Online Grocery Store

PACEINSTITUTEOF TECHNOLOGY & SCIENCES

2nd SHIFT POLYTECHNIC

COMPUTER ENGINEERING

Submitted by

R.Ravi teja	23471-cm-085
P.Abhi ram	23471-cm-077
P.Venkat teja	23471-cm-119
T.Rohith	23471-cm-104
SK.Abdul haseeb	23471-cm-091
G.Rohith kumar	23471-cm-123
T.Mokshagna	23471-cm-098
P.Arya	23471-cm-071
p.Krishna	23471-cm-073

Undertheesteemedguidance of

P.Jahnavi B.Tech

Pace institute of technology &sciences

(ApprovedbyAICTE,NewDelhi,AccreditedbyNBAandNCCAwith Agrade)

(Permanently Affiliated to Jawaharlal Nehru technological university. Kakinada)

(Vallur NH-16, Ongole, prakasam district, AP-523272)

2ndSHIFT POLYTECHNIC

Departmentofcomputerengineering

CERTIFICATE

R.Ravi teja :23471-cm-085

P.Abhi ram :23471-cm-077

P.Venkat teja:23471-cm-119

T.Rohith:23471-cm-104

SK.Abdul haseeb:23471-cm-085

G.Rohith kumar:23471-cm-085

T.Mokshagna :23471-cm-085

P.Arya :23471-cm-085

p.Krishna:23471-cm-085

PROJECT BY

The infinity Bank Of India (IBI) is a software project that automates and organizes bank operations, allowing customers to manage accounts, perform transactions like deposits and withdrawals, and access their information online. The project's core purpose is to improve efficiency by reducing manual work, minimizing errors, and providing a secure, user-friendly platform for both customers and bank staff to manage financial activities

Signature of Guidance.

P.Jahnavi B.Tech

Signature of Head of department

B.Satish Kumar (MTech)

Signature of the External Examiner

Signature of the principal

ACKNOWLEDGMENT

We thank to almighty forgivingus the course andperseverance in counting the main project. The project itself is Acknowledgment for all those people who have give as their heart-felt co-operation in making this project a grand success

We extend our sincere thanks to Dr.M.Venugopalrao. BE,MBA chairman of our college for providing sufficient infrastructure and good environment in the college to complete our course .

We are thankful to our secretary Dr.M.Sridhar mtech,MBA for providing the necessary infrastructure and labs and permitting to carry out this project .

We are thankful to our principal Mr. G. Koti reddy M.tech,PhD for providing the necessary infrastructure and labs and also permitting to carry out this project.

We extreme jubilance and deepest gratitude, we would like to thank our head of the CME department Mr. B.Satish Kumar M.Tech for his constant encouragement.

We are greatly indebted to project guide P.Jahnavi B.tech Lecturer, of computer engineering, for providing valuable guidance at every stage of this project work, we are profoundly grateful towards the unmatched services rendered by him. Our special thanks to all the facility of computer engineering and peers for their valuable advises at every stage of this work

Online Grocery Store

R.Ravi teja

P.Abhi ram

P.Venkat teja

T.Rohith

SK.Abdul

haseeb

G.Rohith kumar

T.Mokshagna

P.Arya

p.Krishna

CONTENTS

1. Online Grocery Store

1.1.

Abstraction :

The Online Grocery Store is a comprehensive web-based e-commerce platform designed to revolutionize the traditional grocery shopping experience.
This digital solution enables customers to browse, select, and purchase groceries from the comfort of their homes while providing administrators with robust tools to manage products, inventory, and customer orders efficiently.
Built using modern web technologies, the system offers a seamless shopping experience with features like real-time cart management, secure user authentication, order tracking, and an intuitive admin dashboard for store management.

1.2. Introduction:

The digital transformation of retail has made online grocery shopping an essential service for modern consumers. This Online Grocery Store project addresses the growing demand for convenient, accessible, and efficient grocery shopping solutions. The platform bridges the gap between traditional brick-and-mortar stores and contemporary e-commerce expectations, offering customers a user-friendly interface to purchase daily essentials while providing store administrators with powerful management capabilities.
The system is designed to cater to both individual consumers and business administrators, creating a symbiotic ecosystem where customers can easily find and order products while administrators can efficiently manage inventory, process orders, and monitor business performance.

1.3. Objective:

Primary Objectives:

- Develop a user-friendly online platform for grocery shopping
- Implement secure user authentication and authorization systems
- Create an efficient product catalog with search and categorization
- Develop a seamless shopping cart and checkout process
- Build comprehensive admin capabilities for store management
- Ensure responsive design for cross-device compatibility

Secondary Objectives:

- Implement real-time inventory management
- Provide order tracking and status updates
- Enable customer order history and reordering
- Develop analytics and reporting features for administrators
- Ensure data security and privacy protection

1.4. Functionalities:

Customer Functionalities:

- User Registration & Authentication: Secure sign-up and login system
- Product Browsing: Categorized product listings with search functionality
- Product Details: Comprehensive product information and images
- Shopping Cart: Add, remove, and modify items in real-time
- Checkout Process: Secure payment and delivery information collection
- Order Management: View order history and track current orders
- Profile Management: Update personal information and delivery addresses

Administrative Functionalities:

- Management: Add, edit, delete, and categorize products
- Inventory Control: Monitor and update stock levels
- Order Processing: View, update, and manage customer orders
- User Management: Monitor customer accounts and activities
- Dashboard Analytics: Business insights and performance metrics
- Image Management: Upload and manage product images

1.5.A dvantagesandDisadvantages:

- **24/7 Accessibility**: Customers can shop anytime, anywhere
- **Time Efficiency**: Eliminates physical store travel and waiting
- **Wide Product Selection**: Comprehensive catalog without space constraints

- **Real-time Inventory:** Accurate stock information prevents disappointment
- **Order History:** Easy reordering of frequently purchased items
- **Business Insights:** Data-driven decision making for administrators
- **Scalability:** Easy to expand product range and customer base

Disadvantages:

- **Technical Dependency:** Requires stable internet connection
- **Delivery Logistics:** Complex last-mile delivery management
- **Product Inspection:** Customers cannot physically examine products
- **Initial Setup:** Requires significant development and testing
- **Security Concerns:** Potential data breaches and payment security issues
- **Customer Adaptation:** Some users may prefer traditional shopping methods

2. Languages Used In grocery store

Frontend Technologies:

- **HTML5:** Structure and semantic markup
- **CSS3:** Styling, animations, and responsive design
- **JavaScript:** Client-side interactivity and dynamic content
- **Bootstrap 5:** CSS framework for responsive layout

Backend Technologies:

- **Python**: Primary server-side programming language
- **Flask**: Web framework for application development
- **Jinja2**: Templating engine for dynamic HTML generation

Data base:

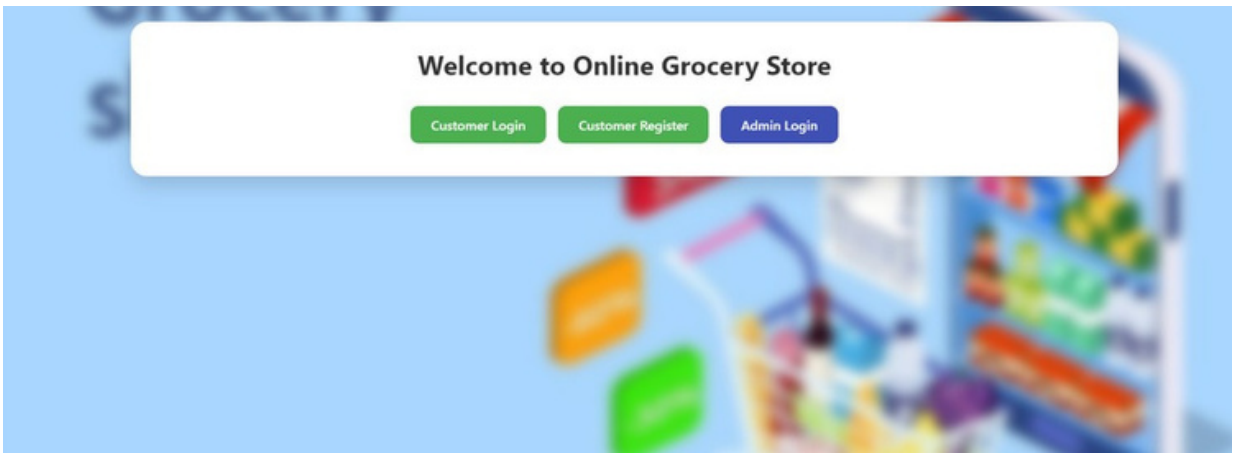
- **SQLite:** Lightweight database for data storage and management

Additional Technologies:

- **SQL:** Database queries and operations
- **RESTful APIs:** For cart updates and dynamic content
- **File Upload Handling:** For product image management
- **Session Management:** For user authentication state

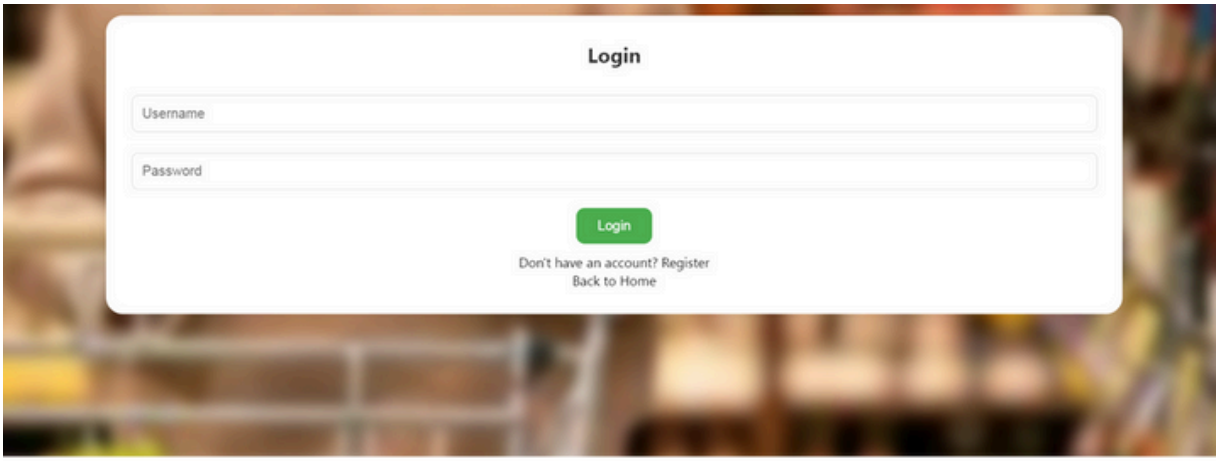
3. Layouts In The Grocery Store:

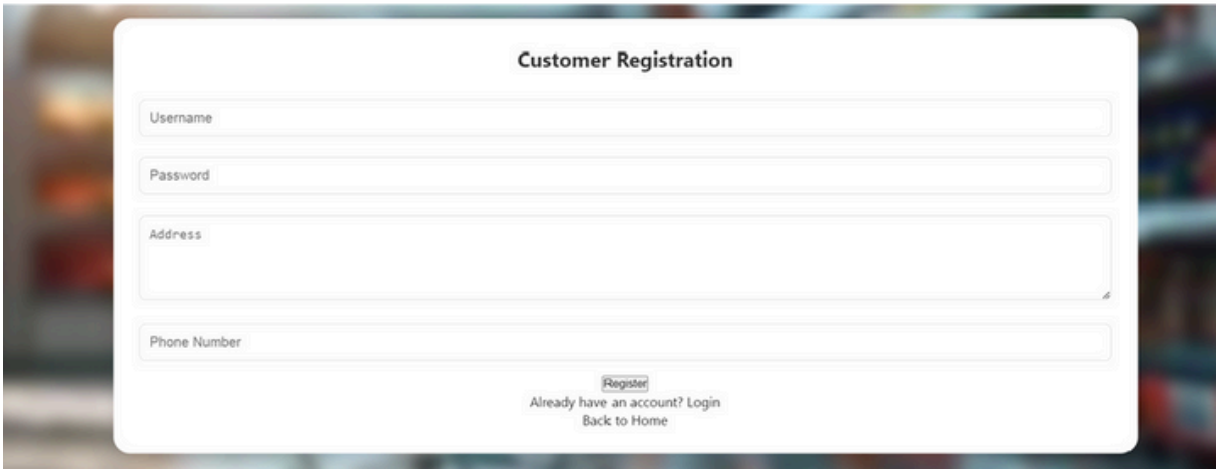
3.1. Home Page (Index.html):



- **Welcome Interface:** Attractive landing page with brand identity
- **Navigation Options:** Clear pathways to login, register, and admin access
- **Visual Appeal:** High-quality background imagery with blurred effects
- **Call-to-Action Buttons:** Prominent customer and admin access points

3.2. login and register page(login.html,register.html):





- **Authentication Forms:** Clean, focused login and registration interfaces
- **Background Design:** Themed background images for visual appeal
- **Form Validation:** Real-time input validation and error messaging
- **Navigation Links:** Easy access to alternative actions (register/login)

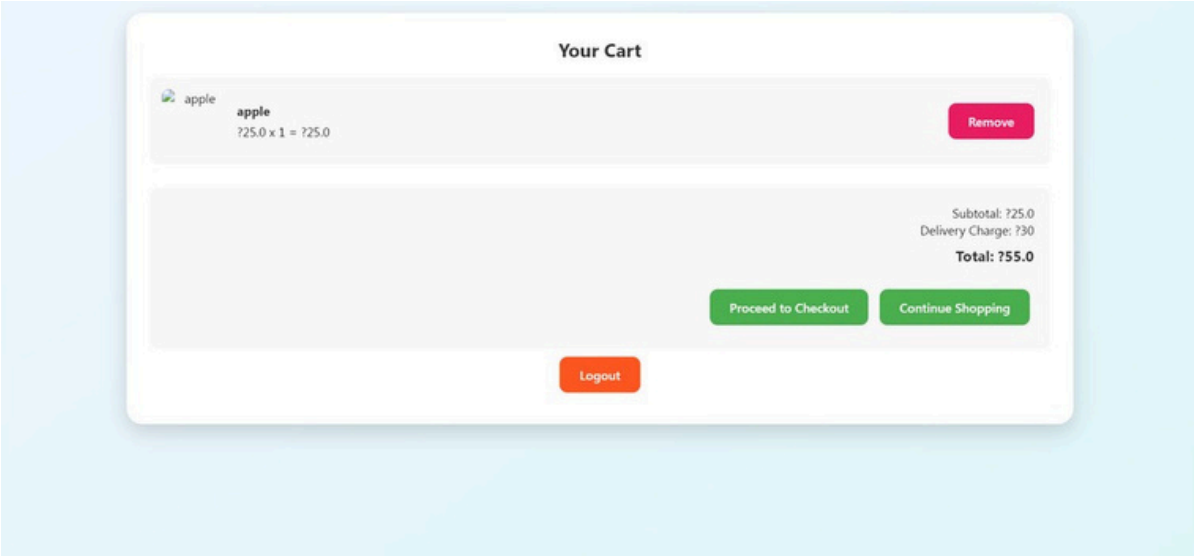
3.6. Admin login page(admin.html):

- **Authentication Forms:** Clean, focused admin login
- **Background Design:** Themed background images for visual appeal
- **Form Validation:** Real-time input validation and error messaging
- **Navigation Links:** Easy access to alternative actions (register/login)

3.3. Shopping Cart Page:

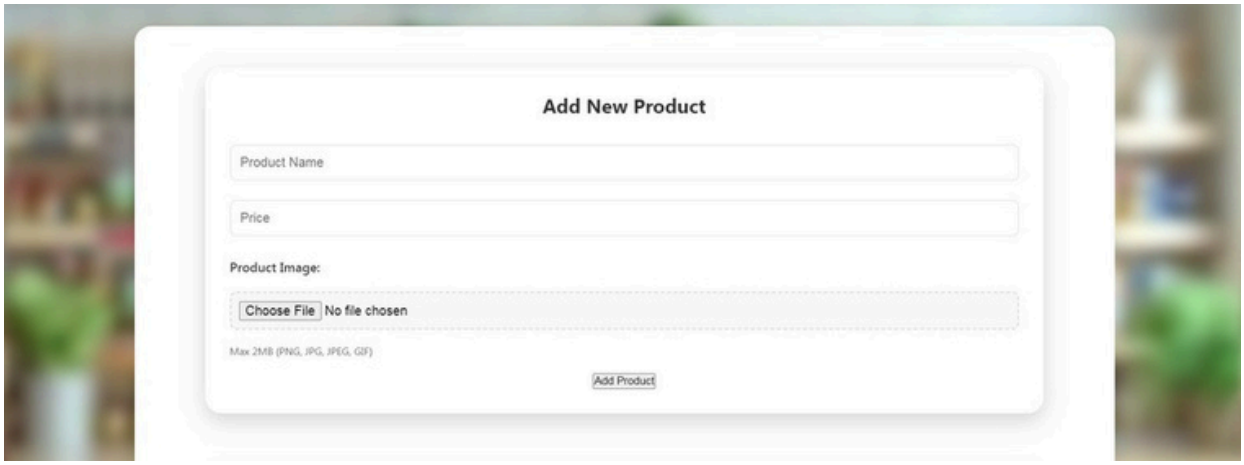
- **Itemized Display:** Clear listing of selected products with quantities
- **Price Summary:** Detailed breakdown of subtotal, delivery charges, and total
- **Management Options:** Easy item removal and quantity adjustments
- **Checkout Progression:** Clear path to complete purchase

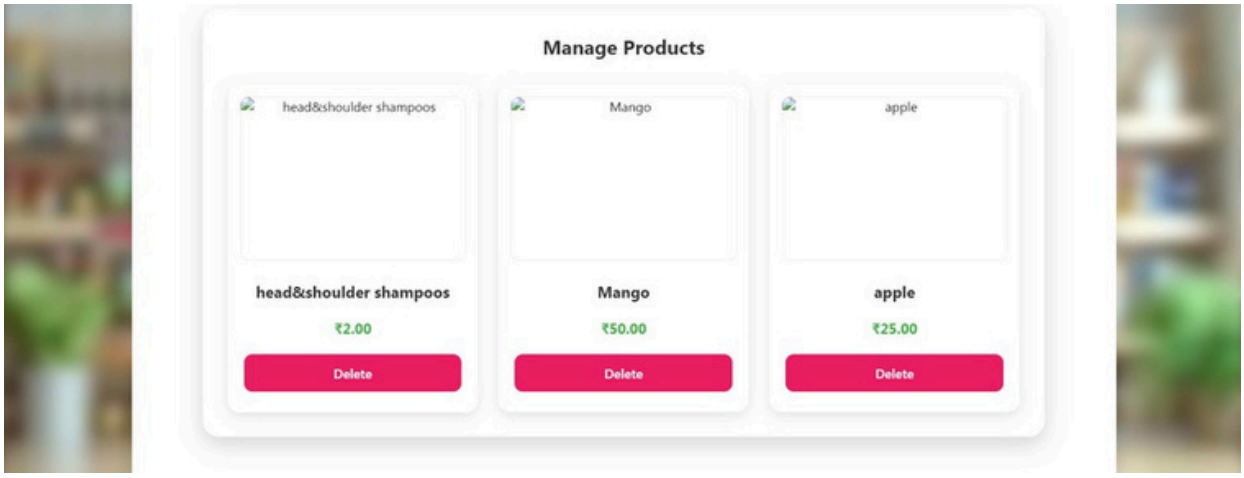
3.4. Checkout Page(checkout.html):

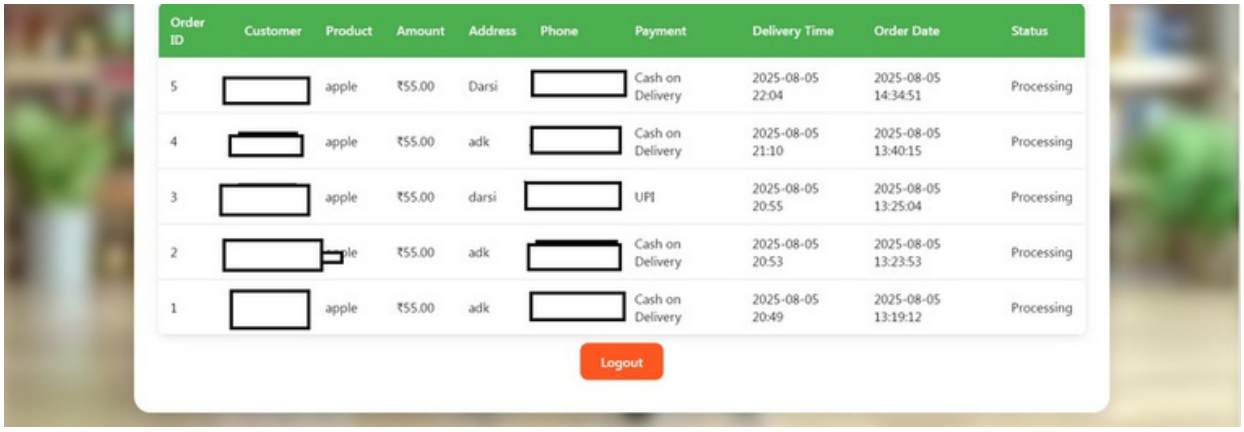


- **Order Summary:** Comprehensive review of purchase items
- **Delivery Information:** Address and contact details collection
- **Payment Options:** Multiple payment method selection
- **Form Validation:** Client and server-side input validation

3.6. Admin Dashboard(admin dashboard.html):







Order ID	Customer	Product	Amount	Address	Phone	Payment	Delivery Time	Order Date	Status
5	[redacted]	apple	₹55.00	Darsi	[redacted]	Cash on Delivery	2025-08-05 22:04	2025-08-05 14:34:51	Processing
4	[redacted]	apple	₹55.00	adk	[redacted]	Cash on Delivery	2025-08-05 21:10	2025-08-05 13:40:15	Processing
3	[redacted]	apple	₹55.00	darsi	[redacted]	UPI	2025-08-05 20:55	2025-08-05 13:25:04	Processing
2	[redacted]	[redacted]	₹55.00	adk	[redacted]	Cash on Delivery	2025-08-05 20:53	2025-08-05 13:23:53	Processing
1	[redacted]	apple	₹55.00	adk	[redacted]	Cash on Delivery	2025-08-05 20:49	2025-08-05 13:19:12	Processing

Logout

- **Management Interface:** Centralized control panel for store operations
- **Product Management:** Add new products and manage existing inventory
- **Order Overview:** Comprehensive order listing with status tracking
- **Statistical Insights:** Business metrics and performance indicators

3.Operations in The Grocery Store:

Customer Operations:

- **Registration Process:** New account creation with profile setup
- **Product Discovery:** Browsing, searching, and filtering products
- **Cart Management:** Adding, updating, and removing items
- **Checkout Process:** Order finalization with delivery and payment details
- **Order Tracking:** Monitoring order status and delivery progress
- **Account Management:** Updating personal information and preferences

Administrative Operations:

1. **Product Catalog Management:** Adding new products and updating existing ones
2. **Inventory Control:** Monitoring stock levels and updating availability
3. **Order Processing:** Reviewing, updating status, and managing fulfillment
4. **User Management:** Overseeing customer accounts and activities
5. **Business Analytics:** Reviewing sales data and performance metrics
6. **Content Management:** Updating product images and descriptions

System Operations:

1. **Authentication & Authorization:** Secure user login and role-based access
2. **Session Management:** Maintaining user state across browsing sessions
3. **Data Persistence:** Reliable storage and retrieval of product and order information

4.**Image Processing**: Handling product image uploads and storage

5.**Real-time Updates:** Dynamic cart and inventory updates

6.**Error Handling:** Graceful management of exceptions and user errors

5. python Code For online grocery store :

SourceCode:

```
from flask import Flask, render_template, request, redirect, session, url_for, jsonify, flash

import sqlite3

import os

from werkzeug.utils import secure_filename

from datetime import datetime, timedelta

import hashlib


app = Flask(__name__)

app.secret_key = 'supersecretkey'

DATABASE = 'grocery.db'

# Image upload configuration

UPLOAD_FOLDER = 'static/uploads/products'

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

app.config['MAX_CONTENT_LENGTH'] = 2 * 1024 * 1024 # 2MB limit
```

```
# Image upload configuration

UPLOAD_FOLDER = 'static/uploads/products'

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 2 * 1024 * 1024 # 2MB limit

# Create upload folder if not exists

os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

def allowed_file(filename):

 return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def hash_password(password):

 returnhashlib.sha256(password.encode()).hexdigest()

def init_db():

   withsqlite3.connect(DATABASE) as con:
      cur = con.cursor()
# Users table
      cur.execute("""
      CREATE TABLE IF NOT EXISTS users (
         idINTEGER PRIMARY KEY AUTOINCREMENT,
         username TEXT UNIQUE NOT NULL,
         password TEXT NOT NULL,
         phone TEXT DEFAULT '',
         address TEXT DEFAULT '',
         is_admin BOOLEAN DEFAULT 0
      )""")
```

```
#Adminstable

  # Admins table

   cur.execute("""

   CREATETABLE IF NOT EXISTS admins (

     idINTEGER PRIMARY KEY AUTOINCREMENT,

     username TEXT UNIQUE NOT NULL,

     password TEXT NOT NULL

   )""")


   #Products table

   cur.execute('''CREATE TABLE IF NOT EXISTS products (

            idINTEGER PRIMARY KEY AUTOINCREMENT,

            name TEXT,

            price REAL,

            image TEXT,

            description TEXT,

            category TEXT,

            stock INTEGER DEFAULT 100

          )''')
```

```
    #Carttable
    cur.execute('''CREATE TABLE IF NOT EXISTS cart (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT,
            product_id INTEGER,
            quantity INTEGER DEFAULT 1,
            added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )''')


    #Insertdefault admin
    try:
      hashed_password = hash_password("admin123")
        cur.execute("INSERT OR IGNORE INTO admins (username, password) VALUES (?,
?)",
              ("admin", hashed_password))
  #Insertsample products
      sample_products = [
        ("Apple", 50, "uploads/products/apple.jpg", "Fresh red apples", "Fruits", 100),
        ("Banana", 30, "uploads/products/banana.jpg", "Ripe bananas", "Fruits", 100),
```

```
("Banana",30, "uploads/products/banana.jpg", "Ripe bananas", "Fruits", 100),
            ("Milk", 25, "uploads/products/milk.jpg", "1L Fresh milk", "Dairy", 100)
        ]
                    cur.executemany("INSERT INTO products (name, price, image, description,
category,stock) VALUES (?, ?, ?, ?, ?, ?)",
                     sample_products)
        con.commit()
    exceptException as e:
        print(f"Error initializing database: {str(e)}")
        con.rollback()
@app.route('/')
def index():
 return render_template('index.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
   ifrequest.method == 'POST':
    uname = request.form.get('username', '').strip()
    pwd = request.form.get('password', '').strip()


    ifnot uname or not pwd:
        flash("Username and password are required", 'error'
```

```
return redirect('/login')


    hashed_pwd = hash_password(pwd)


    withsqlite3.connect(DATABASE) as con:
      #Checkin users table
      user=con.execute("SELECT * FROM users WHERE username=? AND password=?",
                  (uname, hashed_pwd)).fetchone()
      if user:
        session['user'] = uname
        session['is_admin'] = False
        return redirect('/products')
 #Checkinadmins table
                 admin = con.execute("SELECT * FROM admins WHERE username=? AND
password=?",
                   (uname, hashed_pwd)).fetchone()
      if admin:
        session['admin'] = uname
        session['is_admin'] = True
        returnredirect('/admin_dashboard')
```

```
        flash("Invalid credentials", 'error')
        return redirect('/login')
    return render_template('login.html')



# Add this search route
@app.route('/search')
def product_search():
    query=request.args.get('q', '').strip()
    if not query:
        return redirect('/products')
 withsqlite3.connect(DATABASE) as con:
        results=con.execute('''SELECT * FROM products
                        WHERE name LIKE ? OR description LIKE ? OR category LIKE ?
                        LIMIT 20''',
                      (f'%{query}%', f'%{query}%', f'%{query}%')).fetchall()


    returnrender_template('products.html', products=results, query=query)
```

```
# Add this product detail route
@app.route('/product/<int:product_id>')
def product_detail(product_id):
  if'user'notin session:
    return redirect('/login')


  withsqlite3.connect(DATABASE) as con:
    product=con.execute("SELECT * FROM products WHERE id=?",
                  (product_id,)).fetchone()
    if not product:
      flash("Product not found", 'error')
      return redirect('/products')
    #Getuser's order history for this product
    orders=con.execute('''SELECT quantity, status, order_date
                  FROM orders
                  WHERE username=? AND product_id=?
                  ORDER BY order_date DESC''',
                 (session['user'], product_id)).fetchall()
 returnrender_template('product_detail.html',
                product=product,
                user_orders=orders)
```

```
@app.route('/products')
def products():
    if 'user' not in session:
        return redirect('/login')


 with sqlite3.connect(DATABASE) as con:
        products = con.execute("SELECT * FROM products WHERE stock > 0").fetchall()
        cart_count = con.execute("SELECT COUNT(*) FROM cart WHERE username=?",
                                 (session['user'],)).fetchone()[0]


 return render_template('products.html', products=products, cart_count=cart_count)
@app.route('/product/basic/<int:product_id>')
def basic_product_detail(product_id):
    if 'user' not in session:
        return redirect('/login')
 with sqlite3.connect(DATABASE) as con:
        product = con.execute("SELECT * FROM products WHERE id=?",
                              (product_id,)).fetchone()
        if not product:
            flash("Product not found", 'error')
            return redirect('/products')
 return render_template('product_detail.html', product=product)
```

```
@app.route('/add_to_cart/<int:product_id>')
def add_to_cart(product_id):
  if'user'notin session:
    return redirect('/login')


  withsqlite3.connect(DATABASE) as con:
    stock=con.execute("SELECT stock FROM products WHERE id=?",
                  (product_id,)).fetchone()
    ifnotstock or stock[0] <= 0:
      flash("Product out of stock", 'error')
      return redirect('/products')


    existing=con.execute("SELECT * FROM cart WHERE username=? AND product_id=?",
                   (session['user'], product_id)).fetchone()
    if existing:
      con.execute("UPDATE cart SET quantity = quantity + 1 WHERE id=?",
              (existing[0],))
    else:
      con.execute("INSERT INTO cart (username, product_id) VALUES (?, ?)",
              (session['user'], product_id))
    con.commit(
  flash("Product added to cart", 'success')
  returnredirect('/products')
```

```
@app.route('/update_cart/<int:product_id>/<action>', methods=['POST'])
defupdate_cart(product_id, action):
  if'user'not in session:
    returnjsonify({'success': False, 'error': 'Not logged in'})


  withsqlite3.connect(DATABASE) as con:
            item= con.execute("SELECT quantity FROM cart WHERE username=? AND
product_id=?",
                (session['user'], product_id)).fetchone()


    if not item:
      returnjsonify({'success': False, 'error': 'Item not in cart'})


    new_quantity = item[0]
    if action == 'increase':
      stock=con.execute("SELECT stock FROM products WHERE id=?",
                (product_id,)).fetchone()[0]
      ifnew_quantity >= stock:
        returnjsonify({'success': False, 'error': 'Not enough stock'})
      new_quantity += 1
    elifaction== 'decrease' and item[0] > 1:
      new_quantity -= 1
```

```
            (new_quantity, session['user'], product_id))
        cart_items = con.execute('''SELECT p.id, p.name, p.price, c.quantity
                        FROM products p JOIN cart c ON p.id = c.product_id
                        WHERE c.username=?''', (session['user'],)).fetchall()
        cart_total = sum(item[2] * item[3] for item in cart_items)
        cart_count = sum(item[3] for item in cart_items)
        con.commit(
    return jsonify({
        'success': True,
        'newQuantity': new_quantity,
        'cartTotal': cart_total,
        'cartCount': cart_count,
        'grandTotal': cart_total + 30
    })
@app.route('/remove_from_cart/<int:product_id>')
def remove_from_cart(product_id):
    if'user'not in session:
        return redirect('/login')
    withsqlite3.connect(DATABASE) as con:
        con.execute("DELETE FROM cart WHERE username=? AND product_id=?",
            (session['user'], product_id))
```

```
 flash("Item removed from cart", 'success')

 return redirect('/view_cart')


@app.route('/view_cart')

def view_cart():

   if 'user' not in session:

      return redirect('/login')


   withsqlite3.connect(DATABASE) as con:

      cart_items=con.execute('''SELECT p.id, p.name, p.price, p.image, c.quantity, p.stock

                        FROM products p JOIN cart c ON p.id = c.product_id

                        WHERE c.username=?''', (session['user'],)).fetchall()

      total=sum(item[2] * item[4] for item in cart_items)

      cart_count=sum(item[4] for item in cart_items)


   returnrender_template('cart.html',

                  cart_items=cart_items,

                  total=total,

                  delivery_charge=30,

         @app.route('/checkout', methods=['GET', 'POST'])

def checkout():

   if 'user' not in session:

      returnredirect('/login')          cart_count=cart_count)
```

```
ififrequest.method== 'GET':
    withsqlite3.connect(DATABASE) as con:
      user=con.execute("SELECT address, phone FROM users WHERE username=?",
                (session['user'],)).fetchone()
      cart_items=con.execute('''SELECT p.id, p.name, p.price, c.quantity, p.stock
                    FROM products p JOIN cart c ON p.id = c.product_id
                    WHERE c.username=?''', (session['user'],)).fetchall()

      if not cart_items:
        flash("Yourcart is empty", 'error')
        return redirect('/products')

      total=sum(item[2] * item[3] for item in cart_items)
      delivery_charge= 30
      grand_total=total + delivery_charge

      out_of_stock=any(item[3] > item[4] for item in cart_items)
      if out_of_stock:
        flash("Someitems in your cart are out of stock", 'error')
        return redirect('/view_cart')
```

```
return render_template('checkout.html',
                    user=user,
                    cart_items=cart_items,
                    total=total,
                    delivery_charge=delivery_charge,
                    grand_total=grand_total)


  elifrequest.method== 'POST':
    address=request.form.get('address', '').strip()
    phone=request.form.get('phone', '').strip()
    payment_method= request.form.get('payment_method', 'COD').strip()


    ifnotaddressornot phone:
      flash("Addressand phone number are required", 'error')
      retu
    withsqlite3.connect(DATABASE) as con:
      try:
        cart_items=con.execute('''SELECT p.id, p.name, p.price, c.quantity, p.stock
                          FROM products p JOIN cart c ON p.id = c.product_id
                          WHERE c.username=?''', (session['user'],)).fetchall()
```

```
            delivery_time = (datetime.now() + timedelta(hours=2)).strftime("%Y-%m-%d
%H:%M")

        foritemin cart_items:
          product_id, name, price, quantity, stock = item

          ifquantity > stock:
            flash(f"Not enough stock for {name}", 'error')
            return redirect('/view_cart')
          con.execute('''INSERT INTO orders
                  (username, product_id, product_name, price, quantity,
                   address, phone, payment_method, delivery_time)
                  VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''',
                 (session['user'], product_id, name, price, quantity,
                  address, phone, payment_method, delivery_time))
          con.execute("UPDATE products SET stock = stock - ? WHERE id = ?",
                 (quantity, product_id))
        con.execute("DELETE FROM cart WHERE username = ?", (session['user'],))
        con.commit()
        flash("Order placed successfully!", 'success')
        returnredirect('/orders')
```

```
')

        except Exception as e:
            con.rollback()
            flash(f"Error processing order: {str(e)}", 'error')
            return redirect('/checkout')

@app.route('/register', methods=['GET', 'POST'])
def register():
    ifrequest.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '').strip()
        address = request.form.get('address', '').strip()
        phone = request.form.get('phone', '').strip()

        ifnot username or not password:
            flash("Username and password are required", 'error')
            return redirect('/register')

        hashed_password = hash_password(password)

        with sqlite3.connect(DATABASE) as con:
```

```
try:
            con.execute("INSERT INTO users (username, password, address, phone) VALUES
(?, ?, ?, ?)",
              (username, hashed_password, address, phone))
            con.commit()
            session['user'] = username
            flash('Registration successful!', 'success')
            return redirect('/products')
        except sqlite3.IntegrityError:
            flash('Username already exists', 'error')
            return redirect('/register')

    return render_template('register.html')

@app.route('/orders')
def user_orders():
    if'user' not in session:
        return redirect('/login')
```

```
  withsqlite3.connect(DATABASE) as con:
    orders=con.execute('''SELECT o.id, o.product_name, o.price, o.quantity,
                    o.delivery_charge, o.status, o.order_date
                    FROM orders o
                    WHERE o.username=?
                    ORDER BY o.order_date DESC''',
                  (session['user'],)).fetchall()
  returnrender_template('user_orders.html', orders=orders)
@app.route('/admin_dashboard')
defadmin_dashboard():
  if'admin' not in session:
    flash('Please login as admin to access this page', 'error')
    return redirect(url_for('admin_login'))


  try:
    withsqlite3.connect(DATABASE) as con:
      #Get products with proper type conversion
      products = []
      raw_products = con.execute("""
        SELECT id, name, price, stock
```

```
ORDERBYidDESC
        LIMIT 50
    """).fetchall()


    forproduct in raw_products:
        products.append((
            product[0],
            product[1],
            float(product[2]) if product[2] is not None else 0.0,
            product[3]
        ))


    #Getrecent orders with proper type conversion
    orders = []
    raw_orders = con.execute("""
        SELECTo.id, o.username, o.product_name, o.price, o.quantity,
            o.address, o.phone, o.payment_method, o.delivery_charge,
            o.delivery_time, o.order_date, o.status
        FROMorders o
        ORDERBY o.order_date DESC
        LIMIT 50
```

```
""").fetchall()


        fororder in raw_orders:
            orders.append((
                order[0], # id
                order[1], # username
                order[2], # product_name
                float(order[3]) if order[3] is not None else 0.0, # price
                order[4], # quantity
                order[5], # address
                order[6], # phone
                order[7], # payment_method
                float(order[8]) if order[8] is not None else 0.0, # delivery_charge
                order[9], # delivery_time
                order[10], # order_date
                order[11] # status
            ))
 # Get statistics
        stats = con.execute("""
            SELECT
                (SELECT COUNT(*) FROM products) as total_products,
                (SELECT COUNT(*) FROM orders) as total_orders,
```

```
(SELECTCOUNT(*) FROM users WHERE is_admin = 0) as total_users,
            (SELECT COUNT(*) FROM users WHERE is_admin = 1) as total_admins
        """).fetchone()


        stats = {
            'total_products': int(stats[0]) if stats[0] is not None else 0,
            'total_orders': int(stats[1]) if stats[1] is not None else 0,
            'total_users': int(stats[2]) if stats[2] is not None else 0,
            'total_admins': int(stats[3]) if stats[3] is not None else 0
        }
    returnrender_template('admin_dashboard.html',
                products=products,
                orders=orders,
                stats=stats)
 exceptsqlite3.Error as e:
    flash(f'Database error: {str(e)}', 'error')
    return redirect(url_for('admin_dashboard'))


@app.route('/admin_login', methods=['GET', 'POST'])
def admin_login():
 if request.method == 'POST':
    username = request.form.get('username', '').strip()
    password = request.form.get('password', '').strip()
```

```
ifnotusername or not password:
        flash('Username and password are required', 'error')
        returnredirect('/admin_login')

    hashed_password = hash_password(password)

    withsqlite3.connect(DATABASE) as con:
                admin = con.execute("SELECT * FROM admins WHERE username=? AND
password=?",
            (username, hashed_password)).fetchone()

        if admin:
            session['admin'] = username
            session['is_admin'] = True
            return redirect('/admin_dashboard')
        else:
            flash('Invalid admin credentials', 'error')
            return redirect('/admin_login')
  return render_template('admin_login.html')
@app.route('/add_product', methods=['POST'])
def add_product():
```

```
if 'admin' not in session:
    return redirect('/admin_login')

  name = request.form.get('name', '').strip()
  price = request.form.get('price', '0')
  description = request.form.get('description', '').strip()
  category = request.form.get('category', 'Other').strip()
  stock = request.form.get('stock', '100')

  ifnot name or not price:
    flash("Product name and price are required", 'error')
    return redirect('/admin_dashboard')

  file = request.files['image']

  iffile.filename == '':
    flash("No image selected", 'error')
    return redirect('/admin_dashboard')

  ifnot allowed_file(file.filename):
    flash("Invalid file type. Only PNG, JPG, JPEG, GIF allowed.", 'error')
    return redirect('/admin_dashboard')
```

```
try:
    filename = secure_filename(file.filename)
    filepath=os.path.join(app.config['UPLOAD_FOLDER'], filename)
    file.save(filepath)

    image_path = f"uploads/products/{filename}"

    withsqlite3.connect(DATABASE) as con:
      con.execute('''INSERT INTO products
                (name, price, image, description, category, stock)
                VALUES (?, ?, ?, ?, ?, ?)''',
              (name, price, image_path, description, category, stock))
      con.commit()

    flash("Product added successfully", 'success')
    re
  exceptException as e:
    flash(f"Error adding product: {str(e)}", 'error')
    returnredirect('/admin_dashboard')turn redirect('/admin_dashboard')
```

```
@app.route('/update_product/<int:product_id>', methods=['POST'])
def update_product(product_id):
    if'admin' not in session:
        return redirect('/admin_login')


    name = request.form.get('name', '').strip()
    price = request.form.get('price', '0')
    description = request.form.get('description', '').strip()
    category = request.form.get('category', 'Other').strip()
    stock = request.form.get('stock', '100')


    ifnot name or not price:
        flash("Product name and price are required", 'error')
        return redirect('/admin_dashboard')
  try:
        price = float(price)
        stock = int(stock)
    except ValueError:
        flash("Invalid price or stock value", 'error')
        return redirect('/admin_dashboard')
```

```
withsqlite3.connect(DATABASE) as con:
    if'image'inrequest.files and request.files['image'].filename != '':
      file = request.files['image']
      if allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath=os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)
        image_path = f"uploads/products/{filename}"


        old_image= con.execute("SELECT image FROM products WHERE id=?",
                    (product_id,)).fetchone()
        ifold_image and old_image[0]:
          try:
            os.remove(os.path.join('static', old_image[0]))
          except:
            pass
con.execute('''UPDATE products SET
                name=?, price=?, image=?, description=?, category=?, stock=?
                WHERE id=?''',
               (name, price, image_path, description, category, stock, product_id))
```

```
else:
        con.execute('''UPDATE products SET
                name=?, price=?, description=?, category=?, stock=?
                WHERE id=?''',
               (name, price, description, category, stock, product_id))


    con.commit()
  flash("Product updated successfully", 'success')
  return redirect('/admin_dashboard')
@app.route('/delete_product/<int:product_id>')
def delete_product(product_id):
 if 'admin' not in session:
    return redirect('/admin_login')
   with sqlite3.connect(DATABASE) as con:
    image_path = con.execute("SELECT image FROM products WHERE id=?",
                    (product_id,)).fetchone()
    if image_path and image_path[0]:
      try:
        os.remove(os.path.join('static', image_path[0]))
      except:
        pass
```

```
 con.execute("DELETE FROM products WHERE id=?", (product_id,))
    con.commit()

  flash("Product deleted successfully", 'success')
  return redirect('/admin_dashboard')

@app.route('/update_order_status/<int:order_id>', methods=['POST'])
def update_order_status(order_id):
  if'admin' not in session:
    return redirect('/admin_login')

  new_status = request.form.get('status', '').strip()
  if not new_status:
    flash("Status is required", 'error')
    return redirect('/admin_dashboard')

  withsqlite3.connect(DATABASE) as con:
    con.execute("UPDATE orders SET status=? WHERE id=?",
            (new_status, order_id))
    con.commit()
  flash("Order status updated", 'success')
  return redirect('/admin_dashboard')
```

```
@app.route('/get_cart_count')
def get_cart_count():
  if 'user' in session:
    with sqlite3.connect(DATABASE) as con:
      count = con.execute("SELECT COUNT(*) FROM cart WHERE username=?",
                  (session['user'],)).fetchone()[0]
    return jsonify({'count': count})
  return jsonify({'count': 0})
@app.route('/global-search')
def global_search():
  query=request.args.get('q', '').strip()
  if not query:
    returnredirect('/products')
  withsqlite3.connect(DATABASE) as con:
    results=con.execute('''SELECT * FROM products
                WHERE name LIKE ? OR description LIKE ? OR category LIKE ?
                LIMIT 20''',
               (f'%{query}%', f'%{query}%', f'%{query}%')).fetchall()

  returnrender_template('search_results.html', results=results, query=query)
```

```
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')


if __name__ == '__main__':
 if not os.path.exists(DATABASE):
        init_db()
    app.run(debug=True)
```

Conclusion:

Project Summary:

The Grocery Store web application represents a complete, functional e-commerce solution built using modern web development technologies. This project successfully demonstrates the implementation of a full-stack web application using Python Flask framework with SQLite database, providing a robust platform for online grocery shopping.

Key Achievements:

The application delivers a comprehensive shopping experience with separate interfaces for customers and administrators. Customers can seamlessly register accounts, browse products, manage shopping carts, and complete purchases, while administrators have full control over product management, inventory, and order processing. The system incorporates essential e-commerce features including user authentication, product catalog management, shopping cart functionality, order processing, and inventory tracking.

Technical Implementation :

Fromatechnicalperspective, the project showcases properimplementation of web application architecture followingtheModel- View-Controller pattern. The use of Flaskframeworkprovides a clean and maintainable codebase,whileSQLite offers a lightweight yet powerful databasesolution.The frontend combines HTML templateswithCSSstyling and JavaScript functionality to createanintuitiveuser interface that works across different devices.

Security and Reliability:

The application incorporates important security measures including password hashing, SQL injection prevention, session management, and input validation. The transactional approach to database operations ensures data integrity, particularly during critical processes like order creation and inventory updates. Error handling and user feedback mechanisms provide a smooth user experience even when unexpected situations occur.