item Details:

Display detailed information about each menu item, including its popularity, nutritional facts, and customer reviews.

3. Selecting and Adding Items to Cart:

Item Selection:

Allow clients to customize their orders by selecting items from the menu and specifying preferences.

Intuitive Cart Management:

Create a user-friendly shopping cart system for customers to review their items.

Customers can add and remove items from cart in real time.

Allow for simple alterations, such as quantity or item removal.

Place order after the selection of items

4. Reviews and Ratings:

Review and Rating Submission:

Encourage customers to share their dining experiences though reviews and ratings.

Allow to provide written input and numerical ratings on a scale.

Transparent Pricing:

Display whole purchase cost, including taxes and other charges.

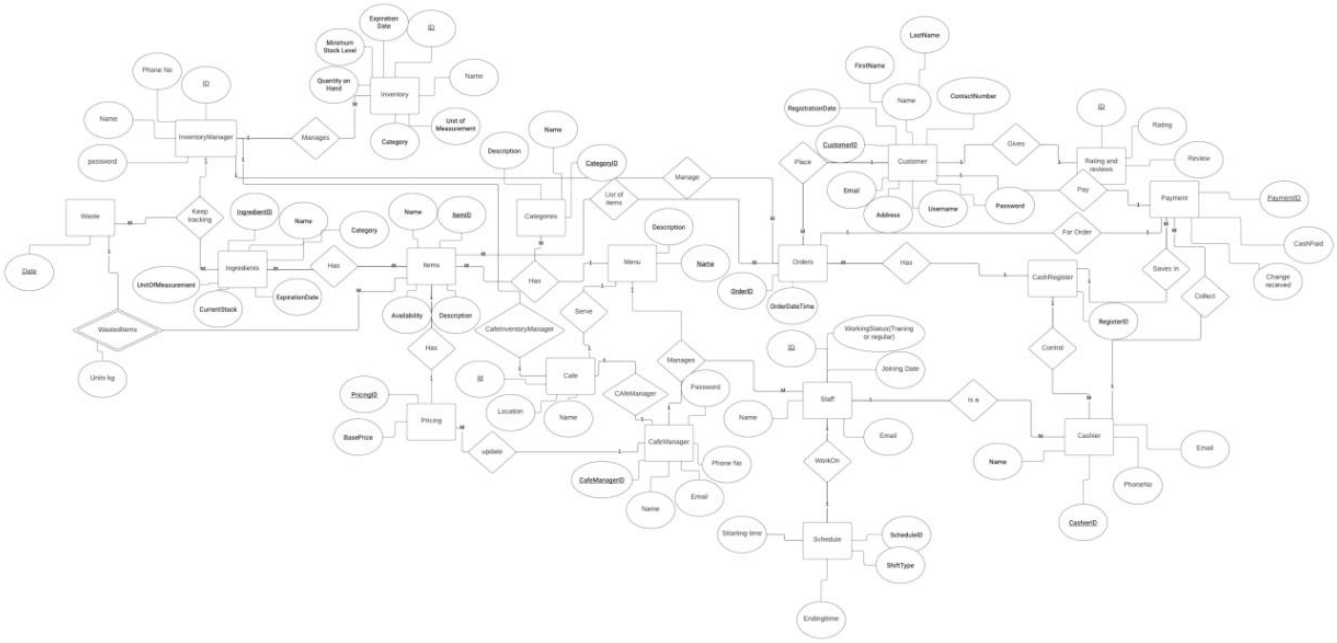
Offer transparent pricing to minimize surprises at checkout.

Secure Checkout Process:

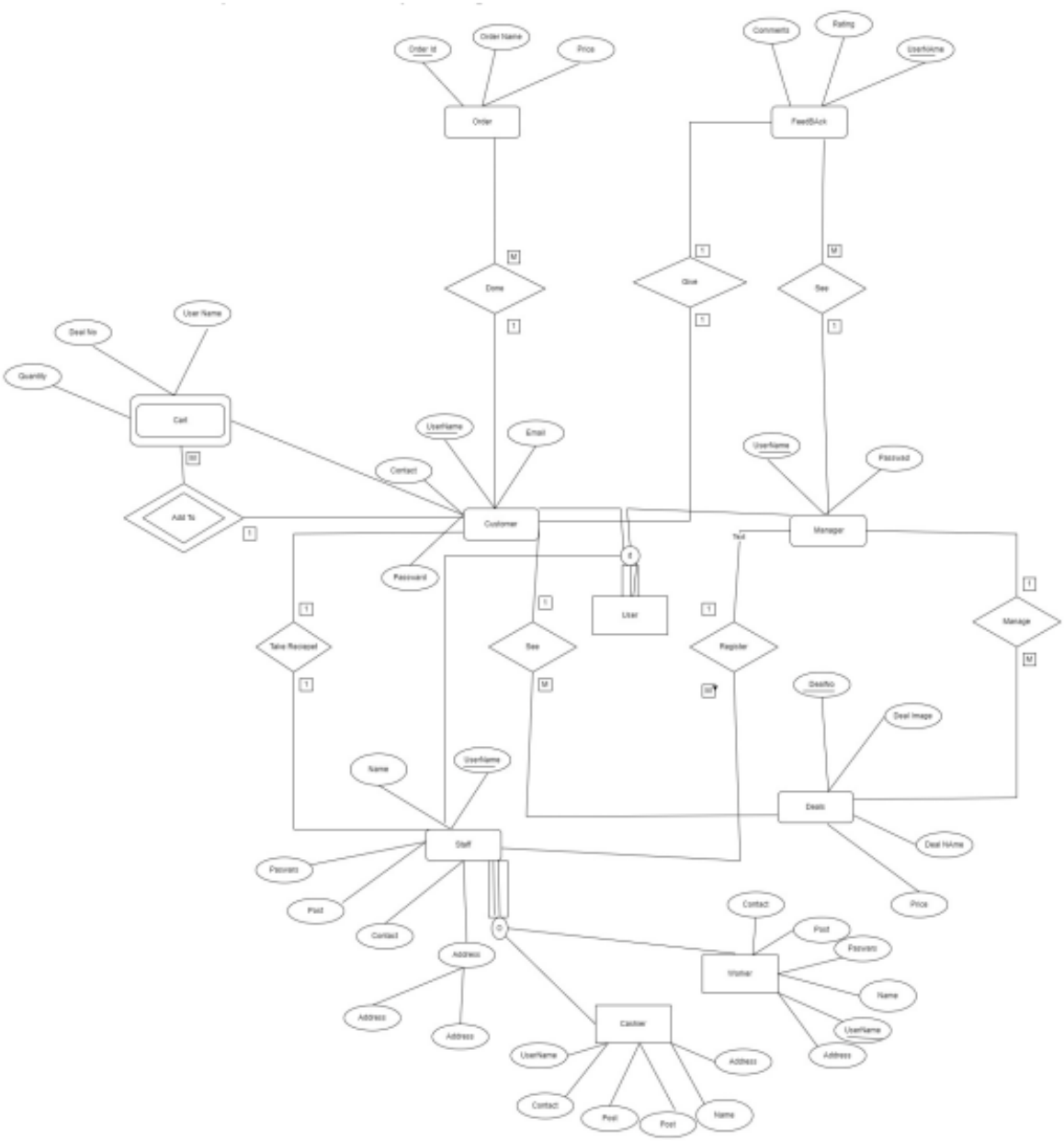
Provide a secure and simple checkout experience for customers to finish their orders.

Accept many payment methods, including credit cards and digital wallets.

Entity Relationship Diagram:



Enhanced Entity Relationship Diagram:



Relational Schema/Logical Schema

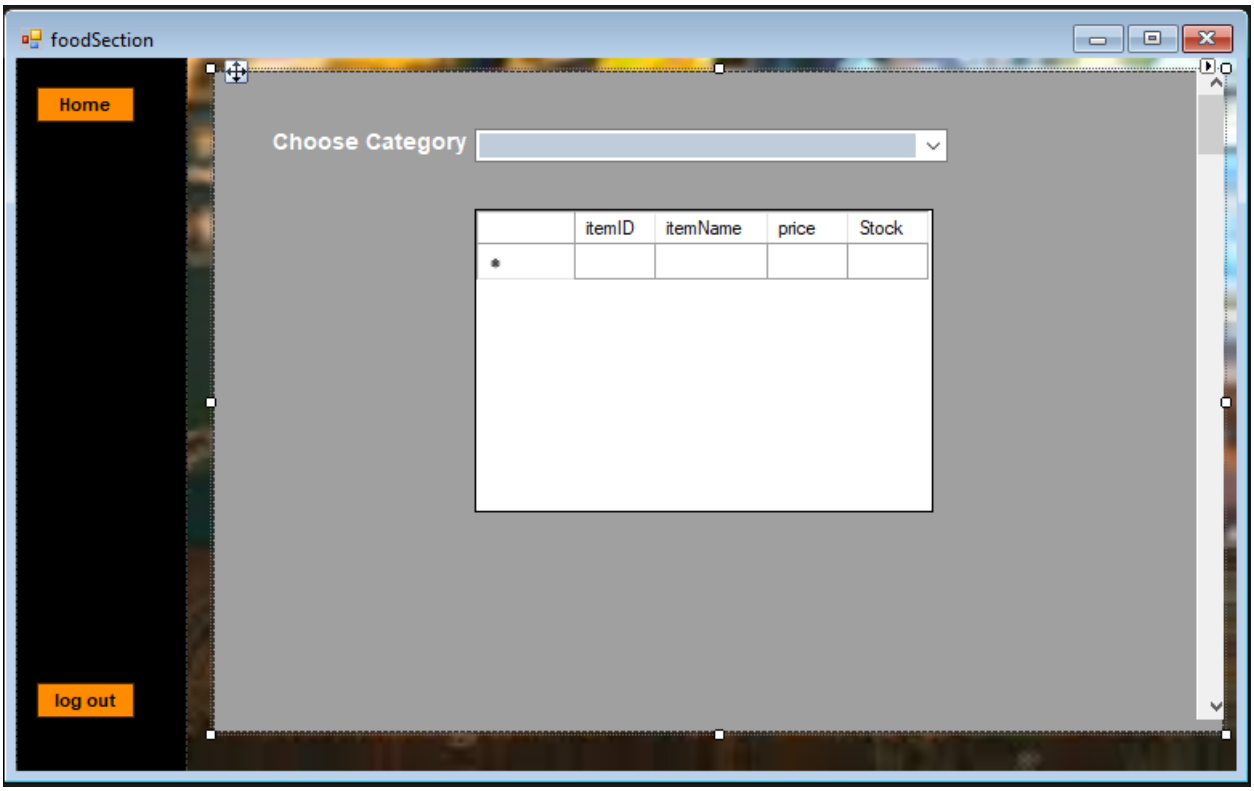
User Documentation and Help

Home Page

Anyone can simply explore the menu of menu by clicking the ‘Explore Now’ Button. No need to log in for exploring. This is only for exploring, order cannot be placed.

Explore Page

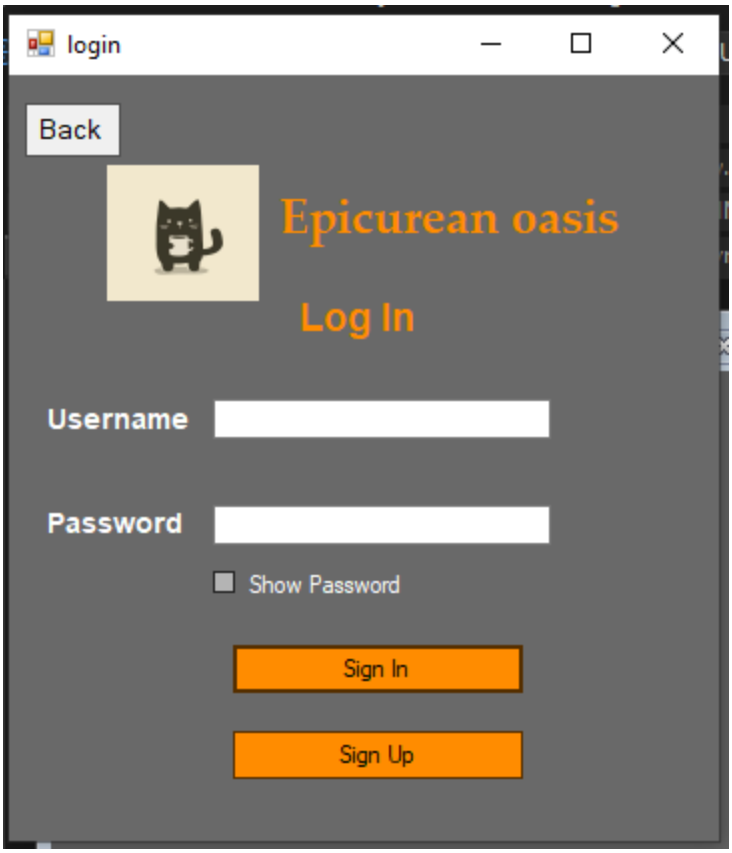
Any user can choose their desired category and explore the available items.



Login Page

Log into your account with credentials.

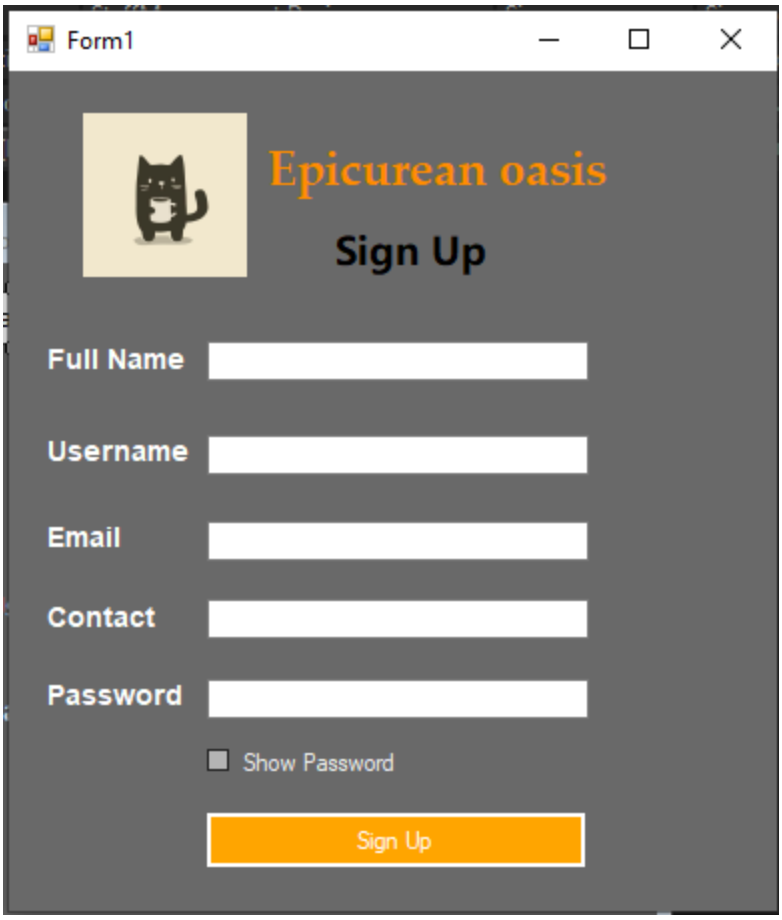
Your dashboard will appear automatically as you enter your login and password. Customers can explore food without logging in, but ordering requires a login.



Sign Up/ Register Page

Customers can create an account by giving their full name, username, email address, password, and other details.

Create an account and smoothly log in.



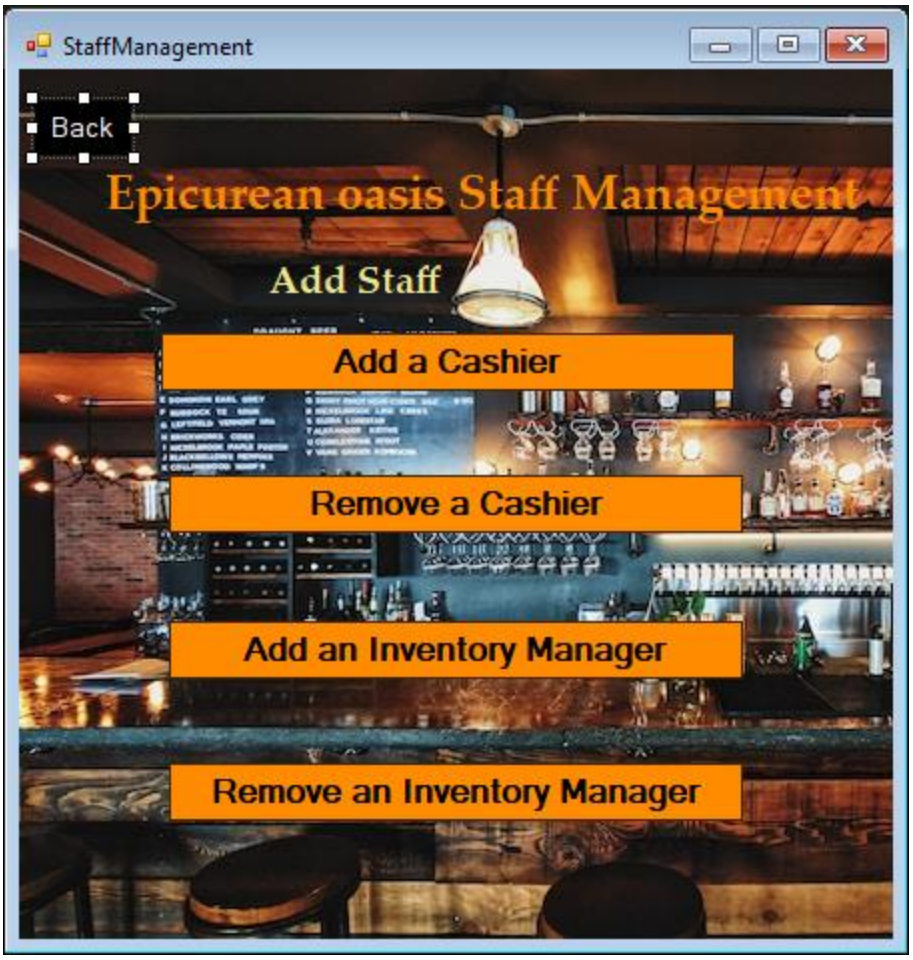
Cafe Manager Dashboard:

After logging in the interface will be shown to cafe manager.

Cafe manager can select any management operation

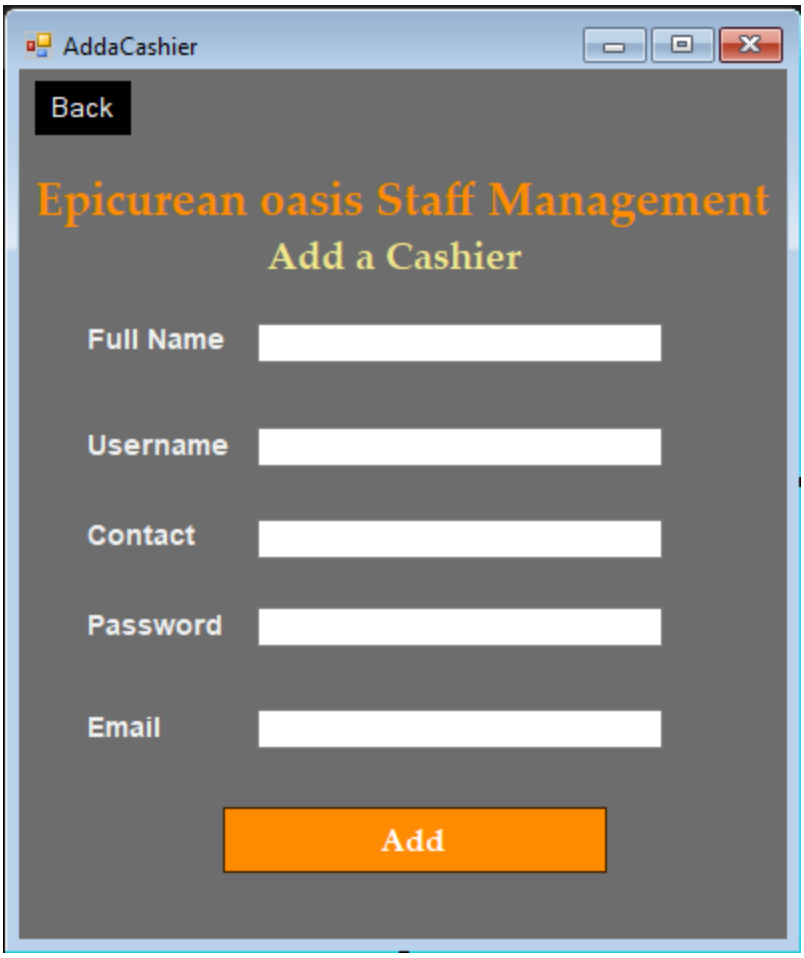
Staff Management

Cafe managers can choose any desired operation for staff management.



Add a cashier

Enter the necessary details to add a cashier.

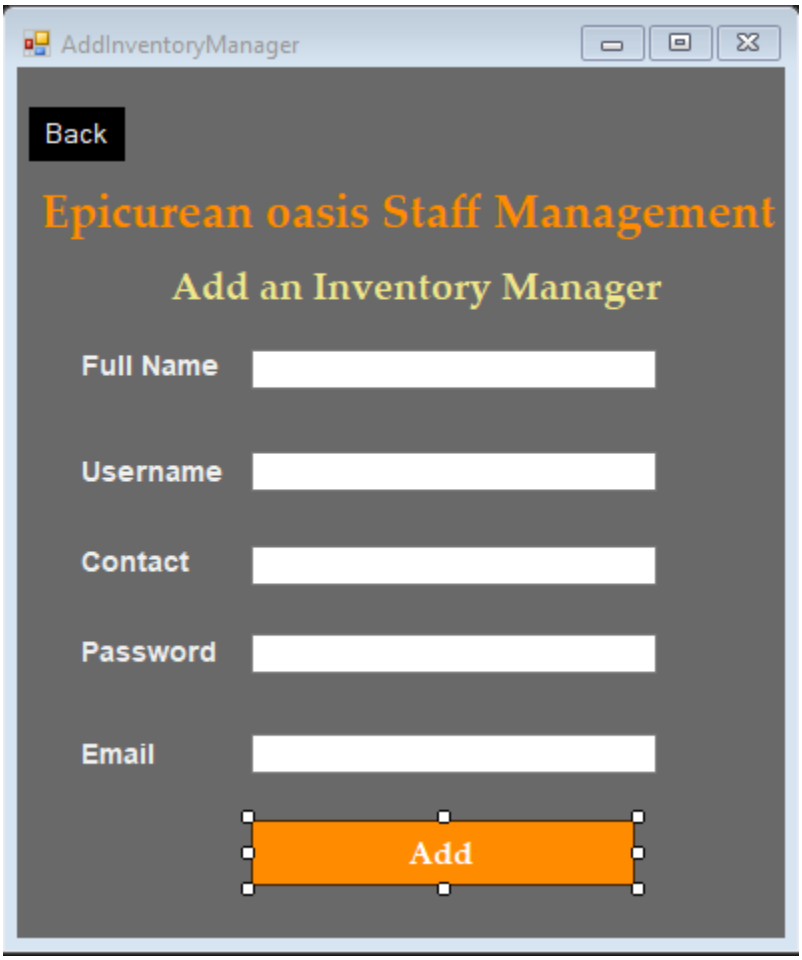


Remove a cashier

Enter the username to remove the cashier.

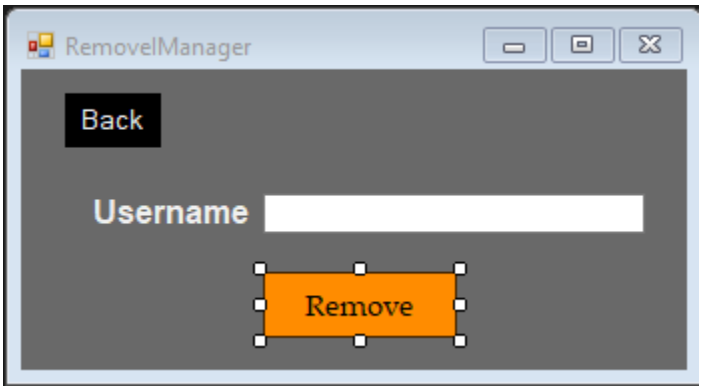
Add An Inventory Manager

Enter the necessary details to add a Inventory manager.



Remove an Inventory manager

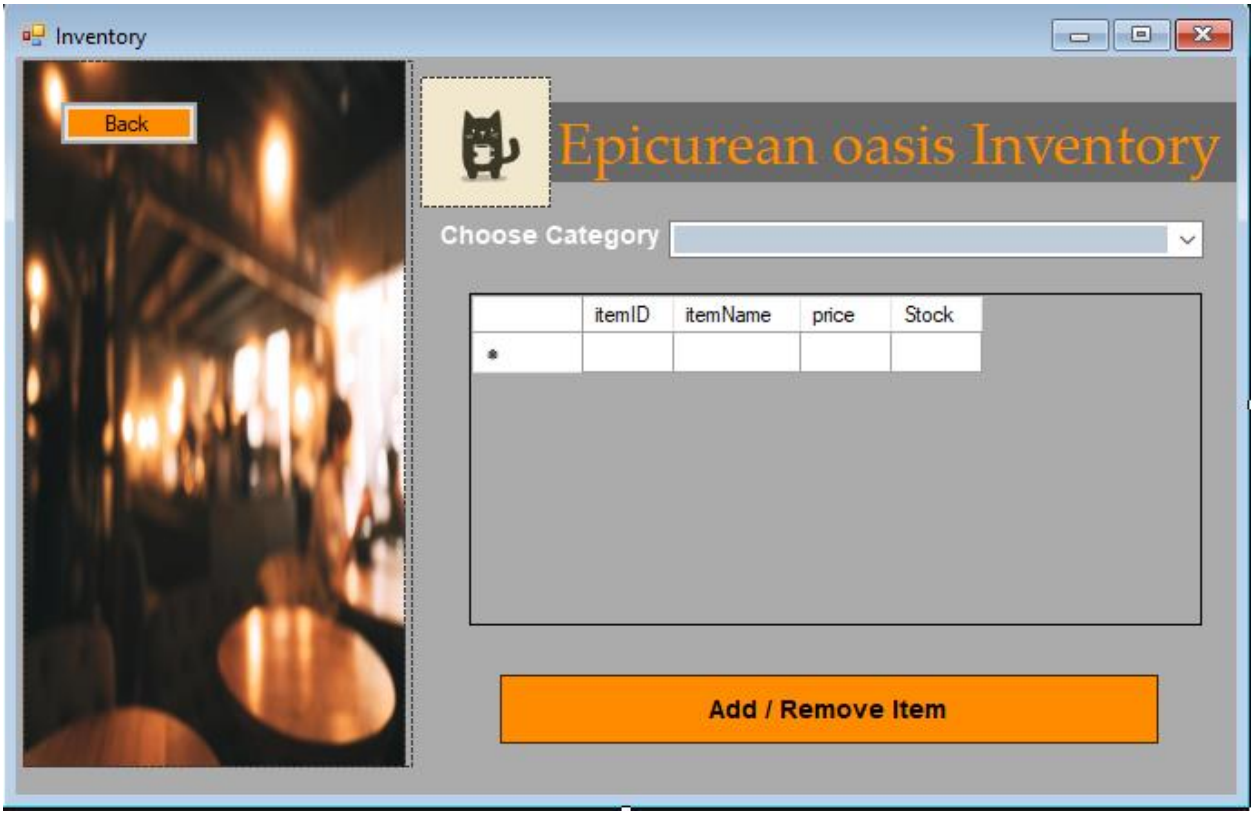
Enter the username to remove the Inventory Manager.



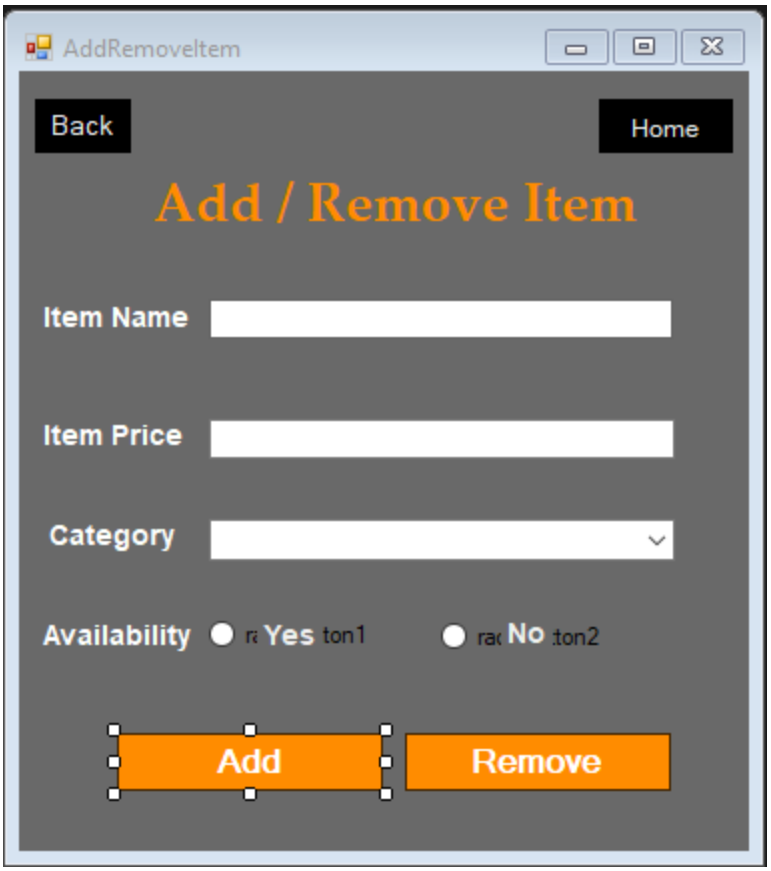
View Inventory

Cafe managers can choose a category to see the items.

Clicl on the add/remove button to add or remove an item from the menu

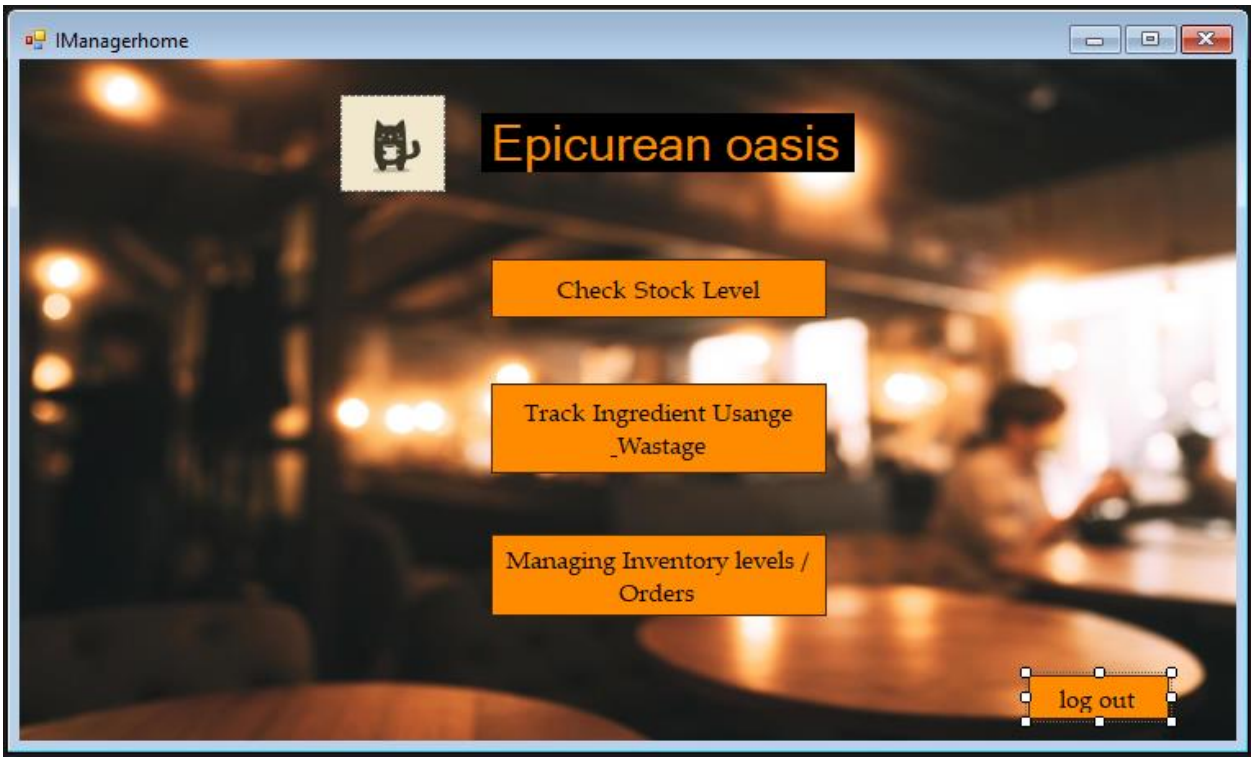


Add/Remove Item

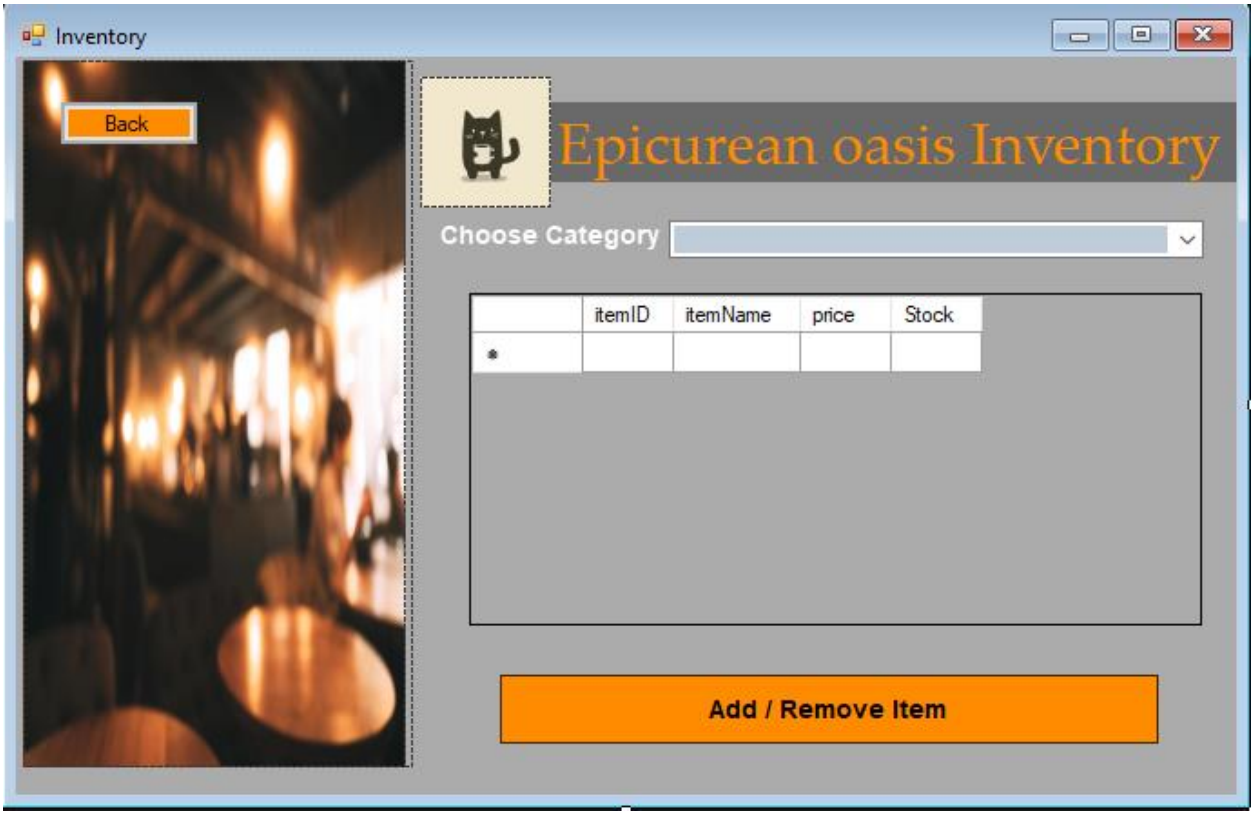


Inventory Manager Dashboard

Access the "Check Stock Level" section to view real-time stock levels for all items. Identify items that are low on stock, out of stock, or nearing expiration



Check Stock Level



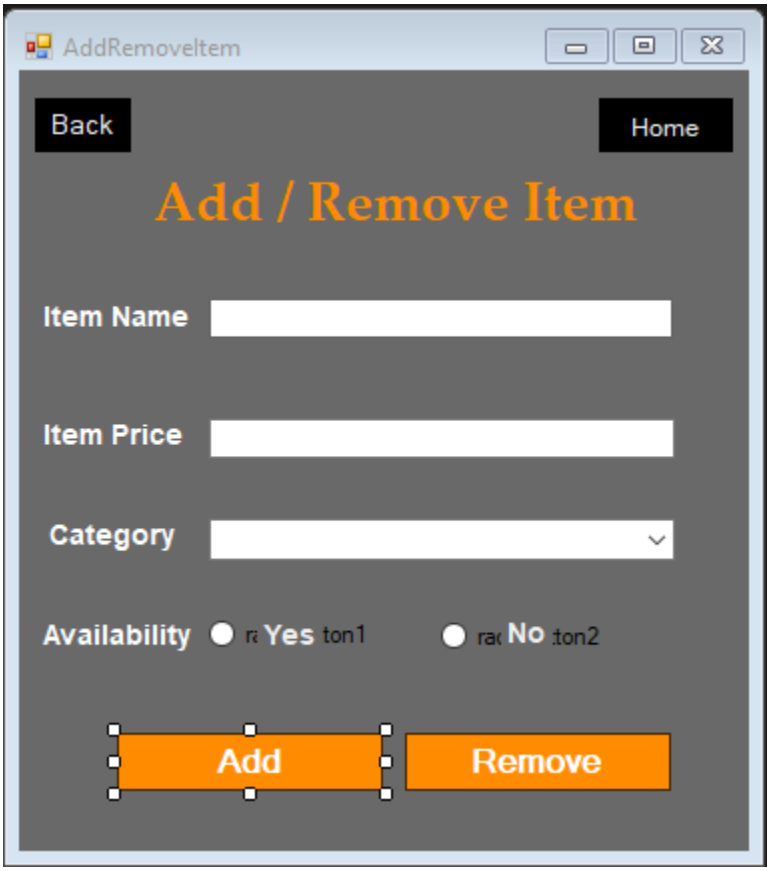
AddRemove Item

Streamlined process to add new items to the inventory.

Enter item details, including name, category, quantity, and expiration date (if applicable)

Easily remove items that are no longer stocked or offered by the cafe.

Confirm the removal action for seamless inventory management.

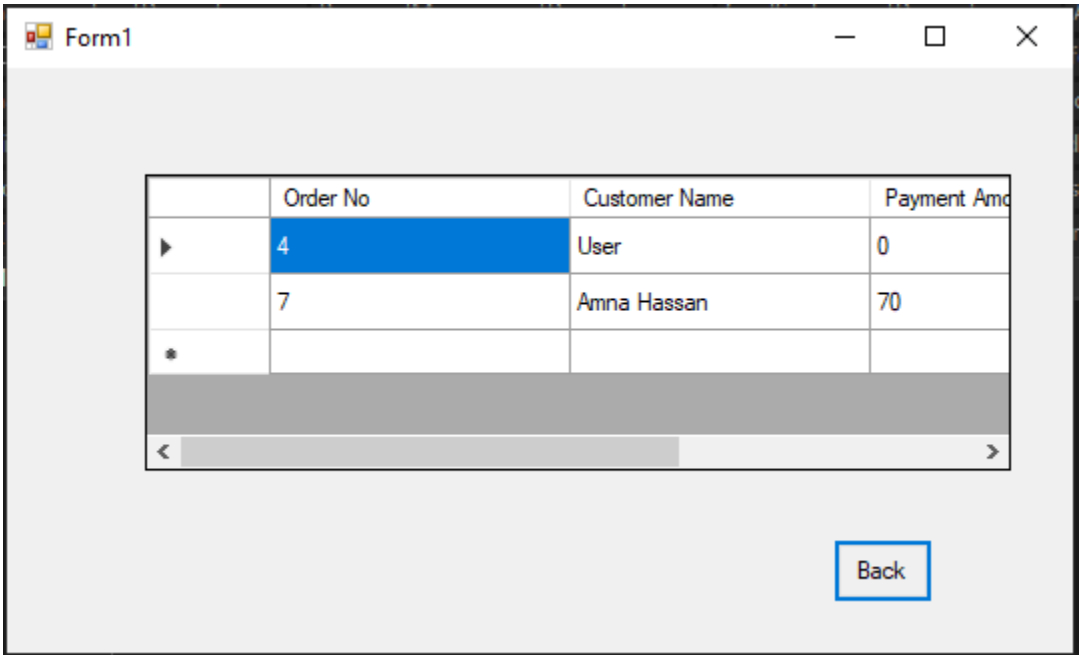


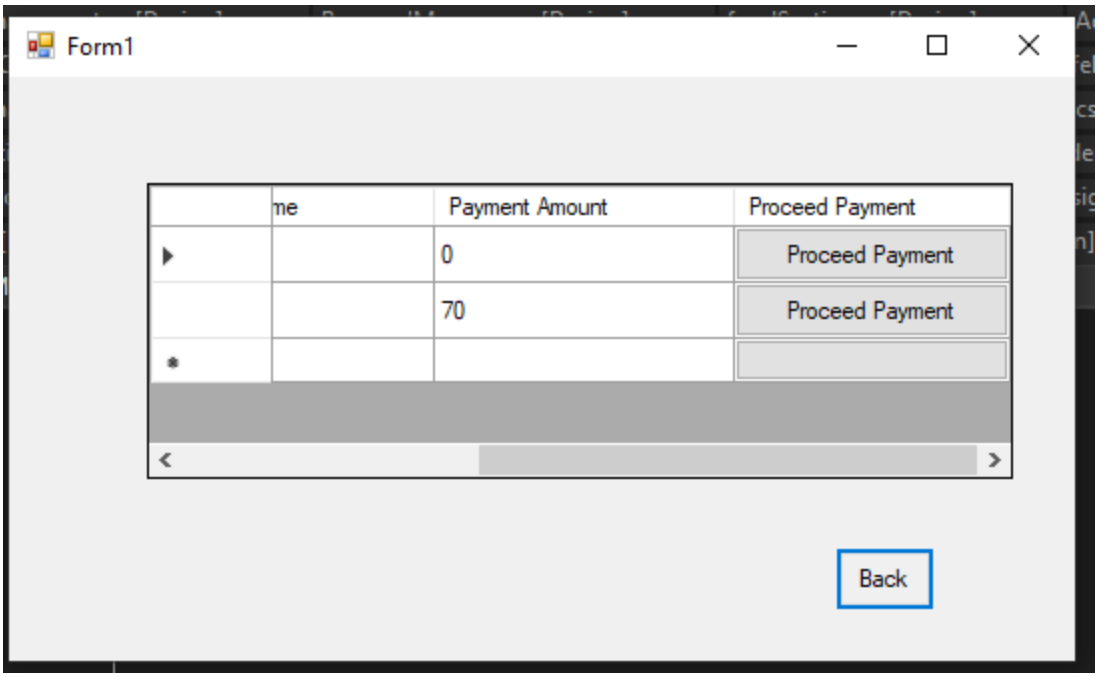
Cashier Dashnoard

Cashier can process payments of order and handle sales registers and records

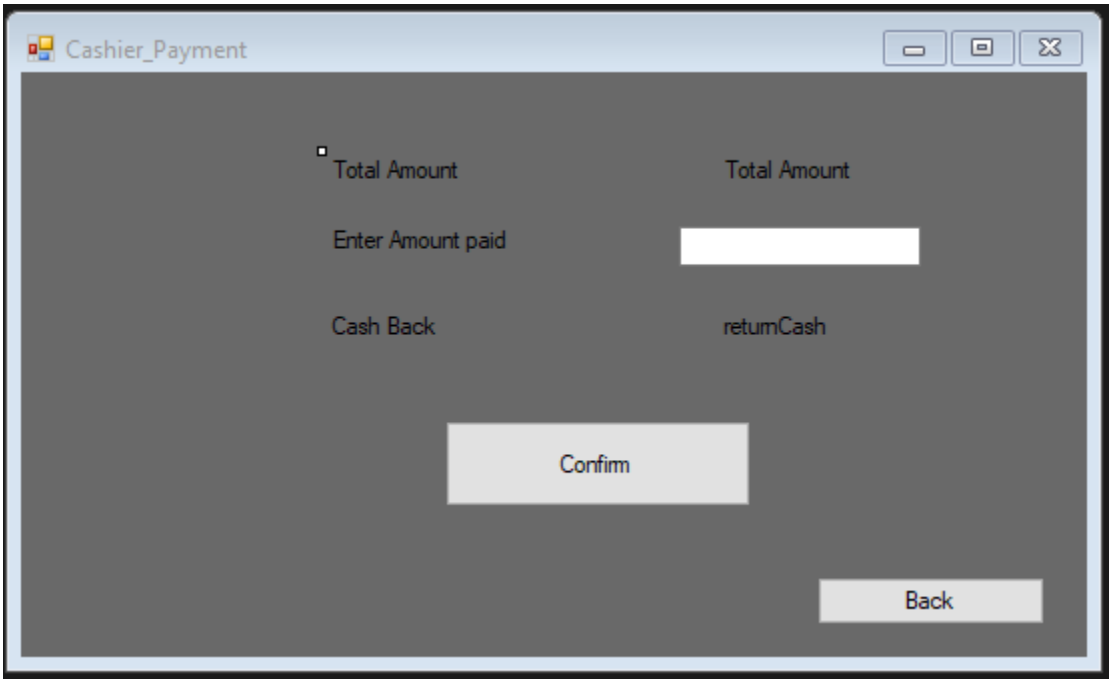
Process payments

Select the Order to proceed

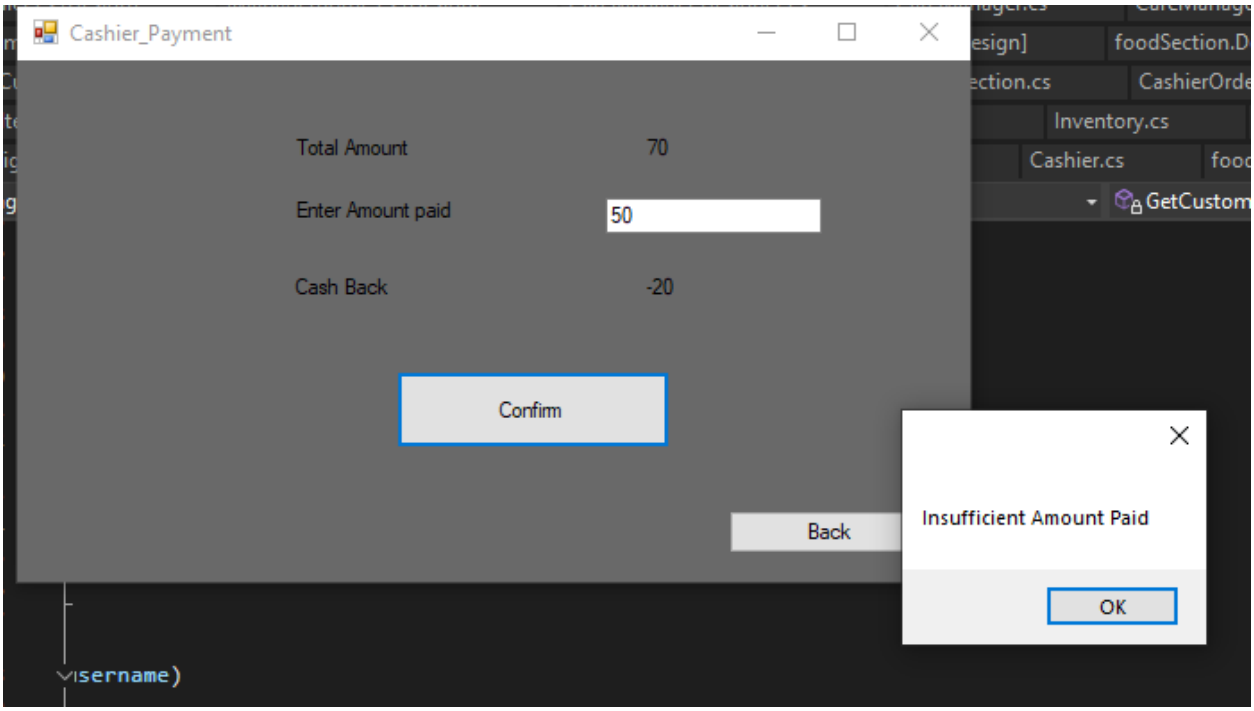




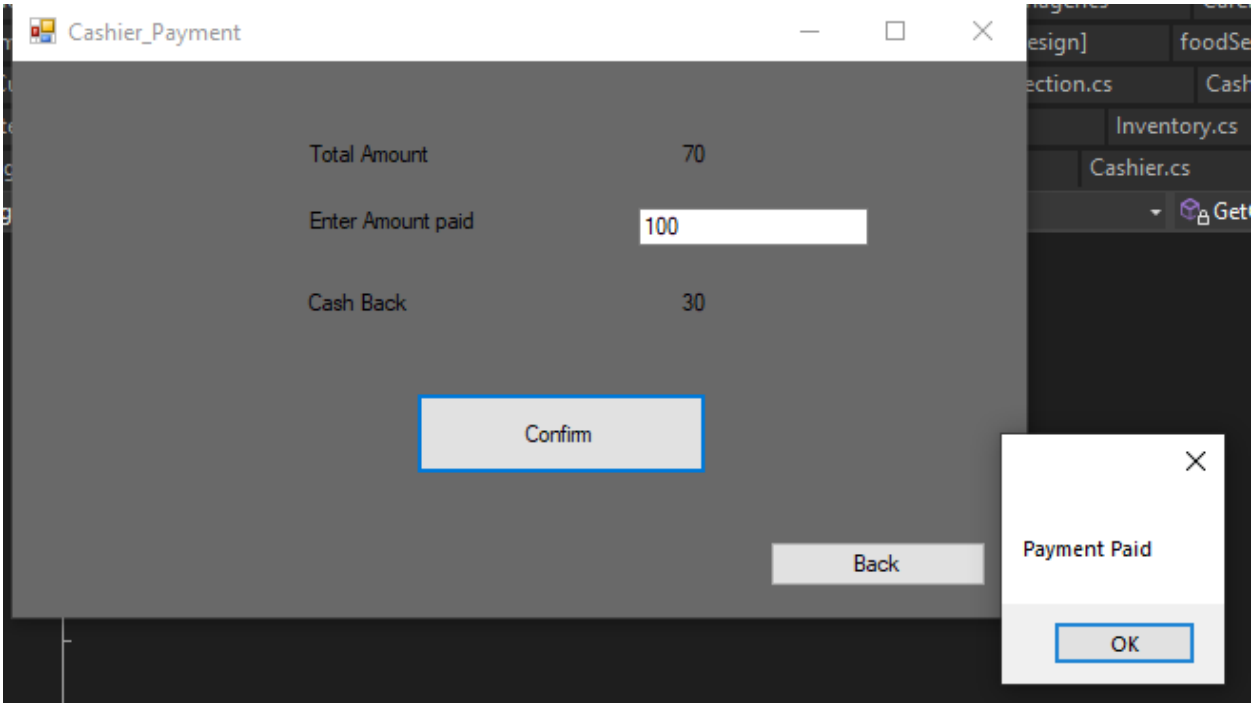
Process the payment from customer (card, cash or digital wallet). Calculate the final and return cash. Update the order status and return the remaining amount to the cahier



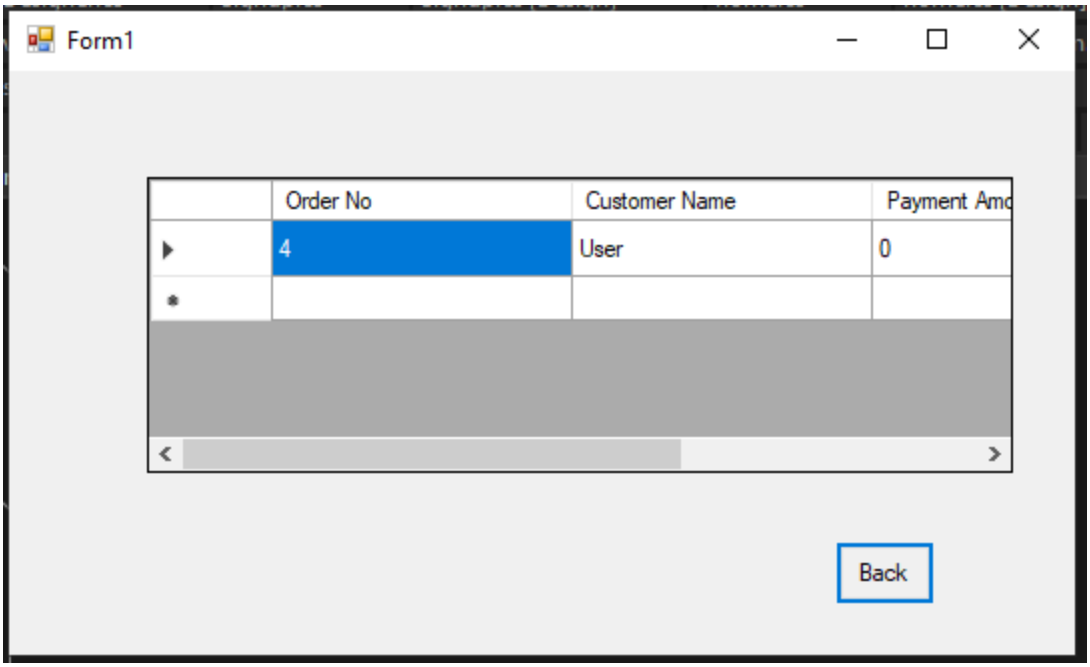
When the Amount paid is insufficient the following Screen is shown



When Sufficient amount is paid the order is cleared

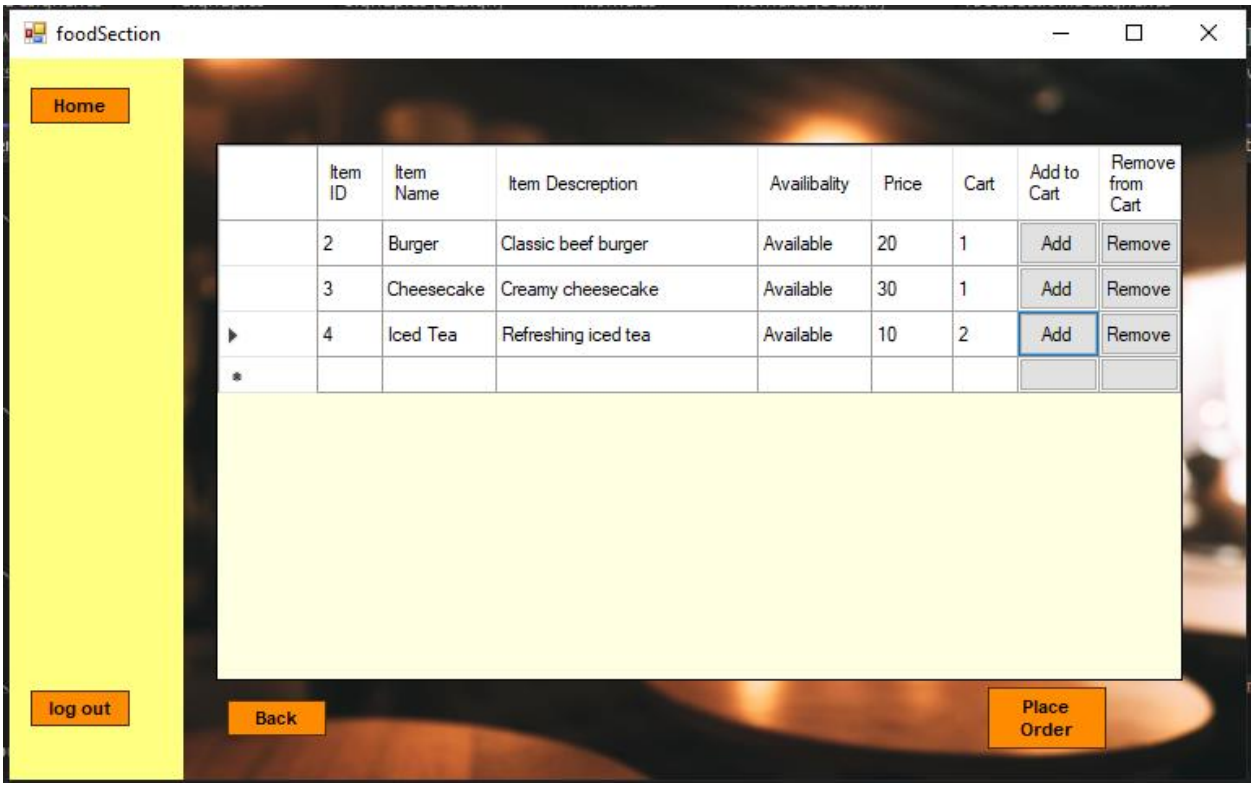


The order status is upated when the payment is done

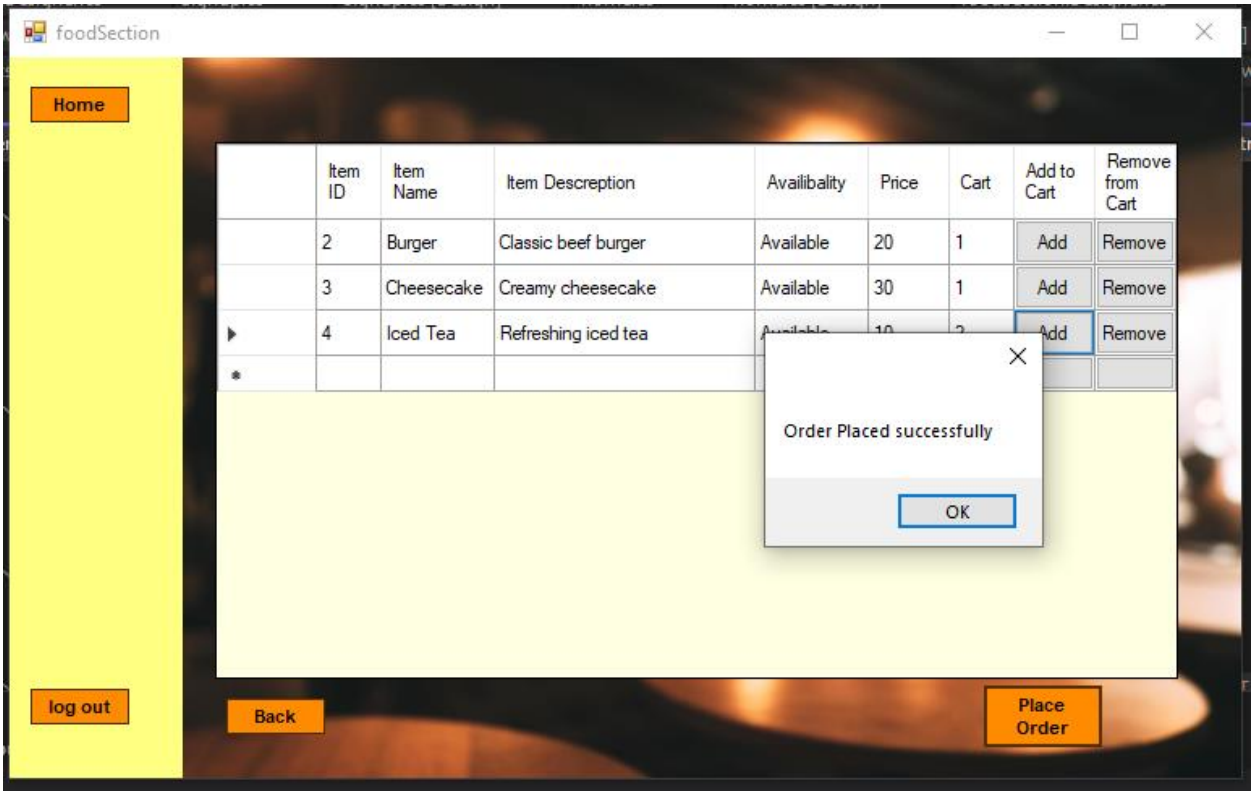


Customers View

When customers log in, it displays a list of all the items. Customers can add or delete items from their cart and then place an order.



When the order is placed..



SQL Queries

Database Creation Queries

```
--table for shedule
create table Shedule(
SheduleID int identity(1,1) primary key,
shiftType varchar(10),
StartingTime time,
EndingTime time,

);


--table for Cashier
create table Cashier(
CashierID int identity(1,1) primary key,
CashierName varchar(30),
Username varchar(10),
CashierContactno varchar(11),
Password varchar(10),
CashierEmail varchar(50),
--ControllerCashRegistered int not null,
--foreign key (ControllerCashRegistered) references CashRegister(RegisterID)
);

alter table Cashier
```

```
add CashierEmail varchar(50)


--table for Cafe Manager
create table CafeManager(
CManagerID int identity(1,1) primary key,
CManagerName varchar(30),
Username varchar(20),
CManagerEmail varchar(20),
CManagerContactno varchar(11),
Password varchar(10),

);


--tabke for Inventory Manager
create table InventoryManager(
IManagerID int identity(1,1) primary key,
IManagerName varchar(20),
Username varchar(20),
IMContactno varchar(11),
Password varchar(10),
Email VARCHAR(30)
);

create table Customer(
CustomerId int identity(1,1)primary key,
FullName varchar(15),
```

```
Email varchar(20),
Password varchar(10),
ContactNo varchar(11),
Username varchar(20) Unique,
);


--table for Orders
create table Orders(
OrderID int identity(1,1) primary key,
Customerid int NOT NULL,
IManagerId int NOT NULL,
payment_status varchar(20) DEFAULT NULL,
foreign key (Customerid) references Customer(CustomerId),
foreign key (IManagerId) references InventoryManager(IManagerID)
);
alter table Orders
add foreign key (ItemsListID) references OrderedItems(ID);


--table for rating and reviews
create table RatingandReviews(
RatingID int identity(1,1) PRIMARY KEY,
Customerid int NOT NULL,
```

```
Rating int,
Review varchar(30),

foreign key (Customerid) references Customer(CustomerId),

);


--table for Payment
create table Payment(
PaymentID int identity(1,1) PRIMARY KEY,
OrderId int NOT NULL,
Amount int,
AmountPaid int,
CashBacktocustomer int,
PaymentDate datetime,
foreign key (OrderId) references Orders(OrderID),
);

--table for Cash Register
create table CashRegister(
paymentID int,
foreign key (paymentID) references Payment(PaymentID),
);


--table for Staff
create table Staff(
```

```
SheduleId int not null,
CafeManagerId int not null,
InventoryManagerId int not null,
CashierId int not null,
foreign key (SheduleId) references Shedule(SheduleID),
foreign key (CafeManagerId) references CafeManager(CManagerID),
foreign key (InventoryManagerId) references InventoryManager(IManagerID),
foreign key (CashierId) references Cashier(CashierID),
);
drop table Staff
Select * from Staff;

--table for menu
create table Menu(
MenuID int identity(1,1) primary key,
MenuName varchar(20),
Description varchar(50),
CafeManagerId int not null,
foreign key (CafeManagerId) references CafeManager(CManagerID)
);


--table for Categories
create table Categories(
CategoryID int not null,
CategoryName varchar(30),
CategoryDes varchar(50),
menuId int not null,
```

```
primary key(CategoryID),
foreign key (menuId) references Menu(MenuID)
);

--table for pricing of item
create table Pricing(
PricingID int identity(1,1) primary key,
BasePrice int,
ControlBymanager int not null,
foreign key (ControlBymanager) references CafeManager(CManagerID)
);

--table for items in cafe
create table Items(
ItemID int identity(1,1) primary key,
ItemName varchar(15),
ItemDes varchar(30),
ItemAvailibility varchar(10),
priceid int not null,
quantity int,

foreign key (priceid) references Pricing(PricingID)
);

-- Drop the foreign key constraint
ALTER TABLE Items
add quantity int;
```

```
ALTER TABLE Items
DROP COLUMN ItemID;
ALTER TABLE Items
ADD ItemID int identity(1,1) primary key;
ALTER TABLE Items
ADD FOREIGN KEY (priceid) REFERENCES Pricing(PricingID);

create table CategoriesHasItems(
    ItemID int not null,
    CategoryID int not null,
    primary key(ItemID, CategoryID),
    foreign key (ItemID) references Items(ItemID),
    foreign key (CategoryID) references Categories(CategoryID)
);


--table for MenuHasCategories
create table MenuHasCategories(
menuId int not null,
categoryId int not null,
foreign key (menuId) references Menu(MenuID),
foreign key (categoryId) references Categories(CategoryID)
);

--6 Finding the Customers who have rated items with a rating higher than the average rating
SELECT FullName
FROM Customer
WHERE CustomerId IN (
```

```
    SELECT Customerid
    FROM RatingandReviews
    WHERE Rating > (
        SELECT AVG(Rating)
        FROM RatingandReviews
    )
);


--table for cafe
create table Cafe(
CafeID int not null,
CafeName varchar(30),
CafeLocation varchar(30),
CafeManagerId int not null,
InventoryManagerId int not null,
CashierId int not null,
ServeMenuid int not null,
Description varchar(50),
primary key(CafeID),
foreign key (CafeManagerId) references CafeManager(CManagerID),
foreign key (InventoryManagerId) references InventoryManager(IManagerID),
foreign key (ServeMenuid) references Menu(MenuId),
);


--table for Inventory
create table Inventory(
```

```
InventoryID int identity(1,1) primary key,
InventoryName varchar(30),
InventoryManagerid int not null,
MinimumStockLevel int,
QuantityInHand int,
ExpirationDate date,

foreign key (InventoryManagerid) references InventoryManager(IManagerID)
);


create table OrderedItems( --
OrderID int,
itemID int,
quantity int,
foreign key (itemID) references items(itemID),
foreign key (OrderID) references Orders(OrderID),
Primary key(OrderID,itemID)
);

alter table OrderedItems
add foreign key (itemID) references items(itemid)
```

Database Insertion Queries

```
INSERT INTO Customer (FullName, Email, Password, ContactNo, Username)
VALUES ('Amna Hassan', 'amnahsn@gmail.com', '1234', '03134556364', 'Amna');
SELECT * FROM Customer;
```

```
INSERT INTO InventoryManager (IManagerName, Username, IMContactno, Password, Email)
VALUES ('Shuja Uddin', 'Shuja', '12654866543', '1234', 'shujauddin@gmail.com');
SELECT * FROM InventoryManager;

INSERT INTO CafeManager VALUES ('Ali hamza', 'Ali', 'alihamza@gmail.com',
'03155148556', '1234');
SELECT * FROM CafeManager;

INSERT INTO Cashier (CashierName, Username, CashierContactno, Password, CashierEmail)
VALUES
    ('Hassan', 'hassan', '1234567890', '1234', 'hassan@example.com');
SELECT * FROM Cashier;

Delete from Cashier where CashierID=2;

INSERT INTO Orders (Customerid, IManagerId)
VALUES (1, 1);
Select * From Orders;

INSERT INTO Payment (OrderId, Amount, AmountPaid, CashBacktocustomer)
VALUES (1, 150.00, 100.00, 0);
SELECT * FROM payment;

-- Insert data into Menu table
insert into Menu (MenuName, Description, CafeManagerId) values
('Main Menu', 'Main menu items', 1);
SELECT * FROM Menu;
```

```
-- Insert data into Categories table
insert into Categories (CategoryID,CategoryName, CategoryDes, menuId) values
(1,'Food', 'Delicious main courses', 1),
(2,'Desserts', 'Sweet treats for dessert', 1),
(3,'Beverages', 'Refreshing drinks', 1);
SELECT * FROM categories;


-- Insert data into Items table
insert into Items (ItemName, ItemDes, ItemAvailibility, priceid,quantity) values
('Burger', 'Classic beef burger', 'Available', 1,10),
('Cheesecake', 'Creamy cheesecake', 'Available', 2,20),
('Iced Tea', 'Refreshing iced tea', 'Available', 3,30);
select * from Items

INSERT INTO Pricing (BasePrice, ControlByManager)
VALUES (20, 1);
INSERT INTO Pricing (BasePrice, ControlByManager)
VALUES (30, 1),(10,1);
SELECT * FROM Pricing;

-- Insert valid data
INSERT INTO CategoriesHasItems (ItemID, CategoryID)
VALUES
    (2, 1),
    (3, 2),
    (4, 3);
```

```
SELECT * FROM CategoriesHasItems;


INSERT INTO MenuHasCategories (menuID,categoryID)
values(1,1),(1,2),(1,3);

select * from Menu;
select * from Categories;

-- Insert invalid data (CategoryID not 1, 2, or 3)
INSERT INTO CategoriesHasItems (ItemID, CategoryID)
VALUES
    (4, 4),
    (5, 1),
    (6, 6);


INSERT INTO Shedule (shiftType, StartingTime, EndingTime)
VALUES ('Morning', '08:00:00', '16:00:00');

INSERT INTO Staff (SheduleId, CafeManagerId, InventoryManagerId, CashierId)
VALUES (1,1 ,1 ,3);

INSERT INTO RatingandReviews (CustomerID, Rating, Review)
VALUES
   (1, 4, 'Great service!'),
(2, 5, 'Amazing food!');
delete from RatingandReviews where CustomerID=2;
```

```
select * from RatingandReviews;

INSERT INTO Inventory (InventoryName, InventoryManagerid, MinimumStockLevel,
QuantityInHand, ExpirationDate)
VALUES ('Item A', 1, 10, 20, '2024-05-01');
```

MultiTable Joins

--4 Table Joins

--1 Joining Menu, Categories, Items, and Pricing:

```
SELECT m.MenuName, cat.CategoryName, i.ItemName, p.BasePrice
FROM Menu m
INNER JOIN Categories cat ON m.MenuID = cat.menuId
INNER JOIN CategoriesHasItems chi ON cat.CategoryID = chi.CategoryID
INNER JOIN Items i ON chi.ItemID = i.ItemID
INNER JOIN Pricing p ON i.priceid = p.PricingID;
```

--2 Joining Orders, Customer, Payment, and Items:

```
SELECT o.OrderID, c.FullName AS CustomerName, p.AmountPaid, i.ItemName
FROM Orders o
INNER JOIN Customer c ON o.Customerid = c.Customerid
LEFT JOIN Payment p ON o.OrderID = p.OrderID
INNER JOIN Items i ON o.Customerid = i.ItemID;
```

```
SELECT o.OrderID, c.FullName AS CustomerName, p.AmountPaid, i.ItemName
FROM Orders o
INNER JOIN Customer c ON o.Customerid = c.Customerid
LEFT JOIN Payment p ON o.OrderID = p.OrderID
INNER JOIN Items i ON o.Customerid = i.ItemID;
```

--3 Selecting order ID, customer's full name, and calculating total payment amount for orders with pending status

```
SELECT O.OrderID, C.FullName AS CustomerName, SUM(P.BasePrice * OI.quantity) AS PaymentAmount
FROM Orders O JOIN Customer C ON O.Customerid = C.CustomerId
LEFT JOIN OrderedItems OI ON O.OrderID = OI.OrderID
LEFT JOIN Items I ON OI.itemID = I.ItemID
LEFT JOIN Pricing P ON I.priceid = P.PricingID
WHERE O.payment_status = 'Pending' GROUP BY O.OrderID, C.FullName;
```

--3 Table Joins

--1 Total Amount Paid by Each Customer:

```
SELECT o.Customerid, c.FullName, SUM(p.AmountPaid) AS TotalAmountPaid
FROM Orders o
INNER JOIN Payment p ON o.OrderID = p.OrderID
INNER JOIN Customer c ON o.Customerid = c.Customerid
GROUP BY o.Customerid, c.FullName;
```

--2 List of Categories and Their Menu Items:

```
SELECT cat.CategoryName, i.ItemName
```

```
FROM Categories cat
INNER JOIN CategoriesHasItems chi ON cat.CategoryID = chi.CategoryID
INNER JOIN Items i ON chi.ItemID = i.ItemID;
```

--3 Orders Placed by a Specific Customer:

```
SELECT o.OrderID, o.payment_status, oi.ItemID, i.ItemName, oi.Quantity
FROM Orders o
INNER JOIN OrderedItems oi ON o.OrderID = oi.OrderID
INNER JOIN Items i ON oi.ItemID = i.ItemID
WHERE o.Customerid = 2; -- write desired CustomerID
```

--4 Details of Items with Their Categories:

```
SELECT i.ItemName, c.CategoryName
FROM Items i
INNER JOIN CategoriesHasItems chi ON i.ItemID = chi.ItemID
INNER JOIN Categories c ON chi.CategoryID = c.CategoryID;
```

--5 Cashiers with Their Shifts:

```
SELECT c.CashierName, sh.shiftType, sh.StartingTime, sh.EndingTime
FROM Cashier c
INNER JOIN Staff s ON c.CashierID = s.CashierId
INNER JOIN Shedule sh ON s.SheduleId = sh.SheduleID;
```

--6Inventory Manager with Their Shifts:

```
SELECT c.IManagerName, sh.shiftType, sh.StartingTime, sh.EndingTime
FROM InventoryManager c
```

```
INNER JOIN Staff s ON c.IManagerID = s.InventoryManagerId
INNER JOIN Shedule sh ON s.SheduleId = sh.SheduleID;
```

--7 Cafe Manager with Their Shifts:

```
SELECT c.CManagerName, sh.shiftType, sh.StartingTime, sh.EndingTime
FROM CafeManager c
INNER JOIN Staff s ON c.CManagerID = s.CafeManagerId
INNER JOIN Shedule sh ON s.SheduleId = sh.SheduleID;
```

--8 SQL query to retrieve items for a specific category with quantity

```
SELECT Items.ItemID, Items.ItemName, Items.ItemDes, Items.ItemAvailibility, Items.priceid,
Items.quantity
FROM Items
INNER JOIN CategoriesHasItems ON Items.ItemID = CategoriesHasItems.ItemID
INNER JOIN Categories ON CategoriesHasItems.CategoryID = Categories.CategoryID
WHERE Categories.CategoryName = 'Desserts'; --Beverages  / Food / Desserts
```

--2 Table Joins

--1 List of Orders with Customer Information:

```
SELECT o.OrderID, c.FullName AS CustomerName, o.Payment_Status
FROM Orders o
INNER JOIN Customer c ON o.Customerid = c.Customerid;
```

--2 Details of Items Ordered in a Specific Order:

```
SELECT oi.OrderID, i.ItemName, oi.Quantity
FROM OrderedItems oi
INNER JOIN Items i ON oi.ItemID = i.ItemID
WHERE oi.OrderID = 4; -- Replace 1 with the desired OrderID

--3 Menu Items with Their Prices:
SELECT i.ItemName, p.BasePrice
FROM Items i
INNER JOIN Pricing p ON i.priceid = p.PricingID;

--4Total Amount Paid for Each Order:
SELECT o.OrderID, SUM(p.AmountPaid) AS TotalAmountPaid
FROM Orders o
INNER JOIN Payment p ON o.OrderID = p.OrderID
GROUP BY o.OrderID;
```

Views

--View 1: Available Items and Their Prices

```
CREATE VIEW AvailableItemsWithPrices AS
SELECT i.ItemID, i.ItemName, i.ItemAvailibility, p.BasePrice
FROM Items i
INNER JOIN Pricing p ON i.PriceID = p.PricingID;

SELECT * FROM AvailableItemsWithPrices;
```

--View 2: Orders with Customer Information

```
CREATE VIEW OrdersWithCustomers AS
SELECT o.OrderID, o.Payment_Status, c.FullName AS CustomerName, c.ContactNo AS CustomerContact
FROM Orders o
INNER JOIN Customer c ON o.CustomerID = c.CustomerID;

SELECT * FROM OrdersWithCustomers;
```

--View 3: Cashiers and Their Contact Information

```
CREATE VIEW CashiersContactInfo AS
SELECT CashierID, CashierName, CashierContactNo
FROM Cashier;

SELECT * FROM CashiersContactInfo;
```

--View 4: Menu Categories with Items

```
CREATE VIEW MenuCategoriesWithItems AS
SELECT c.CategoryID, c.CategoryName, i.ItemName
FROM Categories c
INNER JOIN CategoriesHasItems ci ON c.CategoryID = ci.CategoryID
INNER JOIN Items i ON ci.ItemID = i.ItemID;

SELECT * FROM MenuCategoriesWithItems;
```

--view 5: Deatils of each order

```
CREATE VIEW OrderDetails AS
SELECT o.OrderID, c.FullName AS CustomerName, i.ItemName, oi.Quantity
FROM Orders o
```

```
INNER JOIN Customer c ON o.CustomerID = c.CustomerID
INNER JOIN OrderedItems oi ON o.OrderID = oi.OrderID
INNER JOIN Items i ON oi.ItemID = i.ItemID;

SELECT * FROM OrderDetails;
```

--View 6: Deatils of complete staff

```
CREATE VIEW AllStaffDetails AS
SELECT 'Cafe Manager' AS Role, CManagerID AS StaffID, CManagerName AS StaffName, CManagerEmail AS StaffEmail, CManagerContactno AS StaffContact
FROM CafeManager
UNION ALL
SELECT 'Inventory Manager' AS Role, IManagerID AS StaffID, IManagerName AS StaffName, Email AS StaffEmail, IMContactno AS StaffContact
FROM InventoryManager
UNION ALL
SELECT 'Cashier' AS Role, CashierID AS StaffID, CashierName AS StaffName, CashierEmail AS StaffEmail, CashierContactno AS StaffContact
FROM Cashier;

SELECT * FROM AllStaffDetails;
```

Nested SubQueries

--1 Customers Who Have Not Given Reviews

```
SELECT FullName
FROM Customer
WHERE CustomerID NOT IN (
```

```
    SELECT DISTINCT CustomerID
    FROM RatingandReviews
);
```

--2 Finding Customers who have placed orders managed by a specific Inventory Manager

```
SELECT FullName
FROM Customer
WHERE CustomerId IN (
    SELECT Customerid
    FROM Orders
    WHERE IManagerId = (
        SELECT IManagerID
        FROM InventoryManager
        WHERE IManagerName = 'Shuja Uddin'
    )
);
```

--3 Finding Inventory Managers who are managing inventories with a quantity below the minimum stock level

```
SELECT IManagerName
FROM InventoryManager
WHERE IManagerID IN (
    SELECT InventoryManagerid
    FROM Inventory
    WHERE QuantityInHand < MinimumStockLevel
);
```

--4 Finding the Inventory Manager who manages the inventory with the earliest expiration date

```
SELECT IManagerName
FROM InventoryManager
WHERE IManagerID = (
    SELECT InventoryManagerid
    FROM Inventory
    WHERE ExpirationDate = (
        SELECT MIN(ExpirationDate)
        FROM Inventory
    )
);
```

--5 Finding the Menu items that belong to a specific Category managed by a particular Cafe Manager

```
SELECT MenuName
FROM Menu
WHERE CafeManagerId = (
    SELECT CManagerID
    FROM CafeManager
    WHERE CManagerName = 'Ali Hamza'
)
AND MenuID IN (
    SELECT MenuID
    FROM MenuHasCategories
    WHERE CategoryID IN (
        SELECT CategoryID
        FROM Categories
```

```
        WHERE CategoryName = 'Beverages'
    )
);

select * from MenuHasCategories
select * from Menu
```

Aggregate and group by Queries (withhaving)

-- 1: Total Quantity of Items Sold per Order

```
SELECT OrderID, SUM(quantity) AS TotalQuantity
FROM OrderedItems
GROUP BY OrderID;
```

--2 Count of Orders Placed by Each Customer

```
SELECT Customerid, COUNT(OrderID) AS TotalOrders
FROM Orders
GROUP BY Customerid
HAVING COUNT(OrderID) >= 0;
```

--3 Total Revenue Generated by Each Menu Category

```
SELECT c.CategoryName, SUM(i.quantity * p.BasePrice) AS TotalRevenue
FROM Categories c
JOIN CategoriesHasItems chi ON c.CategoryID = chi.CategoryID
JOIN Items i ON chi.ItemID = i.ItemID
JOIN Pricing p ON i.priceid = p.PricingID
JOIN OrderedItems oi ON i.ItemID = oi.itemID
GROUP BY c.CategoryName
```

```
HAVING SUM(i.quantity * p.BasePrice) > 0;

select * from Payment;
```

--4 Average rating by each customer

```
SELECT CustomerId, AVG(Rating) AS AverageRating
FROM RatingandReviews
GROUP BY CustomerId
HAVING AVG(Rating) > 3.5;
```

--5 Total Revenue per Customer with Total Exceeding x

```
SELECT o.Customerid, SUM(p.AmountPaid) AS TotalRevenue
FROM Orders o
JOIN Payment p ON o.OrderID = p.OrderId
GROUP BY o.Customerid
HAVING SUM(p.AmountPaid) > 0; --x
```

Procedures

--Procedure to check inventory Level

```
CREATE PROCEDURE CheckInventoryLevels
AS
BEGIN
   DECLARE @ItemId INT;
   DECLARE @CurrentStock INT;
   DECLARE @MinStockLevel INT;
   DECLARE @ManagerEmail VARCHAR(100);
```

```
    DECLARE cur CURSOR FOR
    SELECT InventoryID, QuantityInHand, MinimumStockLevel, Email
    FROM Inventory INNER JOIN InventoryManager ON Inventory.InventoryManagerid =
InventoryManager.IManagerID;

    OPEN cur;
    FETCH NEXT FROM cur INTO @ItemId, @CurrentStock, @MinStockLevel,
@ManagerEmail;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @CurrentStock < @MinStockLevel
        BEGIN
            -- Send alert to Inventory Manager
            EXEC SendInventoryAlert @ItemId, @ManagerEmail;
        END;

        FETCH NEXT FROM cur INTO @ItemId, @CurrentStock, @MinStockLevel,
@ManagerEmail;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
```

--Procedure to Send Email for inventory alert

```
CREATE PROCEDURE SendInventoryAlert
```

```
    @ItemId INT,
    @ManagerEmail VARCHAR(100)
AS
BEGIN
    DECLARE @Subject NVARCHAR(255);
    DECLARE @Body NVARCHAR(MAX);

    SELECT @Subject = 'Alert: Low Inventory',
           @Body = 'Item with ID ' + CAST(@ItemId AS VARCHAR(10)) + ' has fallen below the minimum stock level.';

    -- Send email notification
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'Cafe System',
        @recipients = @ManagerEmail,
        @subject = @Subject,
        @body = @Body;
END;
```

Triggers

-- Create a trigger named CheckAmountPaid

```
CREATE TRIGGER CheckAmountPaid
ON Payment
AFTER INSERT
AS
BEGIN
```

```
    -- Check if AmountPaid is less than Amount
    IF (SELECT COUNT(*) FROM inserted WHERE AmountPaid < Amount) > 0
    BEGIN
        -- Perform actions when AmountPaid is less than Amount
        print('AmountPaid cannot be less than Amount');
        ROLLBACK; -- Rollback the transaction to prevent the invalid data from being inserted
    END
END;
```

-- Create a trigger after insert on the Pricing table

```
CREATE TRIGGER tr_PriceInserted
ON Pricing
AFTER INSERT
AS
BEGIN
    DECLARE @BasePrice int;
    DECLARE @ControlByManager int;

    SELECT @BasePrice = BasePrice, @ControlByManager = ControlByManager
    FROM inserted;

    -- Your logic here to handle the newly inserted base price
    -- For example, you can log the information or perform additional actions

    PRINT 'New base price inserted. BasePrice: ' + CAST(@BasePrice AS varchar(10)) +
        ', ControlByManager: ' + CAST(@ControlByManager AS varchar(10));
```

```
END;
```

-- Create a trigger after insert on the CategoriesHasItems table

```
CREATE TRIGGER tr_InvalidCategory
ON CategoriesHasItems
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @InvalidCategoryCount int;

    -- Check if there are any invalid CategoryIDs (not 1, 2, or 3)
    SELECT @InvalidCategoryCount = COUNT(*)
    FROM inserted
    WHERE CategoryID NOT IN (1, 2, 3);

    -- If there are invalid CategoryIDs, perform the desired action
    IF @InvalidCategoryCount > 0
    BEGIN
        -- Your logic here, for example, you can log an error message or rollback the transaction
        -- For demonstration purposes, we are rolling back the transaction and printing a message
        ROLLBACK;
        PRINT 'Invalid CategoryID detected. Rollback initiated.';
```

```
    END
END;
```

--checking order status is valid or not

```
CREATE TRIGGER EnforcePaymentStatusTrigger
ON Orders
AFTER INSERT
AS
BEGIN
    IF EXISTS (SELECT * FROM inserted WHERE payment_status NOT IN ('Pending', 'Processing', 'Completed'))
    BEGIN
        RAISERROR('Invalid payment status. Payment status must be "Pending", "Processing", or "Completed".', 16, 1);
        ROLLBACK TRANSACTION;
    END;
END;
```

--prevent deletion of inventory manager

```
CREATE TRIGGER PreventInventoryManagerDeleteTrigger
ON InventoryManager
INSTEAD OF DELETE
AS
BEGIN
    RAISERROR ('Deletion from InventoryManager table is not allowed.', 16, 1);
    ROLLBACK TRANSACTION;
END;
```

```
delete from InventoryManager where IManagerID=1;
select * from InventoryManager
```