

***Mini project report on***

Online Grocery Management System

*Submitted in partial fulfilment of the requirements for the award of degree of*

**Bachelor of Technology**

**in**

**Computer Science & Engineering**

**UE21CS351 – DBMS Project**

***Submitted by:***

|  |  |
| --- | --- |
| **CHANDANA DONGRE**  **BHODHITHA K S** | **PES2UG21CS134**  **PES2UG21CS122** |

Under the guidance of

**Prof. Niveditha**

Assistant Professor

|  |
| --- |
| PES University |

**AUG - DEC 2023**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

FACULTY OF ENGINEERING

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India



**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

**CERTIFICATE**

*This is to certify that the mini project entitled*

**ONLINE GROCERY MANAGEMENT SYSTEM**

*is a bonafide work carried out by*

|  |  |
| --- | --- |
| **CHANDANA DONGRE**  **BHODHITHA K S** | **PES2UG21CS134**  **PES2UG21CS122** |

In partial fulfilment for the completion of fifth semester DBMS Project (UE20CSS301) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2022 – DEC. 2022. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

|  |  |
| --- | --- |
| Signature  Prof. Niveditha  Assistant Professor |  |

**DECLARATION**

We hereby declare that the DBMS Project entitled **Online Grocery Management System** has been carried out by us under the guidance of **Prof.Niveditha, Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2023.

|  |  |
| --- | --- |
| **CHANDANA DONGRE**  **BHODHITHA K S** | **PES2UG21CS134 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **PES2UG21CS122 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

**ACKNOWLEDGEMENT**

I would like to express my gratitude to Prof. Niveditha S, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE21CS351 - DBMS Project.

I take this opportunity to thank Dr. Sandesh B J, C, Professor,ChairPerson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

**ABSTRACT**

The Grocery Management System (GDMS) is a comprehensive database management system designed to streamline and optimize the operations of an online grocery store. In the modern era, the demand for online grocery shopping has witnessed significant growth, and the GDMS aims to provide an efficient platform for managing the complexities associated with grocery services.

The GDMS encompasses a relational database model, implemented using MySQL, to manage various aspects of the online grocery store, including customer data, product inventory, order processing and administrative functions. The system is designed to enhance user experience, improve order fulfilment efficiency, and provide valuable insights through data analytics.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
|  | **INTRODUCTION** | **7** |
|  | **PROBLEM DEFINITION** | **8** |
|  | **ER MODEL** | **9** |
|  | **ER TO RELATIONAL MAPPING** | **10** |
|  | **DDL STATEMENTS** | **11** |
|  | **DML STATEMENTS** | **19** |
|  | **QUERIES (SIMPLE QUERY AND UPDATE AND DELETE OPERATION, CORRELATED QUERY AND NESTED QUERY)** | **35** |
|  | **FRONT END DEVELOPMENT** | **39** |
|  | **CONCLUSION** | **43** |
| **10. REFERENCES/BIBLIOGRAPHY** | | **44** |
|  | |  |

# INTRODUCTION

Our Grocery Management System is a dynamic and user-friendly platform developed using Flask, PHP, and MySQL technologies. This project aims to revolutionize the online grocery shopping experience by seamlessly connecting customers, products, and services. With an intuitive user interface built on Flask and PHP, customers can effortlessly browse a comprehensive product catalog, place orders, and track deliveries in real-time. The MySQL database ensures efficient data management, allowing for secure storage of customer details, product inventory, and order information. This project not only simplifies the ordering process for users but also streamlines administrative tasks, providing valuable insights for business optimization. Experience the future of grocery shopping with our innovative and robust Grocery Management System.

# 

# 

# 2. PROBLEM DEFINITION

In the rapidly evolving landscape of retail, traditional grocery shopping faces challenges in meeting the growing demands for convenience and efficiency. Our project addresses the following problems:

1. **Inefficient Order Processing:** Traditional grocery shopping lacks a streamlined order processing system, leading to delays, order errors, and an overall inefficient customer experience.
2. **Limited Access to Product Information:** Customers often face challenges in accessing comprehensive and up-to-date information about the available products, hindering their ability to make informed purchasing decisions.
3. **Manual Delivery Management:** The absence of an automated delivery management system results in manual assignment of delivery personnel, leading to delays, miscommunication, and a lack of real-time tracking capabilities.
4. **Data Silos and Ineffective Analytics:** Traditional systems often lead to data silos, making it challenging for administrators to extract meaningful insights. Lack of analytics hampers strategic decision-making and optimization.
5. **Complex Administrative Tasks:** Administrative functions such as user management, inventory tracking, and order monitoring are often complex and time-consuming, limiting the scalability of the business.

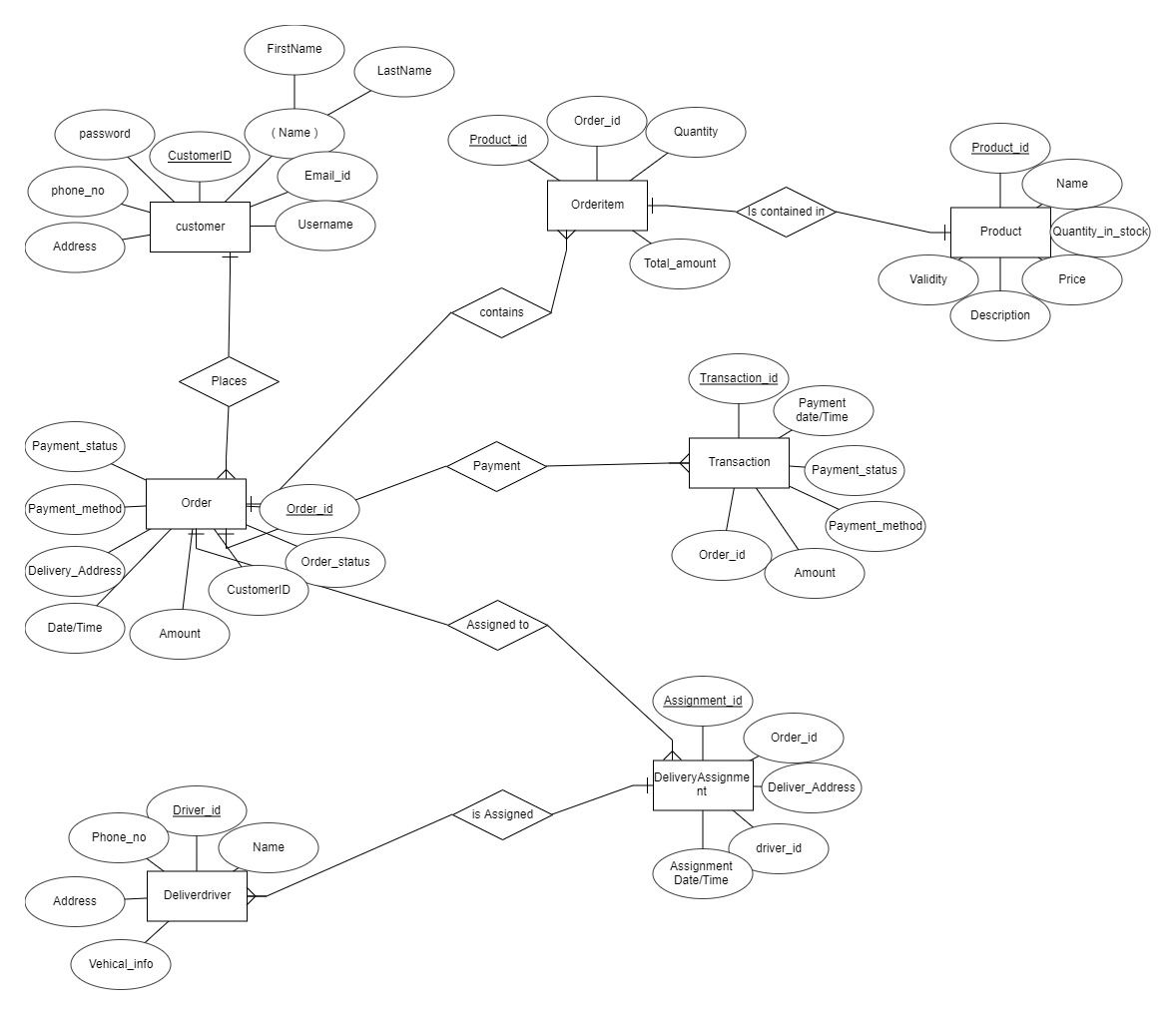
Our Grocery Management System aims to overcome these challenges by leveraging Flask, PHP, and MySQL to create a modern, efficient, and user-centric platform that transforms the online grocery shopping experience for both customers and administrators.

Top of Form

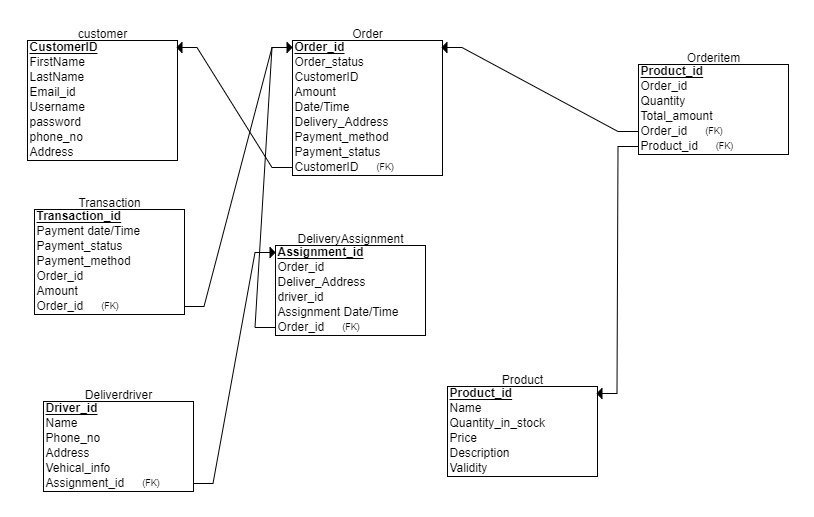
# 

# 

# 3. ER MODEL



# 4. ER TO RELATIONAL MAPPING



**4.1 STEPS OF ALGORITHM FOR CHOOSEN PROBLEM**

**Algorithmic Steps used for Online Grocery Management System:**

1. User Registration and Login:
   * Validate user registration details.
   * Authenticate users during login.
2. Product Catalog Management:
   * Maintain a product catalog with details such as name, description, price, and availability.
   * Ensure real-time updates for product availability.
3. Order Processing:
   * Allow users to browse the catalog and add products to the cart.
   * Implement a secure checkout process for order placement.
   * Update product inventory upon successful order placement.
4. Delivery Management:
   * Assign delivery personnel to orders based on location and availability.
   * Implement real-time tracking of order status and delivery personnel.
   * Notify users and delivery personnel about order updates.
5. User Dashboards:
   * Provide customers with a dashboard to view order history and track deliveries.
   * Admins should have dashboards for managing users, products, and orders.
6. Data Storage and Retrieval:
   * Use MySQL to store and retrieve data related to users, products, orders, and deliveries.
   * Ensure data integrity and security through proper database design.
7. Analytics and Reporting:
   * Implement analytics tools to generate reports on customer behavior, product popularity, and delivery performance.
   * Provide administrators with insights to optimize business processes.
8. Security Measures:
   * Implement secure practices for user authentication and data transmission.
   * Protect against common security threats, such as SQL injection and cross-site scripting.
9. Error Handling:
   * Include robust error-handling mechanisms to gracefully manage exceptions and provide informative error messages.
10. User Interface (UI) Design:
    * Design a user-friendly interface using Flask and PHP for seamless interaction.
    * Ensure responsiveness across devices for a consistent user experience.
11. Deployment and Testing:
    * Deploy the system on a secure web server.
    * Conduct thorough testing, including unit testing, integration testing, and user acceptance testing.
12. Documentation:
    * Provide comprehensive documentation for users and administrators, including user manuals and system architecture documentation.

These steps outline the core functionality of the Grocery Management System algorithm. The actual implementation would involve translating these steps into code, integrating external libraries as needed, and conducting rigorous testing to ensure a robust and user-friendly system.

Top of Form

**4.2 COMPLETE DIAGRAM OF RELATIONAL MAPPING**

A diagram of a company

Description automatically generated

# 

# 5. DDL STATEMENTS

**STATEMENTS WITH SCREEN SHOTS OF THE TABLE CREATION**

**.** **Tables created**

CREATE TABLE Admins (

    AdminID INT PRIMARY KEY,

    FirstName VARCHAR(255),

    LastName VARCHAR(255),

    EmailID VARCHAR(255) UNIQUE,

    Username VARCHAR(255),

    Password VARCHAR(255),

    PhoneNo VARCHAR(20),

    Address VARCHAR(255)

);

CREATE TABLE Customer (

    CustomerID INT PRIMARY KEY,

    FirstName VARCHAR(255),

    LastName VARCHAR(255),

    EmailID VARCHAR(255) UNIQUE,

    Username VARCHAR(255),

    Password VARCHAR(255),

    PhoneNo VARCHAR(20),

    Address VARCHAR(255)

);

CREATE TABLE Orders (

    OrderID INT PRIMARY KEY,

    PaymentMethod VARCHAR(50),

    PaymentStatus VARCHAR(50),

    DeliveryAddress VARCHAR(255),

    CustomerID INT,

    Amount DECIMAL(10, 2),

    OrderDateTime TIMESTAMP,

    OrderStatus VARCHAR(50),

    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

CREATE TABLE Product (

    ProductID INT PRIMARY KEY,

    ProductName VARCHAR(255),

    Price DECIMAL(10, 2),

    QuantityInStock INT,

    Description TEXT

);

CREATE TABLE OrderItem (

    OrderItemID INT PRIMARY KEY,

    OrderID INT,

    Quantity INT,

    ProductID INT,

    TotalAmount DECIMAL(10, 2),

    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)

);

CREATE TABLE Transaction (

    TransactionID INT PRIMARY KEY,

    OrderID INT,

    PaymentDateTime TIMESTAMP,

    Amount DECIMAL(10, 2),

    PaymentMethod VARCHAR(50),

    PaymentStatus VARCHAR(50),

    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

CREATE TABLE DeliveryDriver (

    DeliveryID INT PRIMARY KEY,

    Name VARCHAR(255),

    Address VARCHAR(255),

    EmailID VARCHAR(255) UNIQUE,

    PhoneNo VARCHAR(20),

    VehicleInfo VARCHAR(255)

);

CREATE TABLE DeliveryAssignment (

    AssignmentID INT PRIMARY KEY,

    OrderID INT,

    DeliveryID INT,

    DeliveryAddress VARCHAR(255),

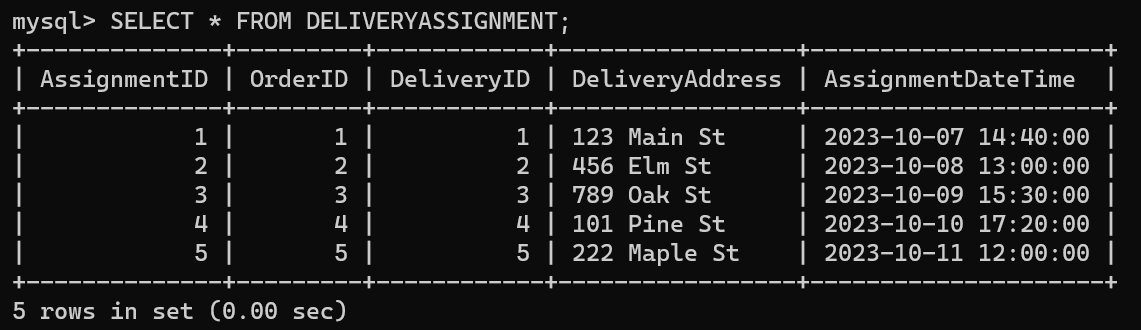
    AssignmentDateTime TIMESTAMP,

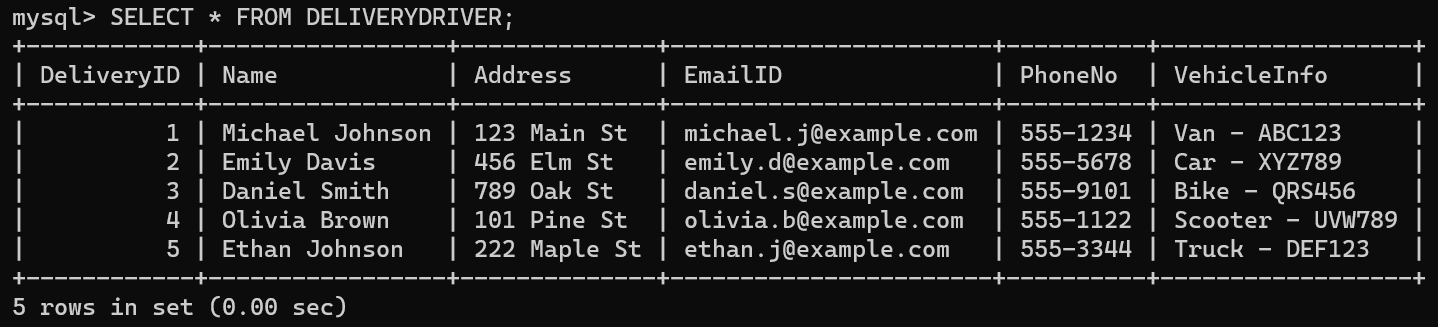
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

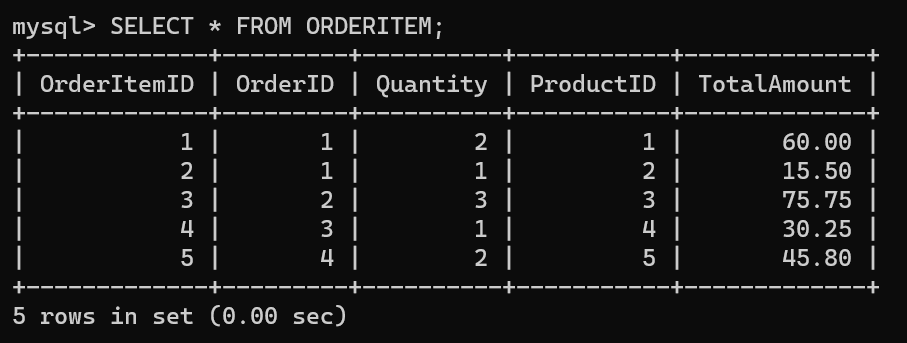
    FOREIGN KEY (DeliveryID) REFERENCES DeliveryDriver(DeliveryID)

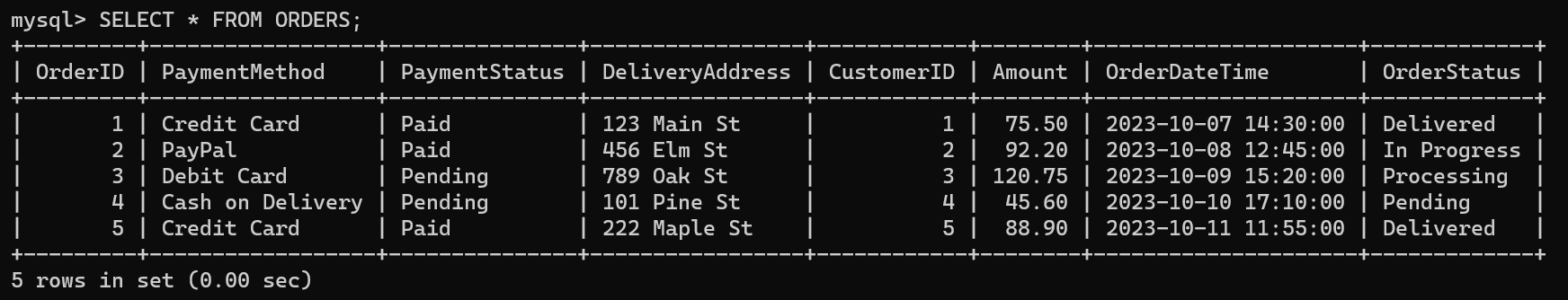
);

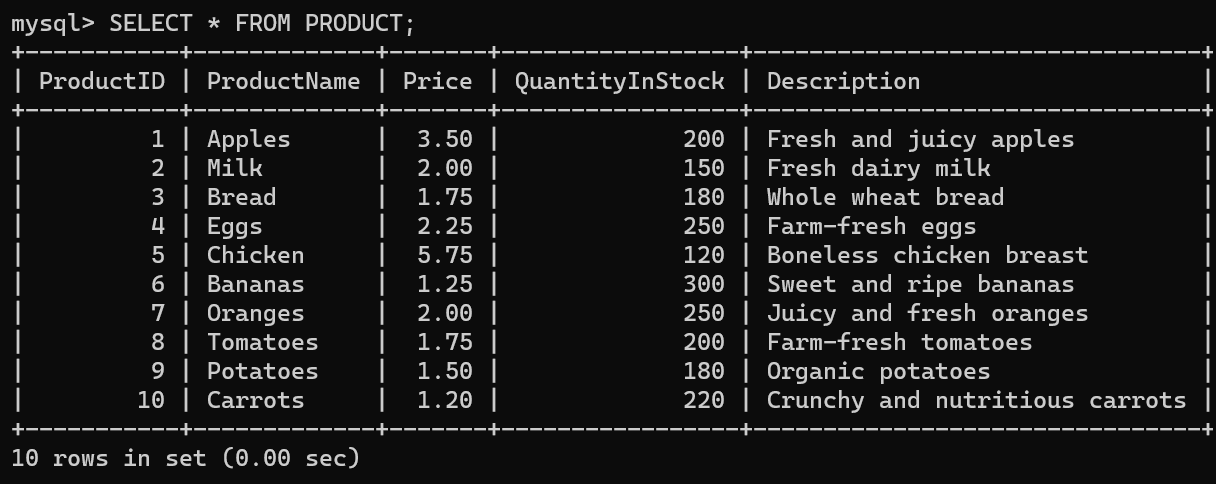


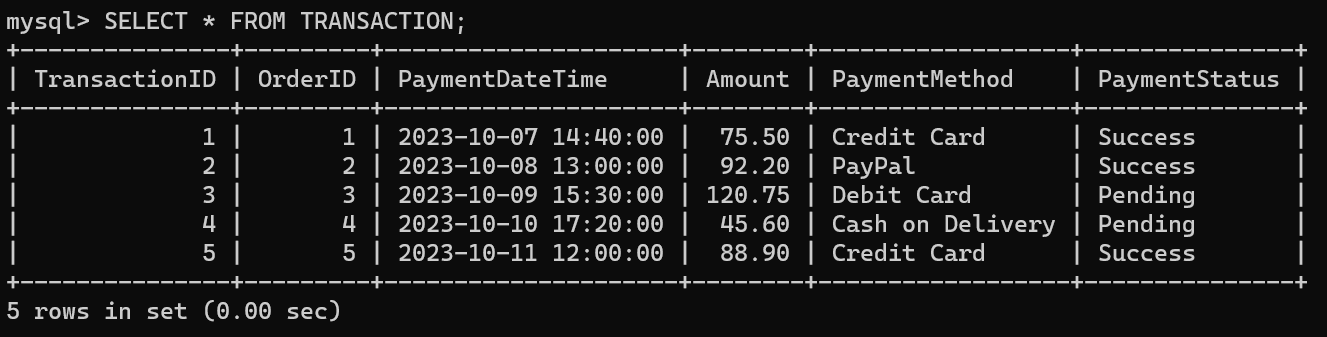












# 6. DML STATEMENTS

**STATEMENTS WITH SCREEN SHOTS OF THE TABLE WITH INSERTED VALUES**

INSERT INTO Customer (CustomerID, FirstName, LastName, EmailID, Username, Password, PhoneNo, Address)

VALUES

(1, 'John', 'Doe', 'john.doe@example.com', 'johndoe123', 'password123', '1234567890', '123 Main St'),

(2, 'Alice', 'Johnson', 'alice.j@example.com', 'alice123', 'securepass', '9876543210', '456 Elm St'),

(3, 'Michael', 'Smith', 'michael.smith@example.com', 'mike123', 'pass123', '2345678901', '789 Oak St'),

(4, 'Emily', 'Davis', 'emily.d@example.com', 'emily456', 'secure123', '3456789012', '101 Pine St'),

(5, 'David', 'Brown', 'david.b@example.com', 'david789', 'davidpass', '4567890123', '222 Maple St'),

(6, 'Emma', 'Johnson', 'emma.j@example.com', 'emma101', 'emma123', '5678901234', '333 Pine St'),

(7, 'Sophia', 'Lee', 'sophia.l@example.com', 'sophia123', 'sophiapass', '6789012345', '444 Elm St'),

(8, 'Noah', 'Williams', 'noah.w@example.com', 'noah789', 'noahpass', '7890123456', '555 Oak St'),

(9, 'Olivia', 'Martin', 'olivia.m@example.com', 'olivia123', 'oliviapass', '8901234567', '666 Maple St'),

(10, 'Liam', 'Thomas', 'liam.t@example.com', 'liam789', 'liampass', '9012345678', '777 Pine St');

INSERT INTO Admins (AdminID, FirstName, LastName, EmailID, Username, Password, PhoneNo, Address)

VALUES

(5, 'Chandana', 'Chandana', 'chandana@example.com', 'Chandana', 'Chandana', '555-555-5559', '202 Banglore'),

(6, 'Bhodhitha', 'Bhodhitha', 'Bhodhitha@example.com', 'Bhodhitha', 'Bhodhitha', '555-555-5560', '303 Banglore'),

(7, 'Rajesh', 'Kumar', 'rajesh.kumar@example.com', 'rajeshkumar', 'rajesh789', '555-555-5561', '404 Lily Road'),

(8, 'Priyanka', 'Singh', 'priyanka.singh@example.com', 'priyankasingh', 'priyanka123', '555-555-5562', '505 Orchid Lane');

INSERT INTO Orders (OrderID, PaymentMethod, PaymentStatus, DeliveryAddress, CustomerID, Amount, OrderDateTime, OrderStatus)

VALUES

(1, 'Credit Card', 'Paid', '123 Main St', 1, 75.50, '2023-10-07 14:30:00', 'Delivered'),

(2, 'PayPal', 'Paid', '456 Elm St', 2, 92.20, '2023-10-08 12:45:00', 'In Progress'),

(3, 'Debit Card', 'Pending', '789 Oak St', 3, 120.75, '2023-10-09 15:20:00', 'Processing'),

(4, 'Cash on Delivery', 'Pending', '101 Pine St', 4, 45.60, '2023-10-10 17:10:00', 'Pending'),

(5, 'Credit Card', 'Paid', '222 Maple St', 5, 88.90, '2023-10-11 11:55:00', 'Delivered');

INSERT INTO Product (ProductID, ProductName, Price, QuantityInStock, Description)

VALUES

(1, 'Apples', 3.50, 200, 'Fresh and juicy apples'),

(2, 'Milk', 2.00, 150, 'Fresh dairy milk'),

(3, 'Bread', 1.75, 180, 'Whole wheat bread'),

(4, 'Eggs', 2.25, 250, 'Farm-fresh eggs'),

(5, 'Chicken', 5.75, 120, 'Boneless chicken breast'),

(6, 'Bananas', 1.25, 300, 'Sweet and ripe bananas'),

(7, 'Oranges', 2.00, 250, 'Juicy and fresh oranges'),

(8, 'Tomatoes', 1.75, 200, 'Farm-fresh tomatoes'),

(9, 'Potatoes', 1.50, 180, 'Organic potatoes'),

(10, 'Carrots', 1.20, 220, 'Crunchy and nutritious carrots');

INSERT INTO OrderItem (OrderItemID, OrderID, Quantity, ProductID, TotalAmount)

VALUES

(1, 1, 2, 1, 60.00),

(2, 1, 1, 2, 15.50),

(3, 2, 3, 3, 75.75),

(4, 3, 1, 4, 30.25),

(5, 4, 2, 5, 45.80);

INSERT INTO Transaction (TransactionID, OrderID, PaymentDateTime, Amount, PaymentMethod, PaymentStatus)

VALUES

(1, 1, '2023-10-07 14:40:00', 75.50, 'Credit Card', 'Success'),

(2, 2, '2023-10-08 13:00:00', 92.20, 'PayPal', 'Success'),

(3, 3, '2023-10-09 15:30:00', 120.75, 'Debit Card', 'Pending'),

(4, 4, '2023-10-10 17:20:00', 45.60, 'Cash on Delivery', 'Pending'),

(5, 5, '2023-10-11 12:00:00', 88.90, 'Credit Card', 'Success');

INSERT INTO DeliveryDriver (DeliveryID, Name, Address, EmailID, PhoneNo, VehicleInfo)

VALUES

(1, 'Michael Johnson', '123 Main St', 'michael.j@example.com', '555-1234', 'Van - ABC123'),

(2, 'Emily Davis', '456 Elm St', 'emily.d@example.com', '555-5678', 'Car - XYZ789'),

(3, 'Daniel Smith', '789 Oak St', 'daniel.s@example.com', '555-9101', 'Bike - QRS456'),

(4, 'Olivia Brown', '101 Pine St', 'olivia.b@example.com', '555-1122', 'Scooter - UVW789'),

(5, 'Ethan Johnson', '222 Maple St', 'ethan.j@example.com', '555-3344', 'Truck - DEF123');

INSERT INTO DeliveryAssignment (AssignmentID, OrderID, DeliveryID, DeliveryAddress, AssignmentDateTime)

VALUES

(1, 1, 1, '123 Main St', '2023-10-07 14:40:00'),

(2, 2, 2, '456 Elm St', '2023-10-08 13:00:00'),

(3, 3, 3, '789 Oak St', '2023-10-09 15:30:00'),

(4, 4, 4, '101 Pine St', '2023-10-10 17:20:00'),

(5, 5, 5, '222 Maple St', '2023-10-11 12:00:00');

**main.py file:**

from flask import Flask,redirect,render\_template,request,flash, session, url\_for

from flask\_sqlalchemy import SQLAlchemy

#database connection

local\_server = True

app = Flask(\_\_name\_\_)

app.secret\_key = "dbmsproject"

#app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://username:password@localhost/databasename'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:Dongre101@localhost/onlinegrocery'

db = SQLAlchemy(app)

class customer(db.Model):

    CustomerID = db.Column(db.Integer, primary\_key=True)

    FirstName = db.Column(db.String(255), nullable=False)

    LastName = db.Column(db.String(255), nullable=False)

    EmailID = db.Column(db.String(255), nullable=False)

    Username = db.Column(db.String(255), nullable=False)

    Password = db.Column(db.String(255), nullable=False)

    PhoneNo = db.Column(db.String(255), nullable=False)

    Address = db.Column(db.String(255), nullable=False)

class deliverydriver(db.Model):

    DeliveryID = db.Column(db.Integer, primary\_key=True)

    Name = db.Column(db.String(255), nullable=False)

    EmailID = db.Column(db.String(255), nullable=False)

    VehicleInfo = db.Column(db.String(255), nullable=False)

    Password = db.Column(db.String(255), nullable=False)

    PhoneNo = db.Column(db.String(255), nullable=False)

    Address = db.Column(db.String(255), nullable=False)

class Product(db.Model):

    ProductID = db.Column(db.Integer, primary\_key=True)

    ProductName = db.Column(db.String(255), nullable=False)

    Price = db.Column(db.Numeric(10, 2), nullable=True)  # Assuming Price can be null

    QuantityInStock = db.Column(db.Integer, nullable=True)  # Assuming QuantityInStock can be null

    Description = db.Column(db.Text, nullable=True)

class order(db.Model):

    OrderID = db.Column(db.Integer, primary\_key=True)

    PaymentMethod = db.Column(db.String(50), nullable=False)

    PaymentStatus = db.Column(db.String(50), nullable=False)

    DeliveryAddress = db.Column(db.String(255), nullable=False)

    CustomerID = db.Column(db.Integer, db.ForeignKey('customer.CustomerID'), nullable=False)  # Assuming there's a 'customer' table

    Amount = db.Column(db.Numeric(10, 2), nullable=False)

    OrderDateTime = db.Column(db.TIMESTAMP, nullable=False, server\_default=db.func.current\_timestamp(), onupdate=db.func.current\_timestamp())

    OrderStatus = db.Column(db.String(50), nullable=False)

class orderitem(db.Model):

    OrderItemID = db.Column(db.Integer, primary\_key=True)

    OrderID = db.Column(db.Integer,db.ForeignKey('order.OrderID'))

    Quantity = db.Column(db.Integer)

    ProductID = db.Column(db.Integer,db.ForeignKey('Product.ProductID'))

    TotalAmount = db.Column(db.Float)

class admins(db.Model):

    AdminID = db.Column(db.Integer, primary\_key=True)

    FirstName = db.Column(db.String(255), nullable=False)

    LastName = db.Column(db.String(255), nullable=False)

    EmailID = db.Column(db.String(255), nullable=False)

    Username = db.Column(db.String(255), nullable=False)

    Password = db.Column(db.String(255), nullable=False)

    PhoneNo = db.Column(db.String(255), nullable=False)

    Address = db.Column(db.String(255), nullable=False)

class deliveryassignment(db.Model):

    AssignmentID = db.Column(db.Integer, primary\_key=True)

    OrderID = db.Column(db.Integer,db.ForeignKey('Order.OrderID'),nullable=False)

    DeliveryID = db.Column(db.Integer,db.ForeignKey('delivery.DeliveryID') ,nullable=False)

    DeliveryAddress = db.Column(db.String(255), nullable=False)

    AssignmentDateTime = db.Column(db.DateTime, nullable=False)

class transaction(db.Model):

    TransactionID = db.Column(db.Integer, primary\_key=True)

    OrderID = db.Column(db.Integer, db.ForeignKey('order.OrderID'), nullable=False)  # Assuming there's an 'order' table

    PaymentDateTime = db.Column(db.TIMESTAMP, nullable=False, server\_default=db.func.current\_timestamp(), onupdate=db.func.current\_timestamp())

    Amount = db.Column(db.Numeric(10, 2), nullable=True)  # Assuming Amount can be null

    PaymentMethod = db.Column(db.String(50), nullable=True)  # Assuming PaymentMethod can be null

    PaymentStatus = db.Column(db.String(50), nullable=True)

#User Login and Authentication Starts

@app.route("/")

def home():

    return render\_template("index.html")

@app.route("/login")

def login():

    return render\_template("login.html")

@app.route("/admin")

def admin\_view():

    return render\_template("Admin.html")

@app.route("/customer")

def customer\_view():

    return render\_template("User.html")

@app.route("/delivery\_person")

def delivery\_person\_view():

    return render\_template("Delivery.html")

@app.route("/login\_process", methods=["POST"])

def login\_process():

    user\_type = request.form["user\_type"]

    if user\_type == "admin":

        return redirect("/admin")

    elif user\_type == "customer":

        return redirect("/customer")

    elif user\_type == "delivery":

        return redirect("/delivery\_person")

    else:

        return "Invalid user type"

@app.route("/signup")

def signup():

    return render\_template("signup.html")

@app.route("/delivery\_login", methods=["POST"])

def delivery\_login():

    username = request.form["username"]

    password = request.form["password"]

    # Check if the username and password match a customer record

    Driver = deliverydriver.query.filter\_by(Name=username, Password=password).first()

    if Driver:

        return redirect("/driver\_home")

    else:

        flash("Invalid username or password. Please try again.", "error")

        return redirect(request.referrer)  # Redirect back to the previous page

@app.route("/admin\_login", methods=["POST"])

def admin\_login():

    username = request.form["username"]

    password = request.form["password"]

    # Check if the username and password match a customer record

    Admin = admins.query.filter\_by(Username=username, Password=password).first()

    if Admin:

        return redirect("/admin\_home")

    else:

        flash("Invalid username or password. Please try again.", "error")

        return redirect(request.referrer)  # Redirect back to the previous page

@app.route("/signup\_process", methods=["POST"])

def signup\_process():

    role = request.form["role"]

    firstname = request.form["firstname"]

    lastname = request.form["lastname"]

    email = request.form["email"]

    username = request.form["username"]

    password = request.form["password"]

    phoneno = request.form["phoneno"]

    address = request.form["address"]

    try:

        if role == "customer":

            ID = customer.query.count()

            new\_customer = customer(

                CustomerID = ID + 1,

                FirstName=firstname,

                LastName=lastname,

                EmailID=email,

                Username=username,

                Password=password,

                PhoneNo=phoneno,

                Address=address

            )

            db.session.add(new\_customer)

            db.session.commit()

        elif role == "delivery":

            ID = deliverydriver.query.count()

            new\_delivery\_person = deliverydriver(

                DeliveryID = ID +1,

                Name=firstname + " " + lastname,

                EmailID=email,

                VehicleInfo="",  # Add the vehicle information if available

                Password=password,

                PhoneNo=phoneno,

                Address=address

            )

            db.session.add(new\_delivery\_person)

        elif role == "admin":

            # Add code to handle admin registration if needed

            ID = admins.query.count()

            new\_admin = admins(

                AdminID = ID +1,

                FirstName=firstname,

                LastName=lastname,

                EmailID=email,

                Username=username,

                Password=password,

                PhoneNo=phoneno,

                Address=address

            )

            db.session.add(new\_admin)

        else:

            flash("Invalid role selected. Please try again.", "error")

            return redirect("/signup")  # Redirect back to the signup page if an invalid role is selected

        db.session.commit()

        flash("User registered successfully!", "success")

        return redirect("/login")  # Redirect to the login page after successful signup

    except Exception as e:

        print(e)

        flash("An error occurred while registering. Please try again.", "error")

        return redirect("/signup")  # Redirect back to the signup page in case of an error

#User login and authentication ends here

#Customer page starts here

@app.route("/customer\_login", methods=["POST"])

def customer\_login():

    username = request.form["username"]

    password = request.form["password"]

    # Check if the username and password match a customer record

    Customer = customer.query.filter\_by(Username=username, Password=password).first()

    if Customer:

        # Store the customer's information in the session

        session['customer\_id'] = Customer.CustomerID

        session['customer\_username'] = Customer.Username

        session['customer\_name'] = f"{Customer.FirstName} {Customer.LastName}"

        # Print the session for debugging

        print(session)

        return redirect("/customer\_home")  # Redirect to the customer home page

    else:

        flash("Invalid username or password. Please try again.", "error")

        return redirect(request.referrer)  # Redirect back to the previous page

from main import app, Product

@app.route("/customer\_home")

def customer\_home():

    # Retrieve the username from the session with a default value of None

    username = session.get("customer\_username")

    # Fetch products from the Product table

    products = Product.query.all()

    # Pass the username and products to the template for rendering

    return render\_template("customer\_home.html", username=username, products=products)

@app.route("/add\_to\_cart/<int:product\_id>", methods=["POST"])

def add\_to\_cart(product\_id):

    selected\_product = Product.query.get(product\_id)

    if selected\_product:

        # Initialize the cart in the session if not already present

        if 'cart' not in session:

            session['cart'] = []

        # Check if the product is already in the cart

        for item in session['cart']:

            if item['id'] == selected\_product.ProductID:

                flash('Product already in cart', 'info')

                return redirect(url\_for("customer\_home"))

        # Add the selected product to the cart

        session['cart'].append({

            'id': selected\_product.ProductID,

            'name': selected\_product.ProductName,

            'price': float(selected\_product.Price),

            'quantity': 1  # Default quantity is set to 1

        })

        flash('Item added to cart successfully', 'success')

        return redirect(url\_for("customer\_home"))

    else:

        # Handle the case where the product is not found

        return render\_template('product\_not\_found.html')

@app.route("/view\_cart")

def view\_cart():

    # Retrieve the cart from the session

    cart = session.get('cart', [])

    for item in cart:

        item['price'] = float(item['price'])

    # Calculate the total price of items in the cart

    total\_price = sum(item['price'] for item in cart)

    # Pass the cart and total\_price to the template for rendering

    return render\_template("view\_cart.html", cart=cart, total\_price=total\_price)

@app.route('/update\_quantity/<int:product\_id>', methods=['POST'])

def update\_quantity(product\_id):

    new\_quantity = int(request.form.get(f'quantity\_{product\_id}', 1))

    remove\_item = request.form.get('remove\_item', False)

    cart = session.get('cart', [])

    selected\_product = next((item for item in cart if item['id'] == product\_id), None)

    if selected\_product:

        if new\_quantity > 0:

            selected\_product['quantity'] = new\_quantity

            # flash('Quantity updated successfully')

        # if new\_quantity == 0:

        #     # Remove the item from the session

        #     cart = session.get('cart', [])

        #     updated\_cart = [item for item in cart if item['id'] != product\_id]

        #     session['cart'] = updated\_cart

        #     flash(f'The product {selected\_product["name"]} is removed from the cart', 'info')

        if remove\_item.lower() == 'true':

            updated\_cart = [item for item in cart if item['id'] != product\_id]

            session['cart'] = updated\_cart

            # flash(f'The product {selected\_product["name"]} is removed from the cart', 'info')

    # Recalculate total price

    total\_price = calculate\_total\_price(cart)

    return render\_template('view\_cart.html', cart=cart, total\_price=total\_price)

def calculate\_total\_price(cart):

    return sum(item['price'] \* item['quantity'] for item in cart)

@app.route('/clear\_cart')

def clear\_cart():

    # Clear the cart when the "Clear the Cart" button is clicked

    session.pop('cart', None)

    # flash('Cart cleared successfully')

    return redirect(url\_for('view\_cart'))

@app.route('/checkout')

def checkout():

    # Clear the cart when proceeding to pay

    session.pop('cart', None)

    return render\_template('checkout.html') # Create a checkout.html template if needed

#Customer page ends here

#Inventory Management Starts

@app.route("/admin\_home")

def admin\_home():

    return render\_template("admin\_home.html")

@app.route("/inventory\_management", methods=["GET", "POST"])

def handle\_CRUD():

    selected\_table = request.form["table\_select"]

    crud\_operation = request.form["crud\_operation"]

    # Redirect to the appropriate route based on the selected table and CRUD operation

    return redirect((f"{crud\_operation}\_{selected\_table}"))

# Placeholder routes for each CRUD operation on the 'product' table (adjust as needed)

@app.route("/view\_<table\_name>")

def view\_product(table\_name):

    # Define a mapping between table names and corresponding models

    table\_model\_mapping = {

        "product": Product,

        "admins" : admins,

        "customer" : customer,

        "deliverydriver" : deliverydriver,

        "order" : order,

        "orderitem" : orderitem,

        "deliveryassignment" : deliveryassignment,

        "transaction" : transaction

    }

    # Check if the requested table\_name is valid

    if table\_name in table\_model\_mapping:

        # Fetch all records from the specified table

        records = table\_model\_mapping[table\_name].query.all()

        # Render the template with the fetched data

        return render\_template("view\_table.html", table\_name=table\_name, records=records)

@app.route("/add\_<table\_name>", methods=["GET", "POST"])

def add\_record(table\_name):

    # Define a mapping between table names and corresponding models

    table\_model\_mapping = {

        "product": Product,

        "admins": admins,

        "deliverydriver": deliverydriver,

        "deliveryassignment": deliveryassignment,

    }

    if table\_name in table\_model\_mapping:

        # Get the model for the specified table\_name

        model = table\_model\_mapping[table\_name]

        # Get the columns of the model

        columns = [column.name for column in model.\_\_table\_\_.columns]

        if request.method == "POST":

            try:

                # Extract form data from the request

                form\_data = {column: request.form[column] for column in columns}

                print(form\_data)

                # Create a new record and add it to the database

                new\_record = model(\*\*form\_data)

                db.session.add(new\_record)

                db.session.commit()

                return "Record added successfully!"

            except Exception as e:

                # Handle exceptions (e.g., validation errors, database errors)

                print(e)

                db.session.rollback()  # Rollback changes in case of an error

                return f"Error: {str(e)}"

        else:

            return render\_template("add\_record.html", table\_name=table\_name, columns=columns)

    else:

        return "Table not found", 404

@app.route("/edit\_product", methods=["GET","POST"])

def edit\_product():

    # Define a mapping between table names and corresponding models

    table\_model\_mapping = {

        "product": Product

    }

    if request.method == "GET":

        # Display a form to select the product ID for editing

        return render\_template("select\_product.html", table\_name="product")

    elif request.method == "POST":

        # Handle the form submission to get the selected product ID

        product\_id = request.form.get("product\_id")

        if product\_id:

            # Redirect to the edit page for the selected product

            return redirect(f"/edit/{product\_id}")

        else:

            # If no product ID is provided, display an error or redirect to the selection page

            return "Please select a product for editing."

@app.route("/edit/<int:product\_id>", methods=["GET", "POST"])

def edit\_product\_details(product\_id):

    # Define a mapping between table names and corresponding models

    # Get the product by ID

    product = Product.query.get\_or\_404(product\_id)

    if request.method == "POST":

        # Handle form submission to update product details

        try:

            # Update product fields based on the form data

            product.ProductName = request.form.get("product\_name", product.ProductName)

            product.Price = float(request.form.get("price", product.Price))

            product.QuantityInStock = int(request.form.get("quantity\_in\_stock", product.QuantityInStock))

            product.Description = request.form.get("description", product.Description)

            # Commit the changes to the database

            db.session.commit()

            return "Updated Product Successfully"

        except Exception as e:

            # Handle validation errors or other exceptions

            db.session.rollback()

            return f"Error: {str(e)}"

    # Render the edit form with the current product details

    return render\_template("edit\_product\_details.html", product=product)

@app.route("/delete\_product", methods=["GET", "POST"])

def delete\_product():

    # Add logic for deleting a product

    # Define a mapping between table names and corresponding models

    table\_model\_mapping = {

        "product": Product

    }

    if request.method == "GET":

        # Display the page to select the product for deletion

        return render\_template("select\_product\_for\_deletion.html")

    elif request.method == "POST":

        # Handle the form submission to get the selected product ID for deletion

        product\_id = request.form.get("product\_id")

        if product\_id:

            # Redirect to the delete page for the selected product

            return redirect(f"/delete/{product\_id}")

        else:

            # If no product ID is provided, display an error or redirect back to the selection page

            return "Please select a product for deletion."

@app.route("/delete/<int:product\_id>", methods=["GET", "POST"])

def delete\_product\_confirm(product\_id):

    if request.method == "GET":

        # Display the confirmation page

        return render\_template("delete\_product\_details.html", product\_id=product\_id)

    elif request.method == "POST":

        # Handle the form submission to delete the product with the given ID

        table\_model\_mapping = {

            "product": Product

        }

        product\_to\_delete = Product.query.get(product\_id)

        if product\_to\_delete:

            try:

                # Delete the product from the database

                db.session.delete(product\_to\_delete)

                db.session.commit()

                flash("Product deleted successfully!", "success")

                return "Product deleted successfully"

            except Exception as e:

                print(e)

                db.session.rollback()

                flash("An error occurred while deleting the product. Please try again.", "error")

        else:

            flash("Product not found.", "error")

            return "Product not found"

    # Return a response for all cases

    return redirect(url\_for("inventory\_management"))

#Inventory Management Ends

@app.route("/logout", methods=["POST"])

def logout():

    # Clear the session

    session.clear()

    # Redirect to the login page

    return redirect("/login")

@app.route('/account')

def account():

    user\_id=session.get("customer\_id")

    # Fetch a sample user from the database

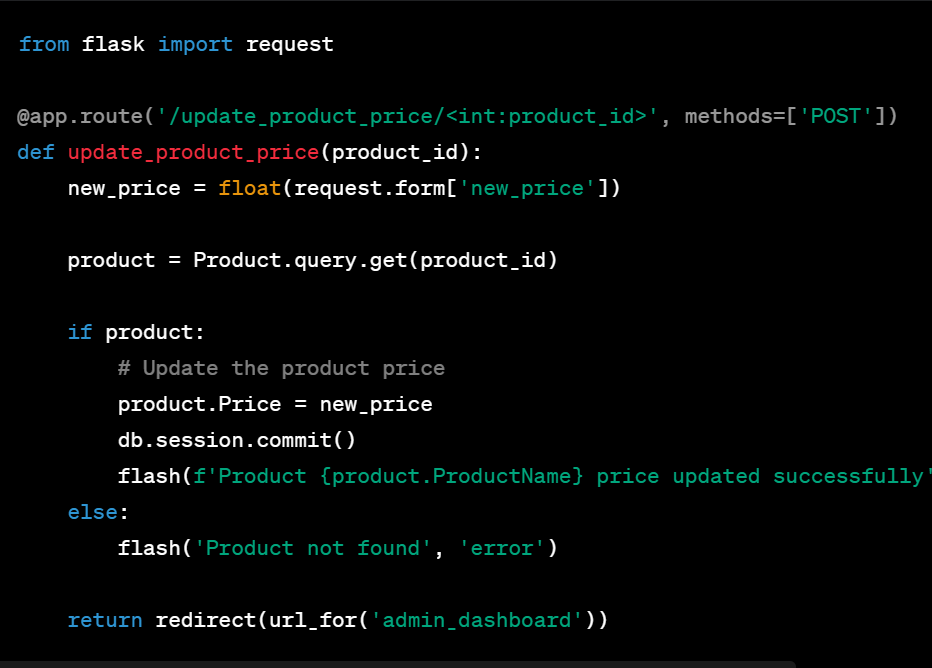
    sample\_customer = customer.query.filter\_by(CustomerID=user\_id).first()

    # Pass the user details to the template

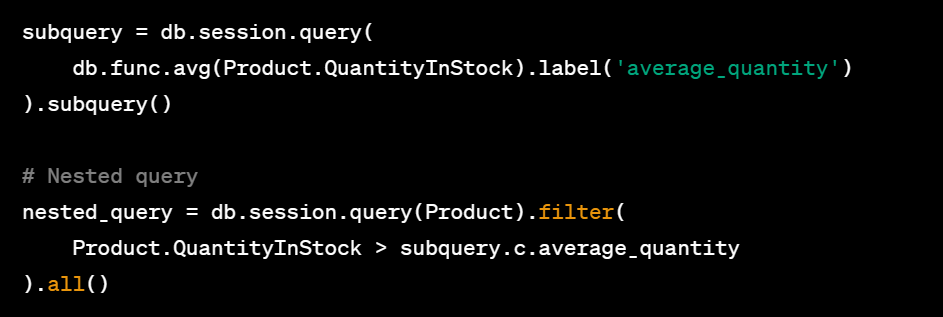
    return render\_template('account.html', customer=sample\_customer)

app.run(debug=True)

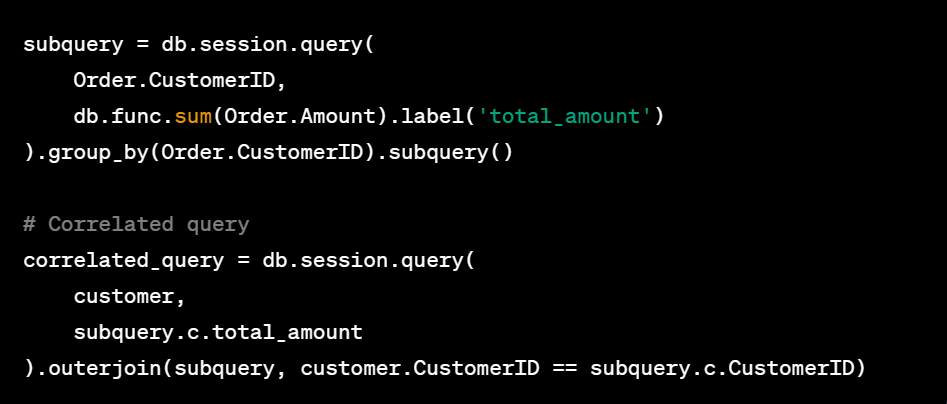
# 7. QUERIES

**7.1 UPDATE OPERATION**

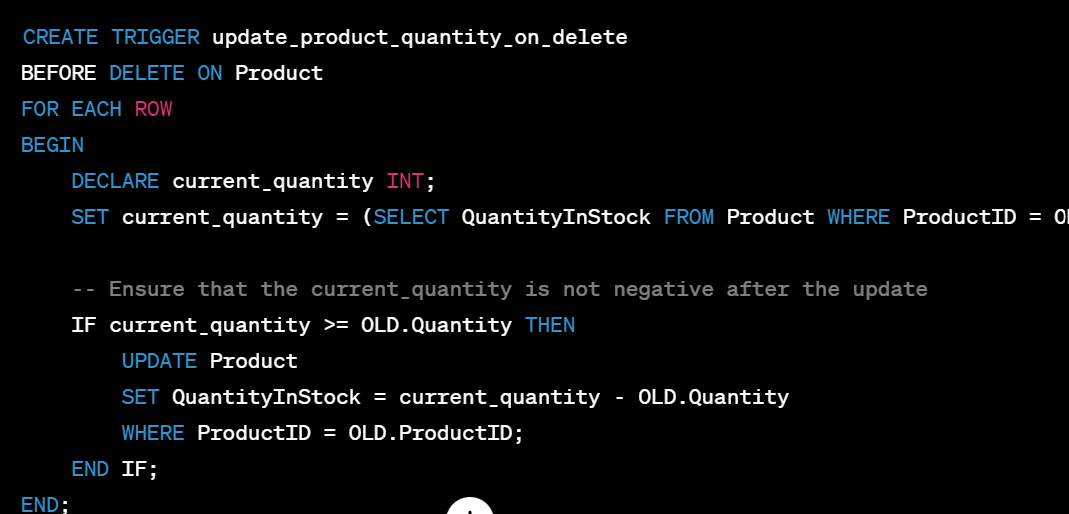
**7.2 NESTED QUERY**

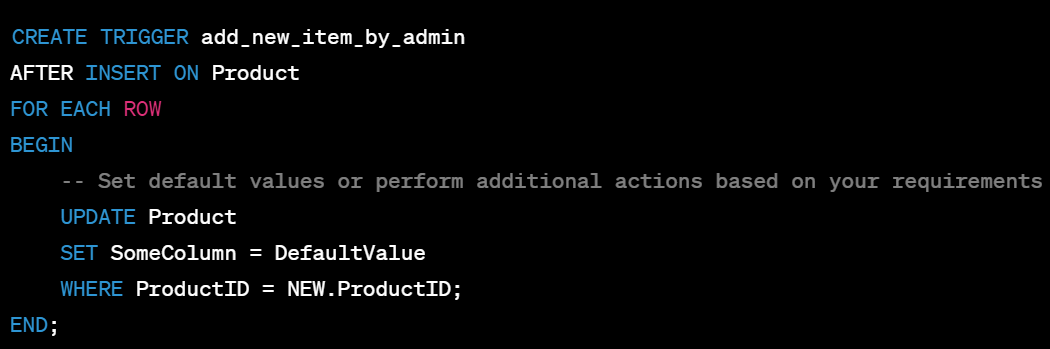
****

**7.3 DELETE OPERATION**

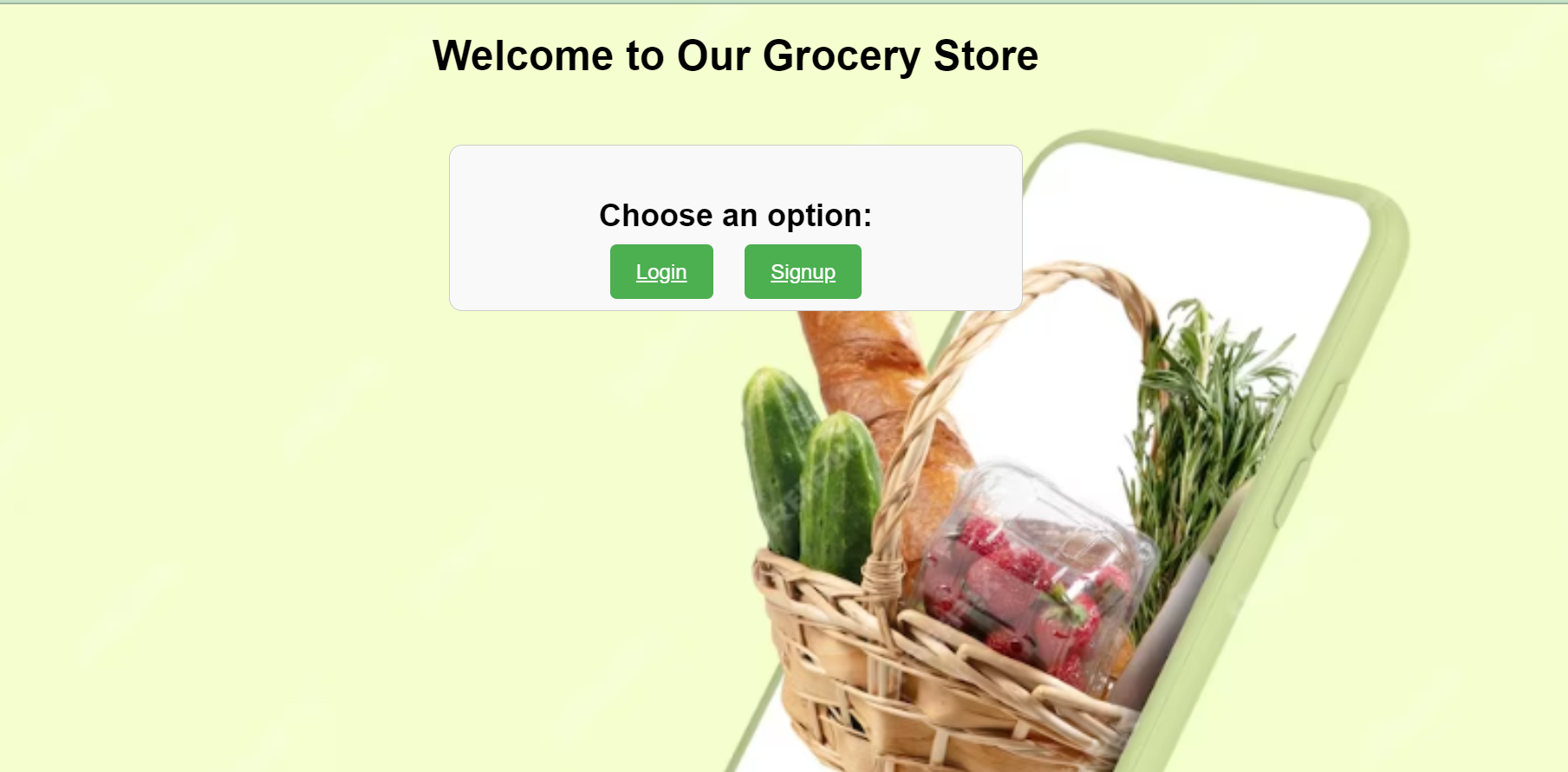
**7.4 CORRELATED QUERY**

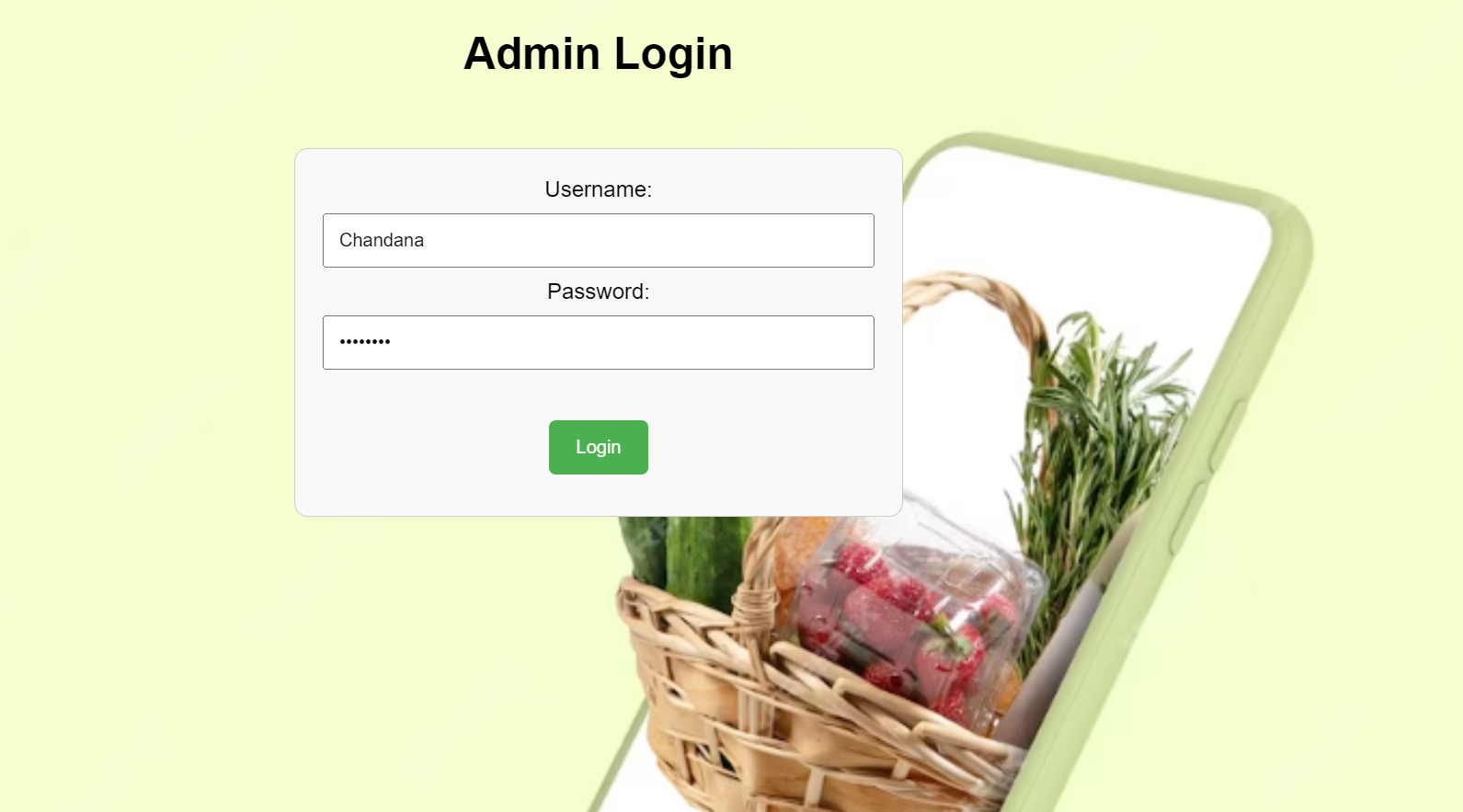
**7.5 TRIGGERS**

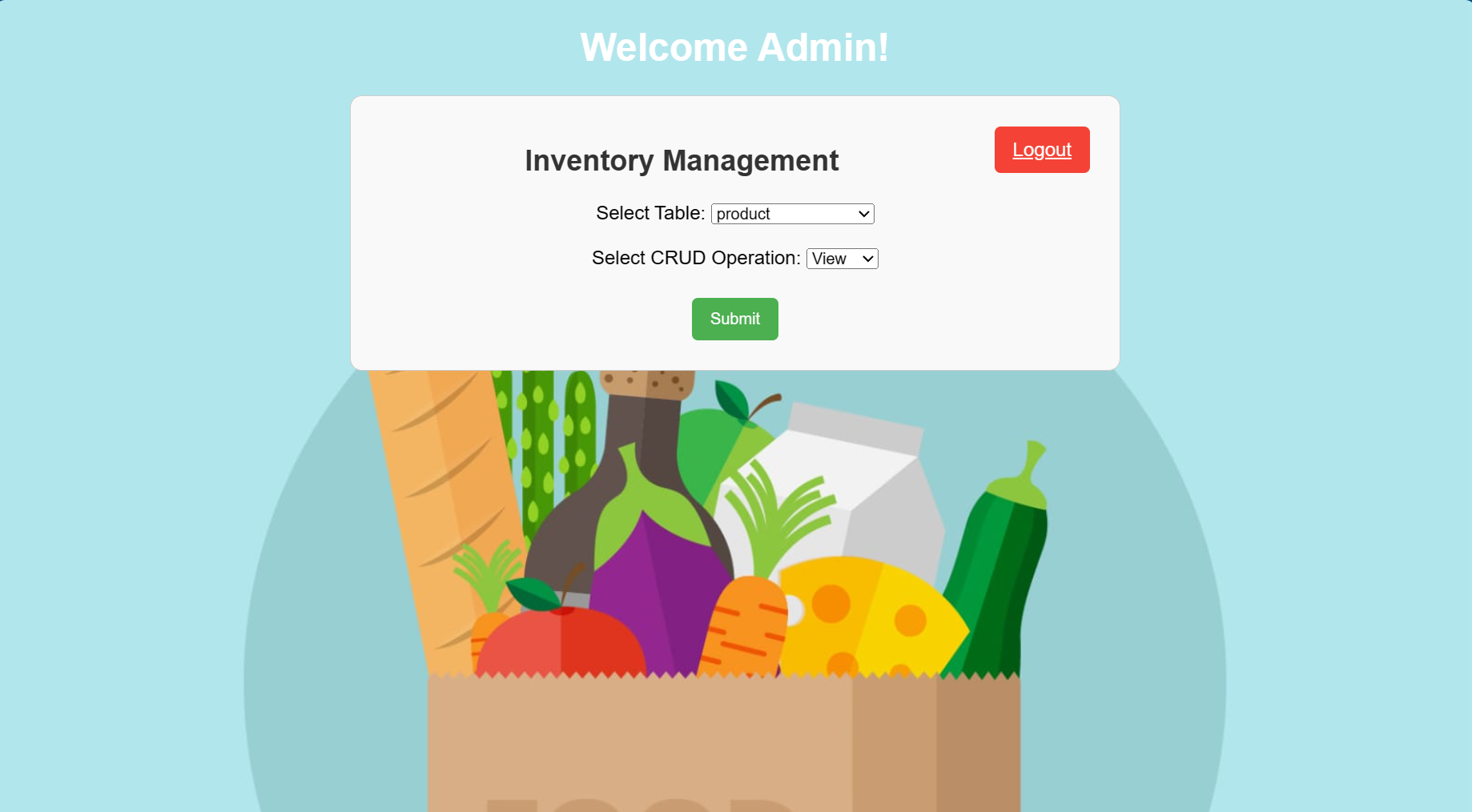
****

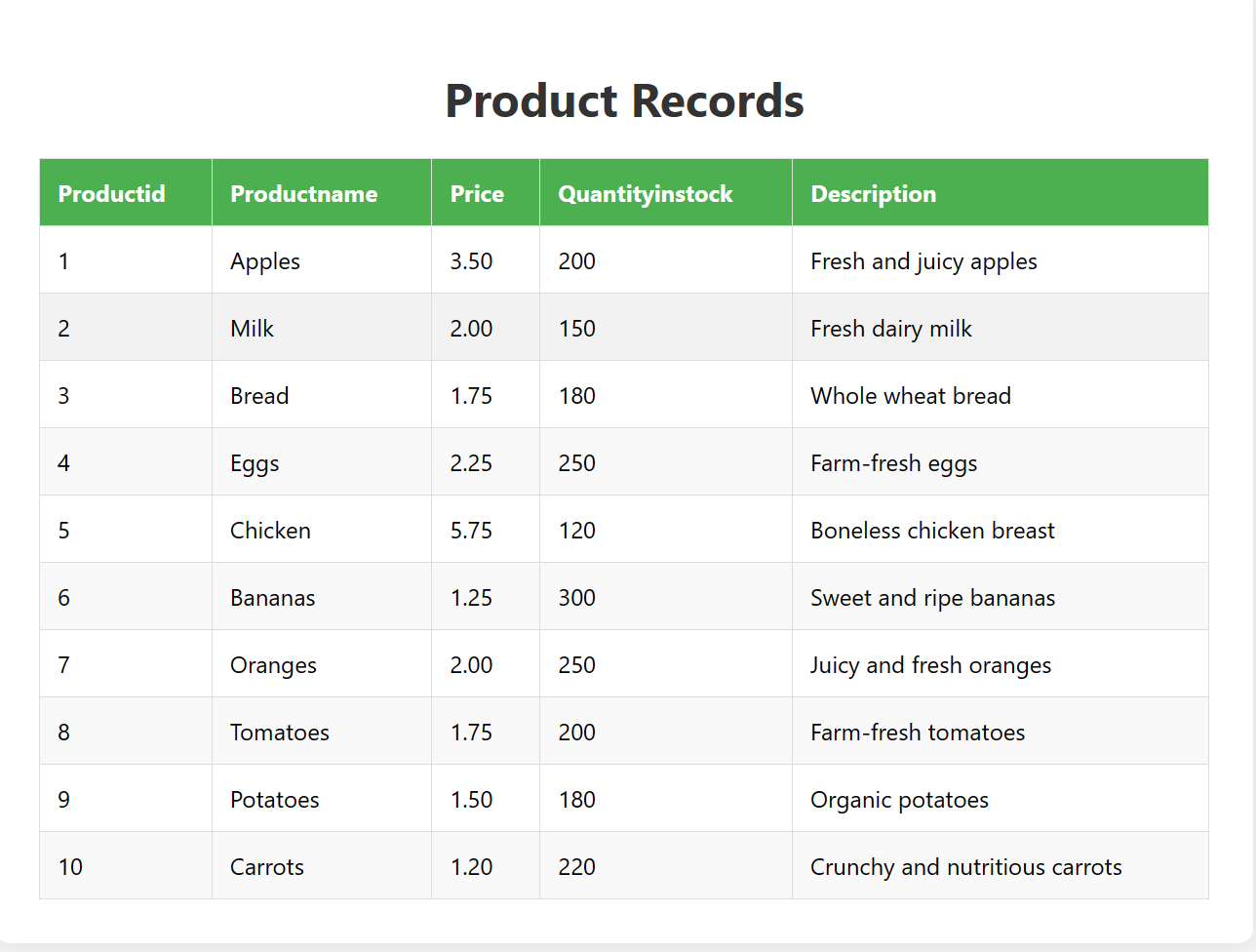
****

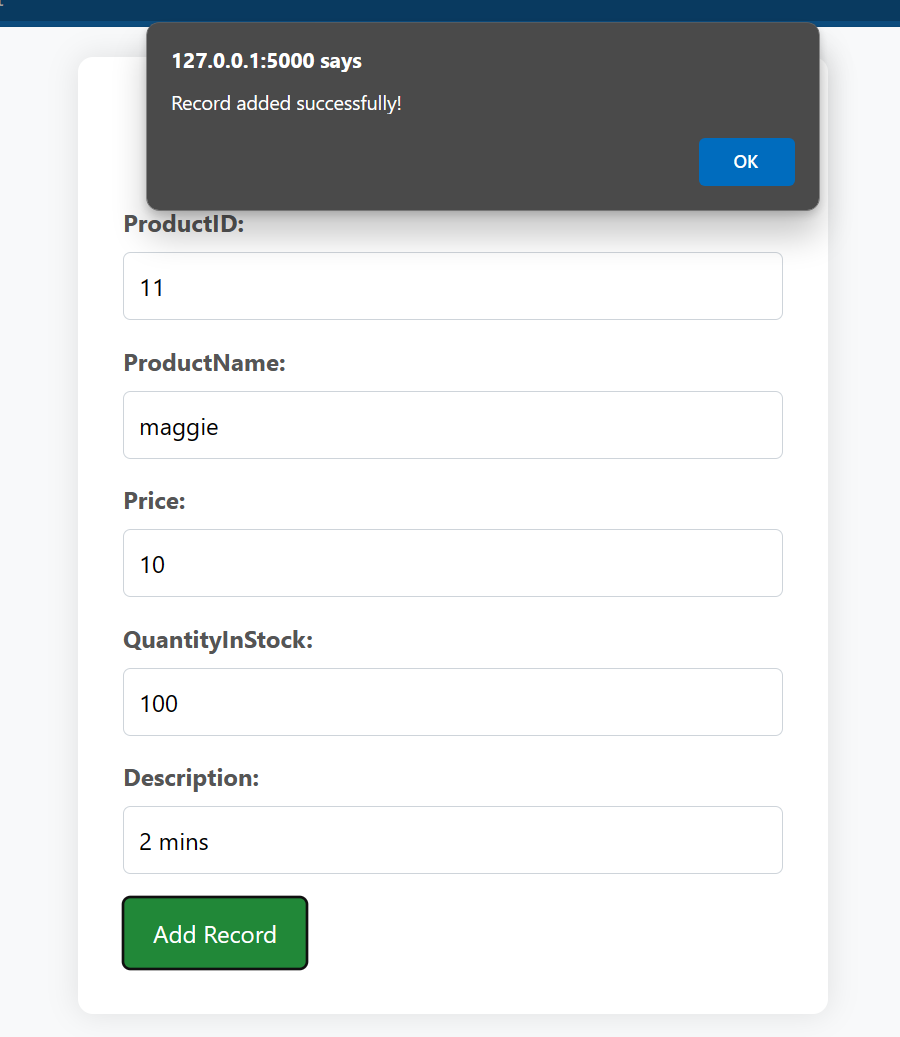
# 8.FRONT END DEVELOPEMNT

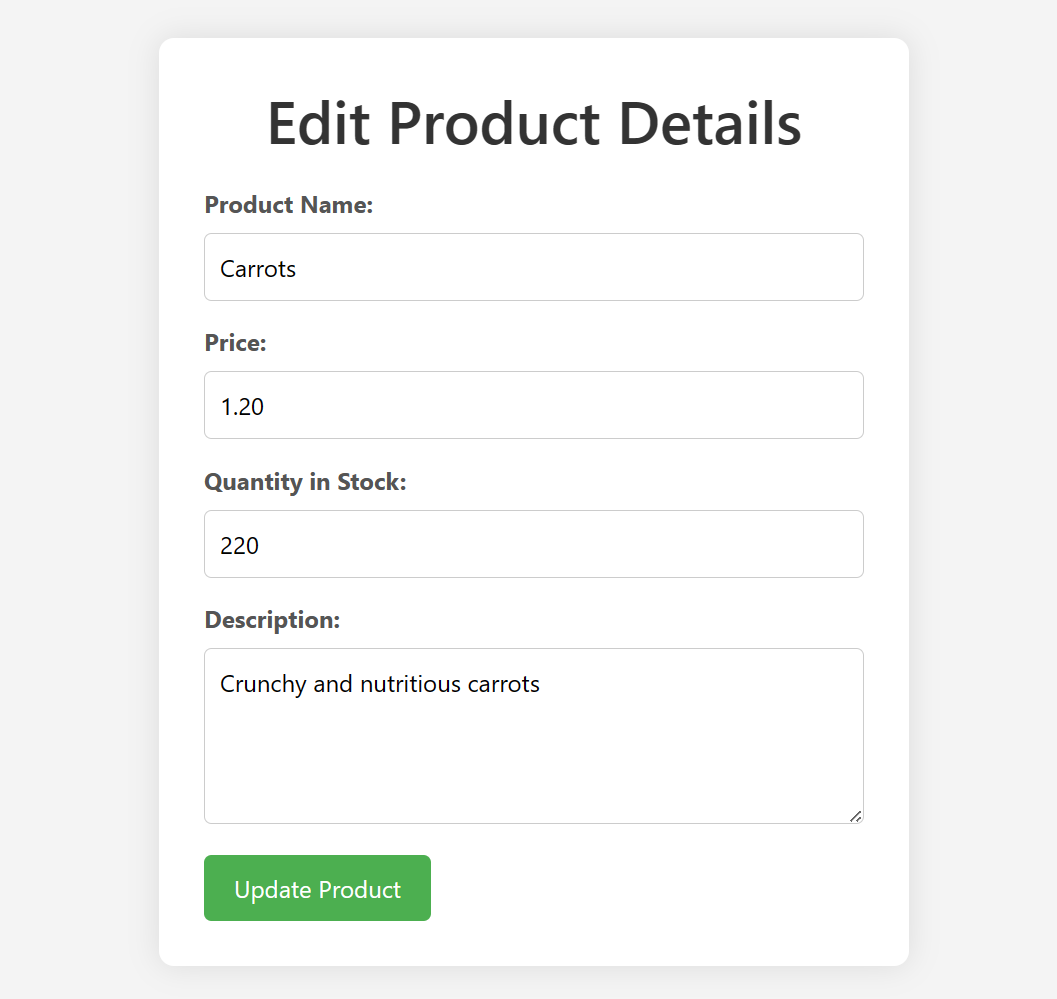


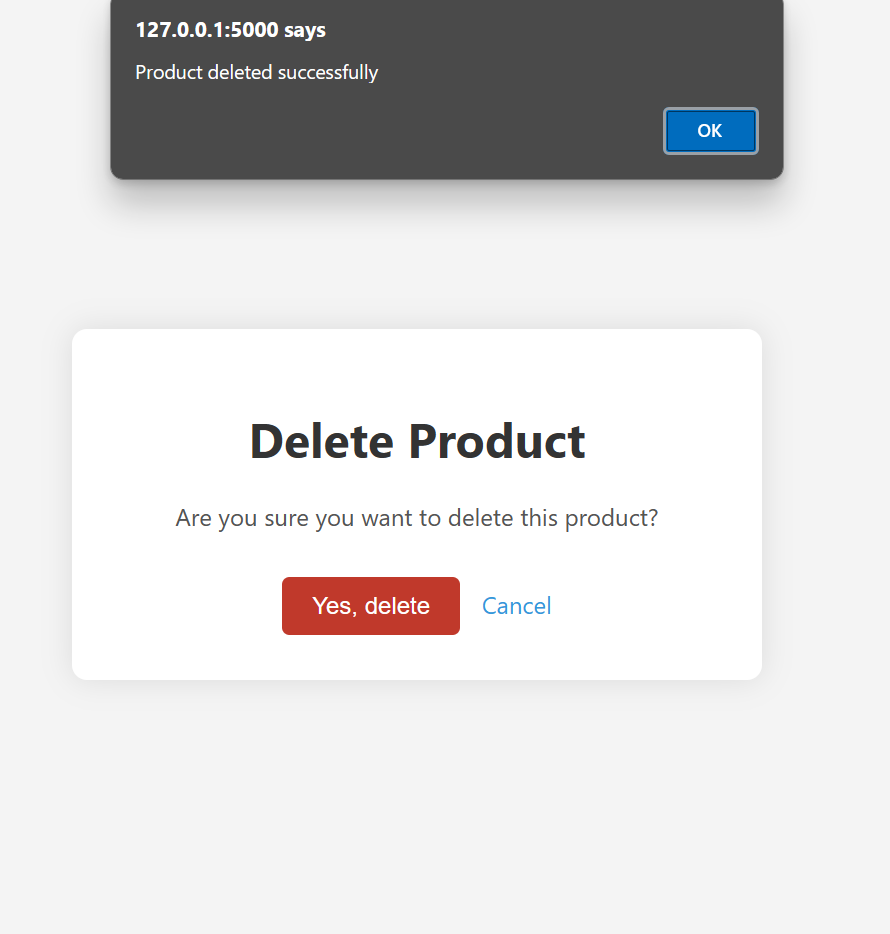


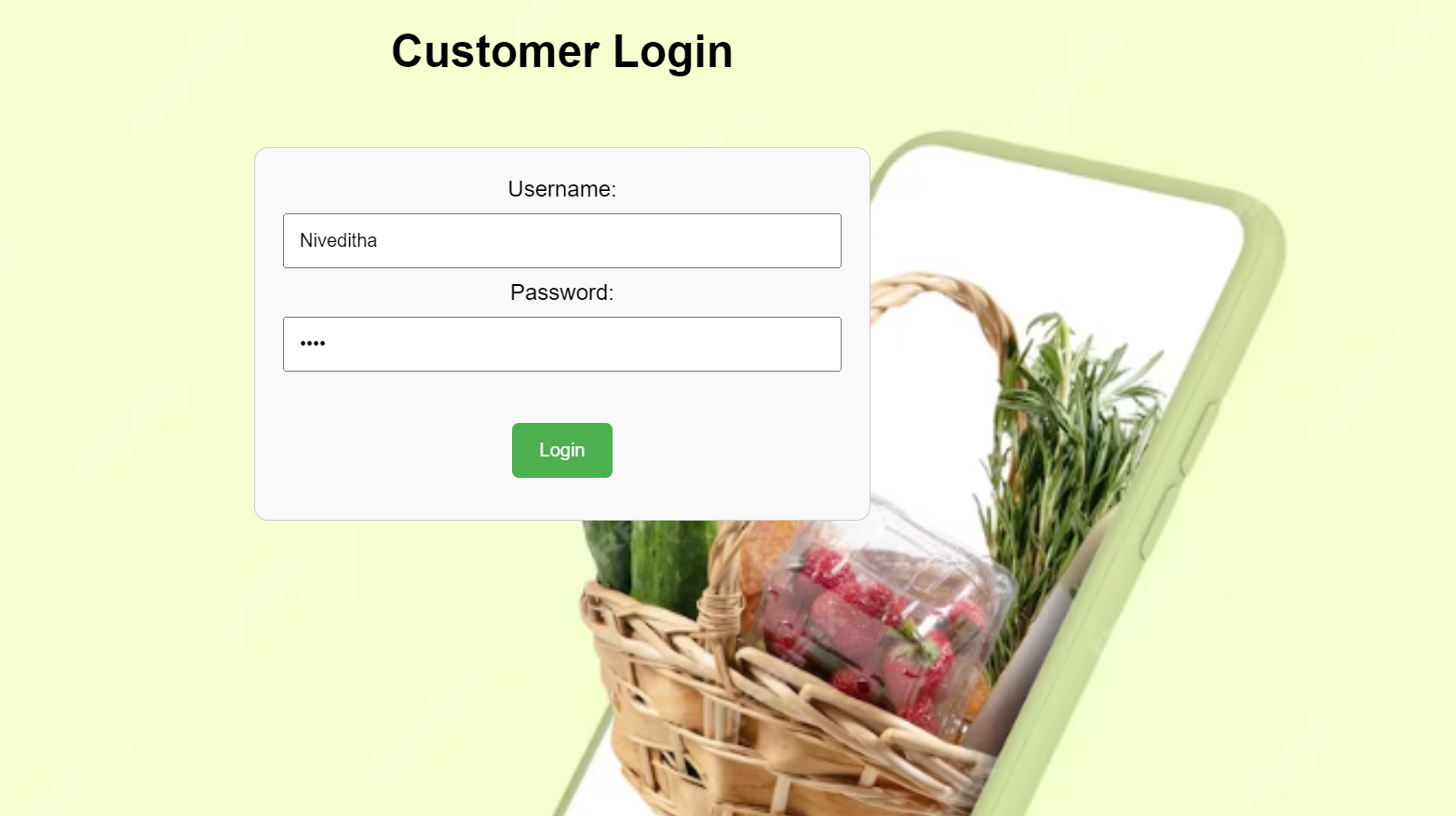


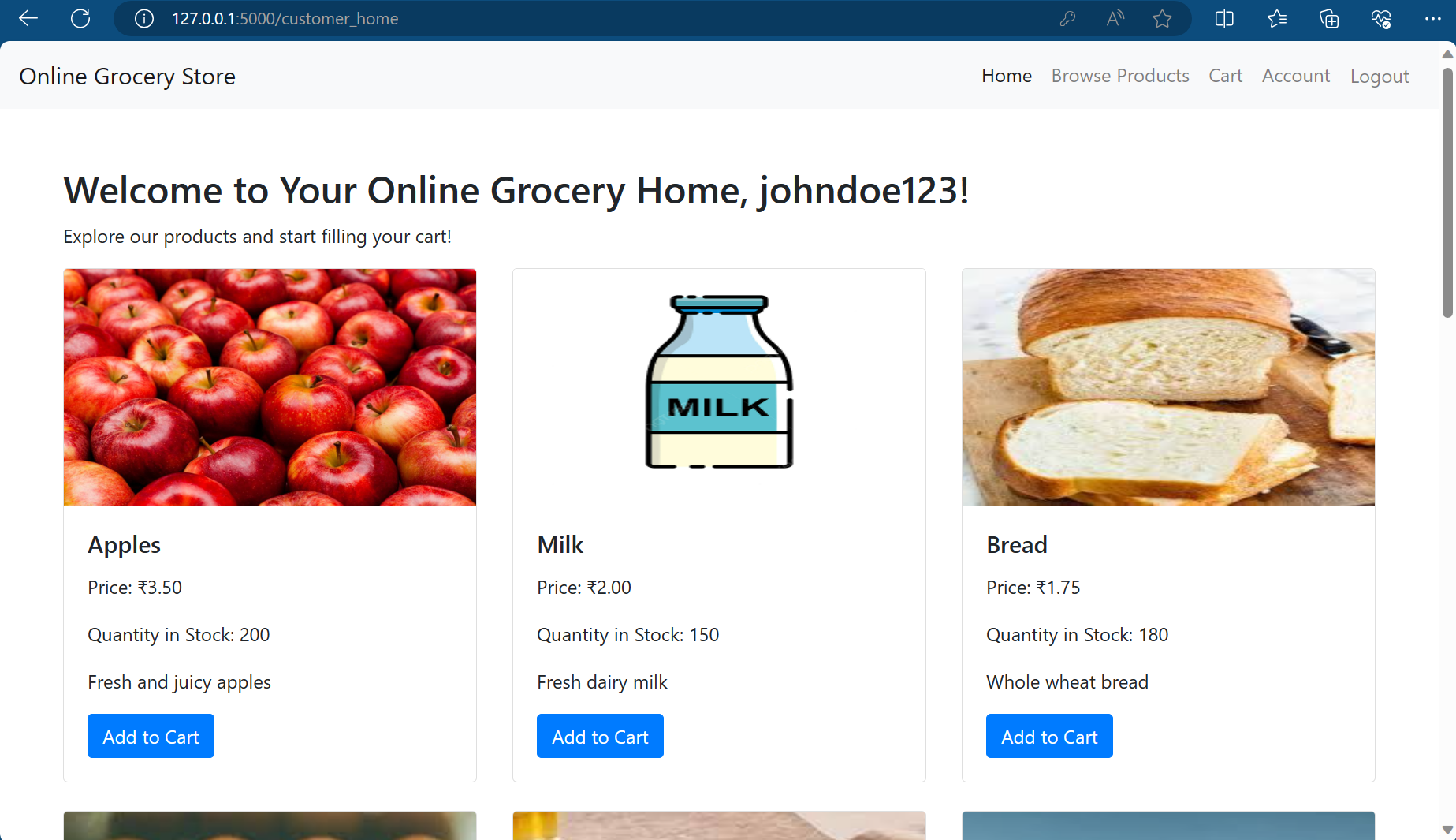


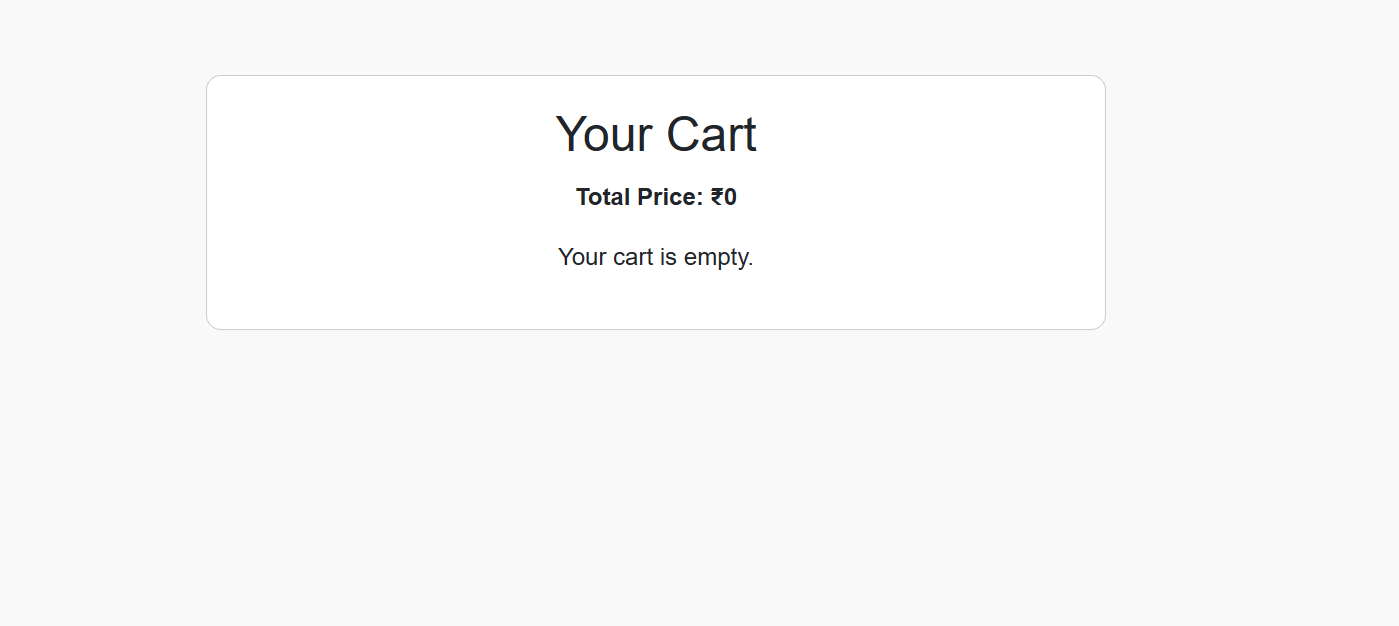


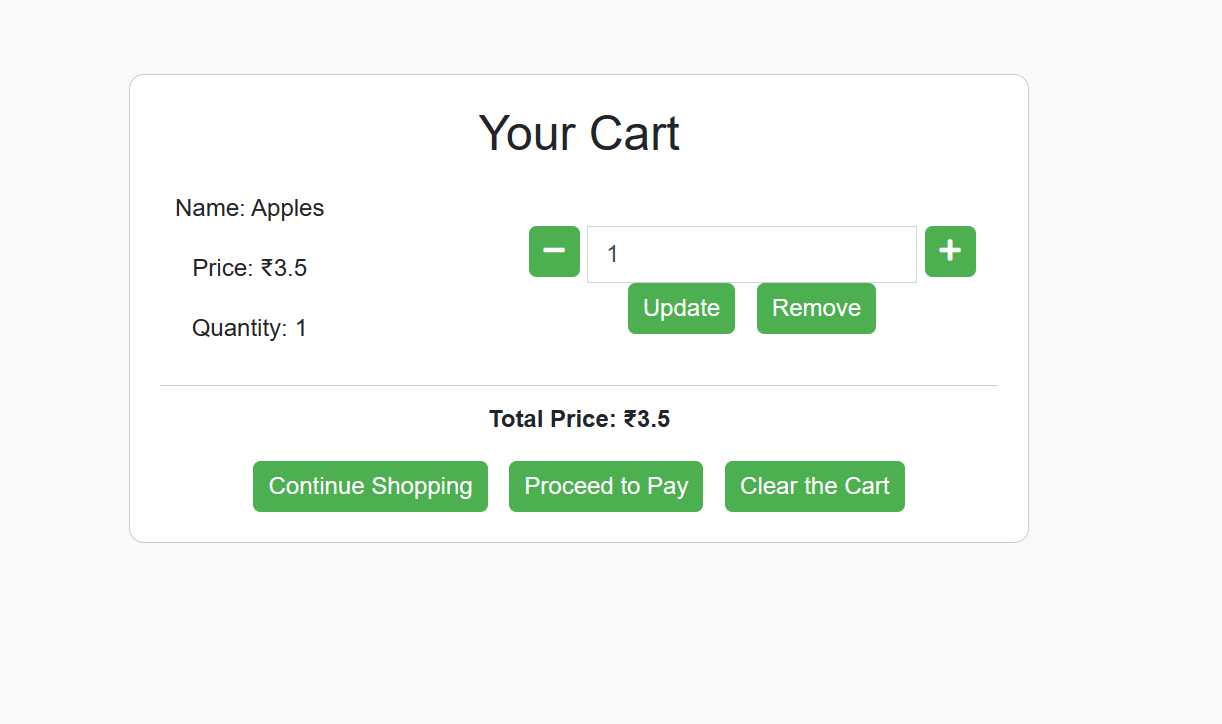


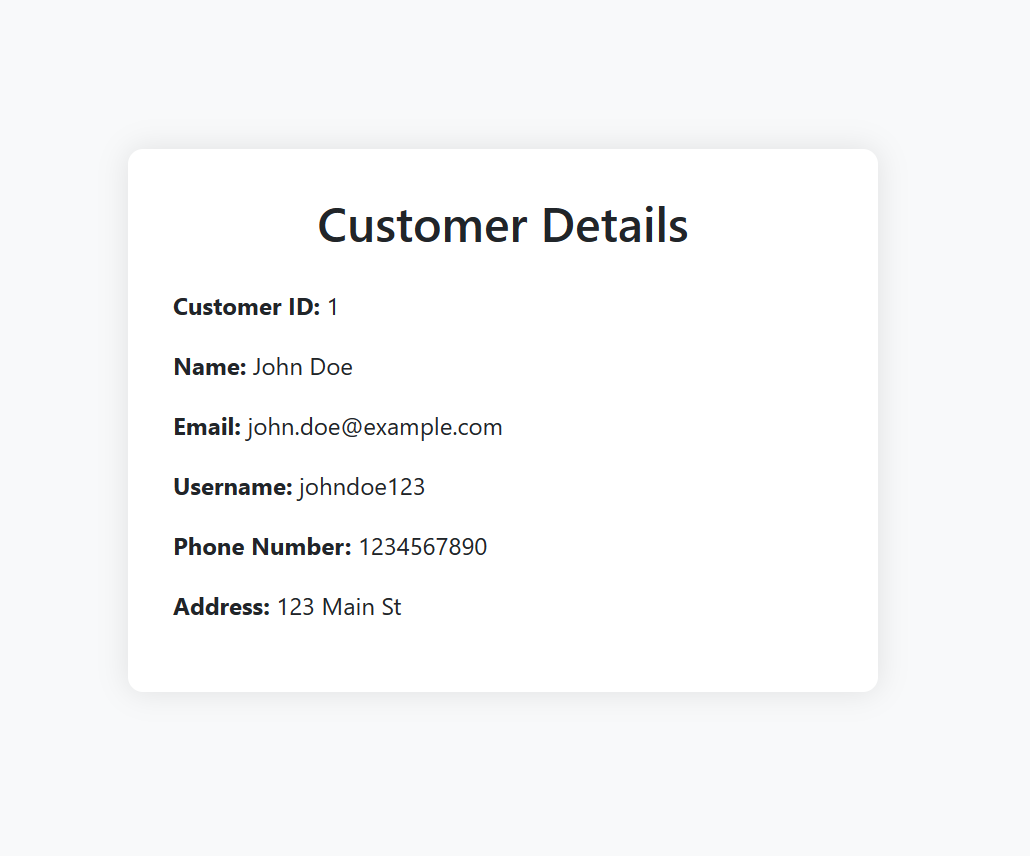








****

****

**Conclusion**

The online grocery management system, developed with Flask, MySQL, and PHP, offers an efficient and user-friendly solution for customers, delivery personnel, and administrators. Customers can seamlessly browse and purchase groceries, while the system ensures secure transactions and order processing. With robust MySQL database integration, the platform efficiently manages product details, user information, and order history. The PHP scripting enhances dynamic content and server-side functionality. The inclusion of Flask, a lightweight web framework, streamlines routing and facilitates a responsive user experience. This system showcases the integration of diverse technologies to create a comprehensive online grocery delivery solution, promoting accessibility, reliability, and effective management across the entire supply chain.

**REFERENCES**

1. **Flask Documentation:**
   * [Flask Official Documentation](https://flask.palletsprojects.com/): The official documentation for Flask, a lightweight web application framework for Python.
2. **PHP Documentation:**
   * [PHP Manual](https://www.php.net/manual/en/): The PHP manual provides comprehensive documentation for PHP, a popular server-side scripting language.
3. **MySQL Documentation:**
   * [MySQL Documentation](https://dev.mysql.com/doc/): The official documentation for MySQL, an open-source relational database management system.
4. **Web Development Tutorials:**
   * [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web): Mozilla Developer Network (MDN) offers extensive web development resources, including tutorials on HTML, CSS, and JavaScript.

**Database Design:**

* Database-System-Concepts-7th-Edition Sudarshan and Korth: This book provides a solid foundation in database concepts and design.
* Fundamentals of Database Systems - 7th Ed- Elmasri and Navathe