**PERSONALIZED DIET RECOMMENDER FOR FITNESS CENTRE**

A Project Report submitted to Madurai Kamaraj university

In Partial Fulfilment of the Requirement for the award of the Degree of

**BACHELOR OF SCIENCE**

**(2022-2025)**

Submitted by

**S ABINANTHAN**

**[22BIT153]**

***Under the Guidance of***

**Ms.T.SHUNMUGA PRIYA,M.E,MBA.,**



**THE AMERICAN COLLEGE**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

(An Autonomous institution affiliated to Madurai Kamaraj University)

Re-accredited (3rd cycle) by NAAC with Grade "A+" CGPA-3.47 on a 4-point scale

Madurai 625002.

**THE AMERICAN COLLEGE**

(An Autonomous institution affiliated to Madurai Kamaraj University)

Re-accredited (3 cycle) by NAAC with Grade "A" CGPA-3.47 on a 4-point scale

Madurai-625002.



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BONAFIDE CERTIFICATE**

This is to certify that the project work entitled as "**Personalized Diet Recommender for Fitness** **Centre**" Submitted by **S.ABINANTHAN [22BIT153]** in partial fulfilment of requirements for the award of the Degree of Bachelor of Science(Information Technology), The American College, Madurai a bonafide record of the work done by him during the year 2022-2025.

**Ms.T.Shunmuga Priya M.E, MBA.,**  **Mrs. J. Christy Jeeva Ratna Devi MCA., M.Phil., (Ph.D)**

**Internal guide** **Head of the Department**

Submitted for the Viva-Voce examination held on

**Internal Examiner**

**DECLARATION**

I hereby declare that this project was carried out by me under the guidance **Ms.T.SHUNMUGA PRIYA, M.E,MBA.**, Assistant Professor, Department of Information Technology, The American College, Madurai. I also declare that this project report is the result of my own effort and that has not been copied from anyone and has not been submitted by anybody in any other Universities / Institution/Research Centers**.**

**Place: Madurai Signature of the Student**

**Date: S.Abinanthan**

**[22BIT153]**

**ACKNOWLEDGEMENT**

I am grateful to convey my sincere thanks to my principal **Dr. M. DAVAMANI CHRISTOBER M.Sc., M.Phil., M.Ed., PGDCA., Ph.D(Math)., Ph.D(Inter Disciplinary)** for his support and encouragement.

I am thankful to our Head & Assistant Professor of the Department of Information Technology **Mrs. J. Christy Jeeva Ratna Devi MCA., M.Phil., (Ph.D)** for her constant support.

I express my deep sense of Gratitude and indebtedness to my internal guide **Ms.T.Shunmuga Priya, M.E,MBA.,** Assistant Professor of Department of Information Technology for him inspiring guidance and continuous motivation encouragement which has enabled me to complete this project.

Also, I would like to thank my Faculty Members. Parents, and Friends who have helped me during the course of completion of this project.

|  |  |  |
| --- | --- | --- |
| **S.NO** | **DESCRIPTION** | **PAGE NO** |
| 1 | Introduction  1.1 Abstract |  |
| 2 | System Analysis  2.1 Existing system  2.2 Proposed system  2.3 Feasibility study |  |
| 3 | System Requirement Specification  3.1 Hardware Requirements  3.2 Software Requirements  3.3 About Software |  |
| 4 | System Design  4.1 Modules Description  4.2 Function Requirement  4.2.1 Data Flow Diagram  4.3 Database design |  |
| 5 | System Implementation |  |
| 6 | System Testing |  |
| 7 | System Maintenance |  |
| 8 | Conclusion |  |
| 9 | Future Enhancement |  |
| 10 | Bibliography |  |
| 11 | Appendix  11.1 Sample User Interface screen  11.2 Sample source code |  |

**CONTENT**

**1.INTRODUCTION**

* 1. **ABSTRACT**

The BMI Calculator with Diet and Workout Plan is a web-based application that helps users assess their Body Mass Index (BMI) and receive personalized health recommendations. The system calculates BMI based on user-provided height and weight, categorizing individuals into different health levels such as underweight, normal, overweight, or obese. This classification serves as the foundation for generating a structured diet and workout plan tailored to the user’s needs.

The application features a BMI calculator, a diet plan module, and a workout plan module, all integrated into an interactive and user-friendly interface. Users can register, log in, and access their personalized recommendations. Once BMI is calculated, a diet plan button appears, leading to a detailed weekly meal plan based on the BMI category. After selecting the diet plan, a workout plan button appears, providing customized exercise routines to help users achieve their fitness goals. Additionally, users can generate a PDF report summarizing their BMI, diet, and workout plan.

Developed using Django for the backend and HTML, CSS, and JavaScript for the frontend, this project ensures a smooth and responsive user experience. The system aims to assist individuals in adopting a healthier lifestyle by providing scientifically backed dietary and workout suggestions. With its structured approach to BMI-based health improvement, this application serves as a valuable tool for fitness-conscious users.

1. **SYSTEM ANALYSIS**

**2.1 Existing System**

The existing system of pharmacy wholesale management is largely manual and inefficient. Pharmacies often need to visit wholesale stores or place orders by phone, leading to delays and errors. Maintaining records of products, orders, and transactions is time-consuming and prone to inaccuracies.

Pharmacies struggle to access up-to-date product information, including availability and pricing. Without a centralized system, tracking stock levels and new arrivals is difficult. This often leads to order modifications or cancellations due to outdated information.

Wholesalers face challenges in managing customers, products, and orders efficiently. The absence of a unified platform results in fragmented data and inconsistent record-keeping. This lack of organization hampers smooth operations and decision-making.

**2.2 Proposed System**

The proposed system for Curakart is an automated and user-friendly online platform for pharmacy wholesalers. It enables pharmacies to browse products, place orders, and manage their profiles seamlessly. This digital approach eliminates the need for manual order placement and reduces errors.

The system provides a centralized database for managing product information, orders, and user details efficiently. Wholesalers can update product availability, pricing, and stock levels in real-time. This ensures that pharmacies have access to accurate and current information.

Admins can manage products, users, and orders through a dedicated dashboard, streamlining daily operations. The platform also supports secure login, registration, and profile management for users. Overall, the system improves efficiency, accuracy, and the overall user experience.

**2.3 Feasibility Study**

1. **Technical Feasibility**

The Curakart project is technically feasible as it uses proven and reliable technologies. The frontend is built using HTML, CSS, and JavaScript, providing a responsive and interactive user interface. The backend is developed with Django, a robust and secure web framework, and the database is managed using MySQL. These technologies are well-documented, widely used, and suitable for developing a scalable and efficient web application.

**2. Operational Feasibility**

Curakart is designed to streamline the pharmacy wholesale process, making it highly feasible for users. The system offers easy navigation for browsing products, placing orders, and managing profiles. Admins can efficiently manage products, users, and orders, reducing the administrative burden. The platform’s user-friendly interface ensures smooth adoption and usability.

**3. Economic Feasibility**

The project requires minimal financial investment as it utilizes open-source technologies. The cost mainly includes server hosting, domain registration, and SMS gateway integration. The potential benefits outweigh the costs as the system reduces manual workload, minimizes errors, and increases operational efficiency, leading to long-term savings and improved customer satisfaction.

**4. Legal Feasibility**

The project complies with data protection regulations by implementing secure login, registration, and profile management. User data is stored securely, and the system ensures data integrity and confidentiality. Compliance with legal requirements related to online wholesale and pharmacy management is considered throughout the development.

#### **5. Schedule Feasibility:**

The project is being developed in phases to ensure timely completion. Key features like login, registration, profile management, and basic product listing are already completed. Upcoming phases will focus on order placement, payment integration and admin functionalities. The structured approach ensures steady progress and timely delivery.

#### **6. Social Feasibility:**

Curakart promotes digital transformation in the pharmacy wholesale sector, benefiting both wholesalers and pharmacies. By reducing manual tasks, the system fosters a more efficient business environment. Additionally, providing real-time product availability and order management improves customer relationships and satisfaction.

**3.System Requirement Specification**

**3.1 Hardware Requirement**

Processor: ryzen 5

Ram: 16GB

Storage: 512

**3.2 Software Requirement**

**Django:** Web framework for building backend logic and APIs.

**Python:** Programming language used with Django.

**MySQL:** Relational database for storing user data, product information, order.

**3.3 About Software**

The Curakart pharmacy wholesale website project is designed to streamline the process of managing and purchasing pharmaceutical products in bulk. The frontend is developed using HTML, CSS, and JavaScript, providing a visually appealing and interactive user experience. The backend is powered by Django, a robust Python framework that efficiently handles server-side operations and user authentication. Data management is carried out using MySQL, which stores information related to users, products, and orders. The platform currently features user registration, login, password reset, profile management, and browsing through categorized products.

Future updates will include the functionality to place and manage orders, enabling customers to make bulk purchases seamlessly. Additionally, the admin dashboard will offer comprehensive control, allowing administrators to manage users, products, and order records effectively. The system is designed to be scalable and adaptable, ensuring that it meets the evolving needs of pharmacy wholesalers. With its intuitive interface and efficient backend processing, Curakart aims to become a reliable platform for bulk pharmaceutical transactions.

1. **System Design**

**4.1 Modules Description**

· User Management Module

· Product Management Module

· Category Management Module

· Order Management Module

· Admin Dashboard Module

**Modules Description**

**User Management Module:**

This module handles user-related operations, including registration, login, password management, and profile updates. It ensures secure user authentication and profile maintenance.

**Product Management Module:**

This module allows administrators to add, update, and delete products, while users can view, search, and filter products. It organizes product data and displays information effectively to the users.

**Category Management Module:**

This module categorizes products into various groups, enabling users to easily browse products by type. Admins can manage categories by adding, updating, or deleting them.

**Order Management Module :**

This module will handle order placement, tracking, and history. Users will be able to view their past orders, while admins can update order statuses and manage order records.

**Admin Dashboard Module:**

This module provides administrators with a comprehensive control panel to manage users, products, and orders. It offers a user-friendly interface for efficient platform management.

**4.2 Functional Requirement**

### **1. User Management**

**1.1 User Registration:**

* The system shall allow users to create an account by providing personal details (name, email, phone number) and login credentials (username, password).
* The system shall validate the registration data before creating an account.
* The system shall send a confirmation email upon successful registration.
  1. **User Login:**
* The system shall allow users to log in using valid credentials.
* The system shall maintain a session for authenticated users.
* The system shall provide a 'Remember Me' option for easier future access.
  1. **Password Management:**
* The system shall provide an option to reset the password via email verification.
* The system shall allow users to change their password from the profile page.

**1.4 Profile Management:**

* The system shall allow users to view and update their profile information.
* The system shall support profile picture upload and update.

### **2. Product Management**

**2.1 Product Listings:**

* The system shall display available products with details such as name, price, description, and images.
* The system shall allow users to filter and sort products by category, brand, and price.

**2.2 Product Details:**

* The system shall provide a detailed view of each product, including availability and specifications.
* The system shall show related products on the product detail page.

**2.3 Product Search:**

* The system shall provide a search functionality to find products by name or category.
* The system shall display search suggestions based on input.

**2.4 Product Inventory Management (Admin):**

* The system shall allow the admin to add, update, and delete products.
* The system shall track inventory levels and show low-stock alerts.

### **3.Order Management**

**3.1 Order Placement:**

* The system shall allow users to place orders by selecting products and specifying quantities.
* The system shall display an order summary before final confirmation.

**3.2 Order Tracking:**

* The system shall display the current status of placed orders (processing, shipped, delivered).
* The system shall update users when the order status changes.

**3.3 Order History:**

* The system shall allow users to view their previous orders and details.

**3.4 Order Management (Admin):**

* The system shall allow the admin to view and manage all orders.
* The system shall enable the admin to update order statuses.

### **4. Payment**

**4.1 Payment Method:**

* The system shall support Cash on Delivery (COD) as the primary payment method.

### **5. Admin Dashboard**

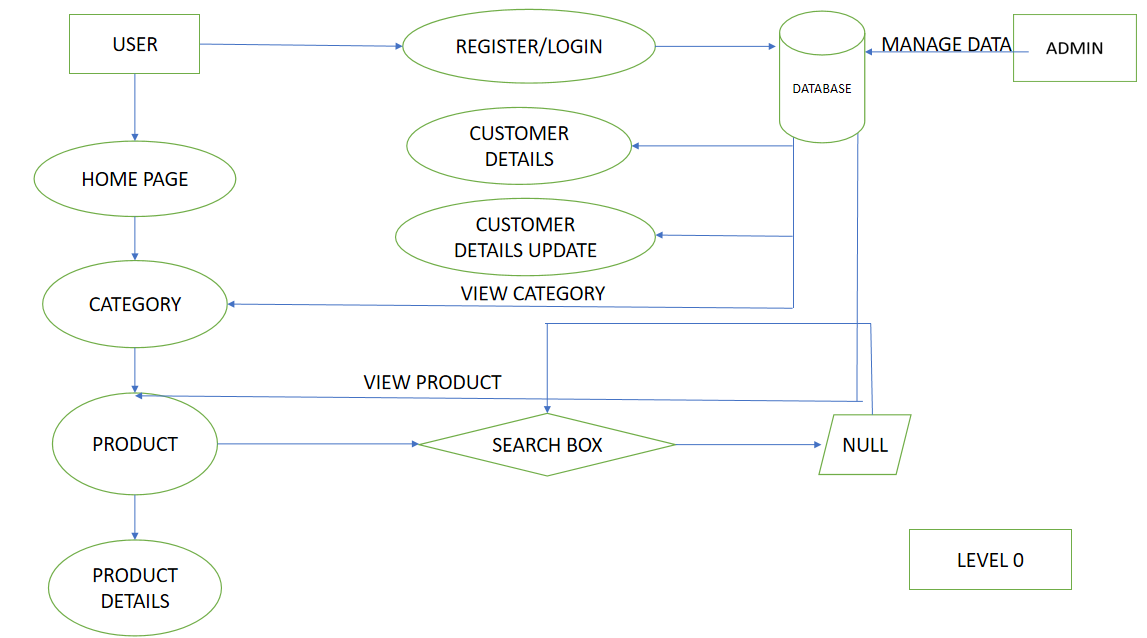
* The system shall provide a dashboard for admins to manage products, users, and orders.
* The system shall generate reports for sales, orders, and user activities.

### **6. Security and Authentication**

* The system shall enforce role-based access control.
* The system shall encrypt sensitive data.
* The system shall provide session management and auto-logout for inactivity.

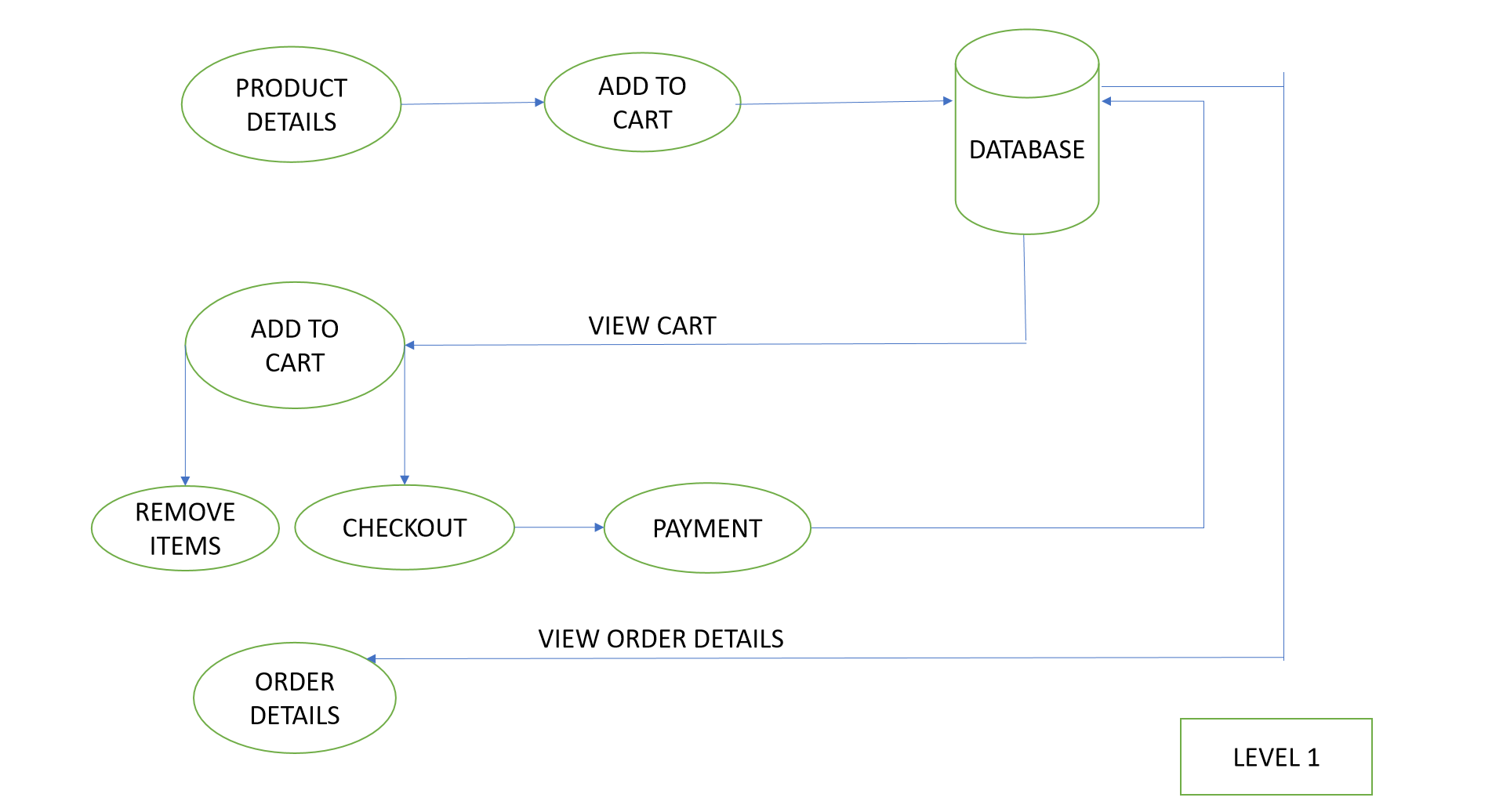
**4.2.1 Data Flow Diagram**

**Level 0 DFD:**

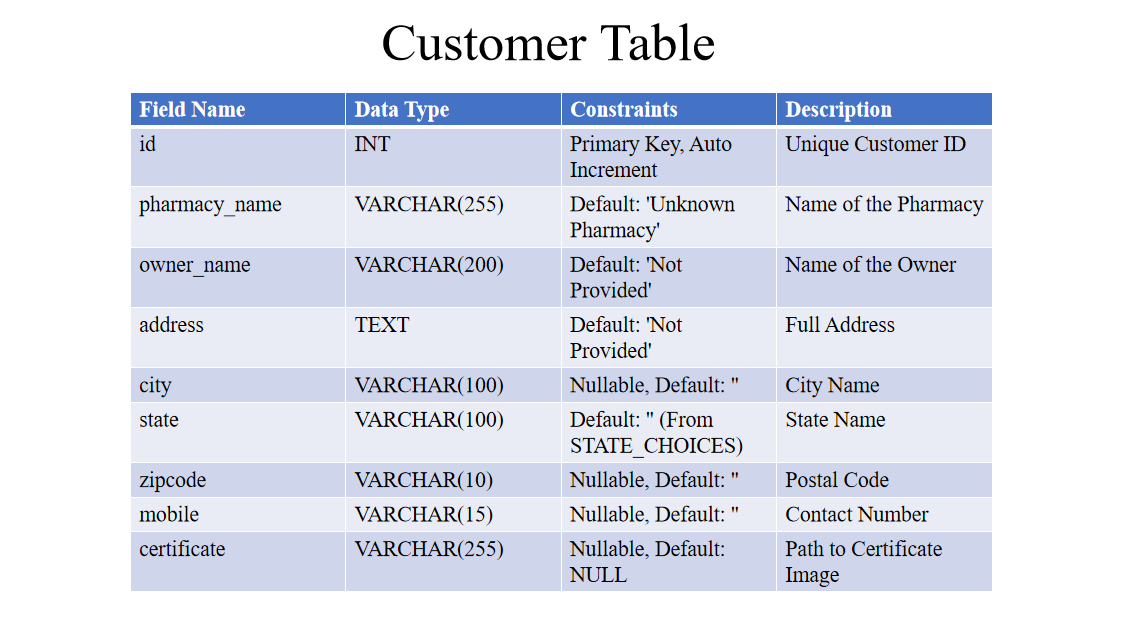


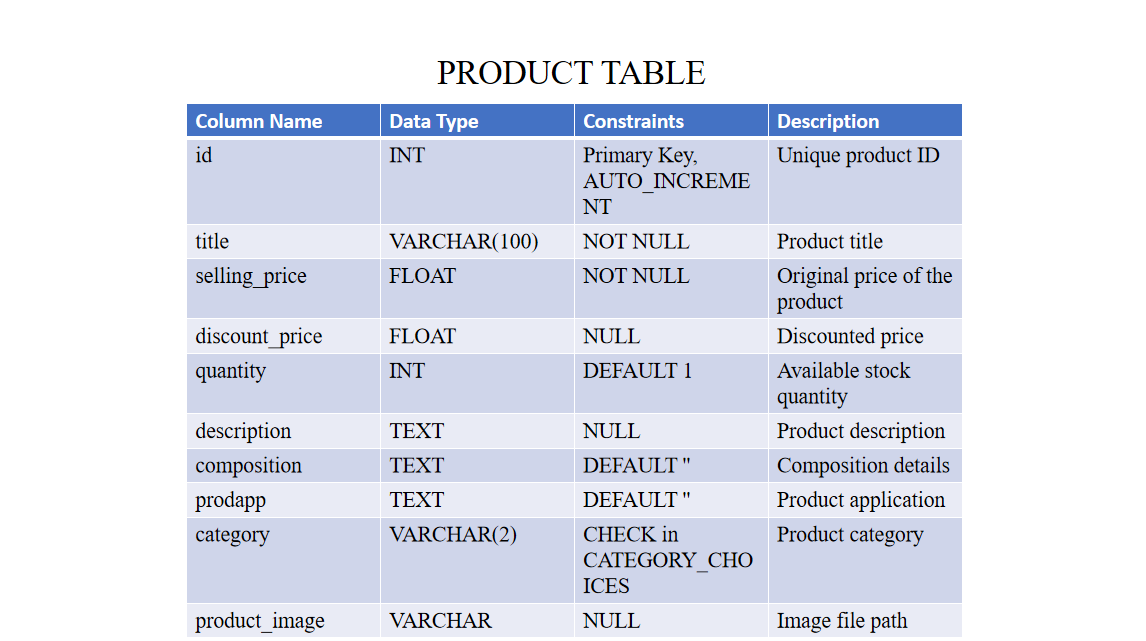
**Level 1 DFD:**

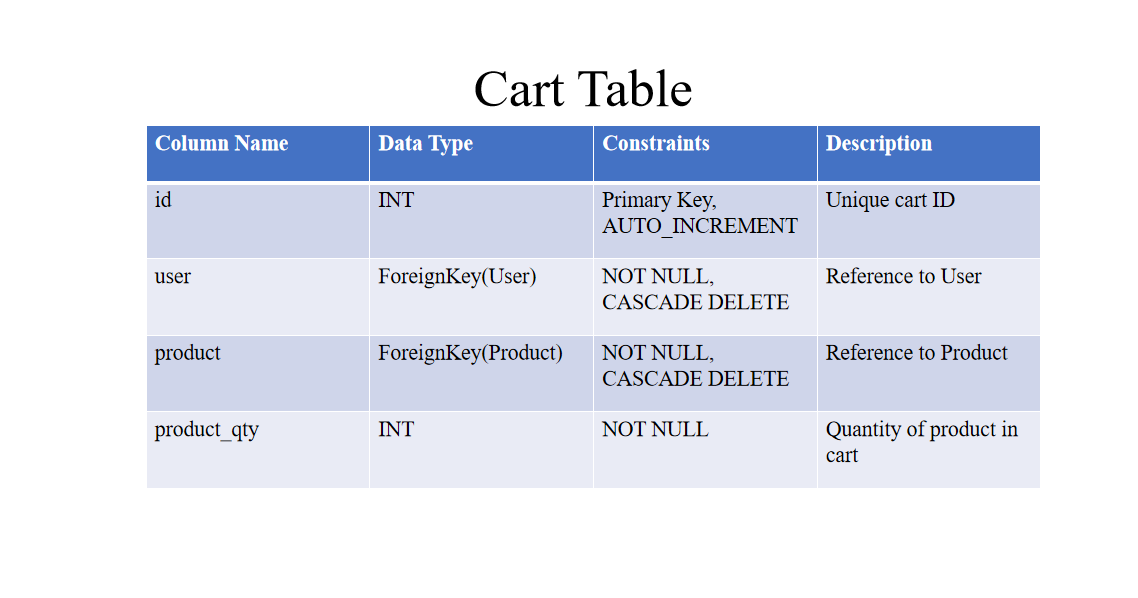
**Data Flow**

****

**4.3 Database Design**



****

****

**5.System Implementation**

### **1.System Architecture**

The system follows a three-tier architecture:

* **Frontend**: Built using HTML, CSS, and JavaScript for user interaction.
* **Backend**: Implemented with Django, handling business logic, authentication, and data processing.
* **Database**: MySQL, used for storing user information, products, orders, and more.

### **2.Frontend Implementation**

* Uses HTML for structure, CSS for styling, and JavaScript for dynamic content.
* Responsive design for mobile and desktop compatibility.
* Key pages include Login, Registration, Home, Profile, Category, and Order Placement.

### **3.Backend Implementation**

* Django framework manages user authentication, profile management, and order processing.
* Uses Django’s ORM to interact with the MySQL database.
* Admin functionalities include product, user, and order management.

### **4.Backend Implementation**

* Django framework manages user authentication, profile management, and order processing.
* Uses Django’s ORM to interact with the MySQL database.
* Admin functionalities include product, user, and order management.

### **5.Database Design**

* **Users Table**: Stores user credentials and profile details.
* **Products Table**: Contains product names, descriptions, prices, and categories.
* **Orders Table**: Records placed orders, associated users, products, and order status.

### **6.Key Features**

* **User Authentication**: Login, registration, and password reset functionalities.
* **Profile Management**: Users can view and update profile details.
* **Product Display**: Categorized product listing with search and filter options.
* **Order Placement**: Users can place orders for selected products.
* **Admin Panel**: Manage products, users, and orders efficiently.

### **7.Future Enhancements**

**Order Tracking**: Real-time order status updates.

**Payment Integration**: Secure online payment methods.

**Advanced Reporting**: Generate sales and user activity reports for admin

**6.System Testing**

**1.Unit Testing**

* Verify individual components such as login, registration, and profile update.
* Test functions separately to ensure they perform as expected.
* Check for edge cases and error handling.
* Validate input data formats and response messages.

### **2.Integration Testing**

* Ensure smooth interaction between frontend, backend, and database.
* Test data flow across different modules (e.g., order placement).
* Check integration of user authentication with profile management.
* Validate data consistency after integrating different components.

### **3.Integration Testing**

* Ensure smooth interaction between frontend, backend, and database.
* Test data flow across different modules (e.g., order placement).
* Check integration of user authentication with profile management.
* Validate data consistency after integrating different components.

**7.System Maintenance**

**1.Regular Updates**

* Keep the Django framework, libraries, and dependencies up to date.
* Update the MySQL database for security patches and performance improvements.
* Regularly update frontend libraries like jQuery or Bootstrap (if used).
* Check for deprecated packages and replace them promptly.

**2.Data Backup and Recovery**

* Schedule regular backups of the MySQL database to prevent data loss.
* Maintain automated backup scripts and periodically test recovery procedures.
* Store backups securely, both on-site and on cloud storage.
* Implement a rollback plan in case of failed updates.

**3.Performance Monitoring**

* Monitor server load and database performance using tools like Grafana or New Relic.
* Track response times for critical features like login and order placement.
* Analyze logs to identify performance bottlenecks or unusual traffic spikes.
* Optimize queries and code that cause slowdowns.

**4.Security Management**

* Regularly scan for vulnerabilities using tools like OWASP ZAP.
* Monitor login attempts and block IPs showing suspicious activity.
* Enforce strong password policies and multi-factor authentication.
* Keep SSL certificates updated to ensure secure connections.

**5.Bug Fixing and User Support**

* Maintain a bug tracking system (like JIRA or GitHub Issues) for reporting and fixing issues.
* Monitor user feedback and address common issues promptly.
* Keep a record of solved bugs for future reference.
* Communicate maintenance schedules to users to minimize disruption.

**Conclusion:**

The Curakart pharmacy wholesale website successfully addresses the needs of pharmacy wholesalers by providing a comprehensive platform for product management, order placement, and user administration. Built with a robust combination of frontend technologies (HTML, CSS, JavaScript), a powerful backend using Django, and a reliable MySQL database, the system ensures seamless user interactions and efficient data handling.

The implementation of user authentication, profile management, product categorization, and an admin panel for product and order management highlights the system’s practicality and efficiency. Rigorous testing, including unit, integration, system, and security testing, has been conducted to ensure the platform's stability and security.

Future enhancements, such as order tracking, secure payment integration, and advanced reporting, are planned to further improve the system’s functionality. The project demonstrates a balanced approach to user-friendly design and robust backend processing, making it a reliable solution for pharmacy wholesalers.

**Future Enhancement:**

**1.Order Tracking**

* Integrate a real-time order tracking feature to allow users to monitor the status and progress of their orders.
* Send automated notifications via email or SMS to update users on order status changes.

**2.Payment Integration**

* Implement secure online payment methods such as credit/debit cards, UPI, and net banking.
* Integrate a payment gateway (like Razorpay, Stripe, or PayPal) to handle transactions.
* Enable invoice generation and digital receipts for each purchase.

**3.Advanced Reporting and Analytics**

* Generate detailed reports on sales, product performance, and user activity.
* Integrate graphical dashboards to visualize sales trends and customer data.
* Provide data export options for analytics and record-keeping.

**4.Enhanced Admin Panel**

* Introduce role-based access control to manage different levels of administrative privileges.
* Enable bulk product upload and update through CSV or Excel files.
* Provide comprehensive logs for tracking admin actions and system changes.

**4.User Experience Improvements**

* Personalize the user dashboard with order history, favorite products, and recommended items.
* Add product reviews and ratings to help users make informed purchasing decisions.
* Implement a wishlist feature to save products for future orders.

**5.Mobile App Integration**

* Develop a mobile application for Android and iOS to enhance accessibility.
* Synchronize the app with the web version to ensure seamless data consistency.
* Implement push notifications for order updates and promotional offers.

**Bibilography**

**1.Django Documentation**

* Django Software Foundation. (n.d.). Django Documentation. Retrieved from <https://docs.djangoproject.com/>
* Used for implementing backend features, user authentication, and database management.

**2.MySQL Reference Manual**

* Oracle Corporation. (n.d.). MySQL 8.0 Reference Manual. Retrieved from <https://dev.mysql.com/doc/>
* Utilized for database design, data handling, and query optimization.

**3.HTML and CSS Standards**

* World Wide Web Consortium (W3C). (n.d.). HTML & CSS Specifications. Retrieved from <https://www.w3.org/>
* Followed best practices for frontend development and responsive design.

**4.JavaScript Guidelines**

* Mozilla Developer Network (MDN). (n.d.). JavaScript Guide. Retrieved from <https://developer.mozilla.org/>
* Referred for implementing dynamic content and client-side scripting.

**5.Bootstrap Documentation**

* Bootstrap. (n.d.). Bootstrap 5 Documentation. Retrieved from <https://getbootstrap.com/>
* Used for responsive and interactive UI elements.

**6.Security Best Practices**

* OWASP Foundation. (n.d.). OWASP Top Ten. Retrieved from <https://owasp.org/>
* Applied security measures to mitigate vulnerabilities like SQL injection and XSS.

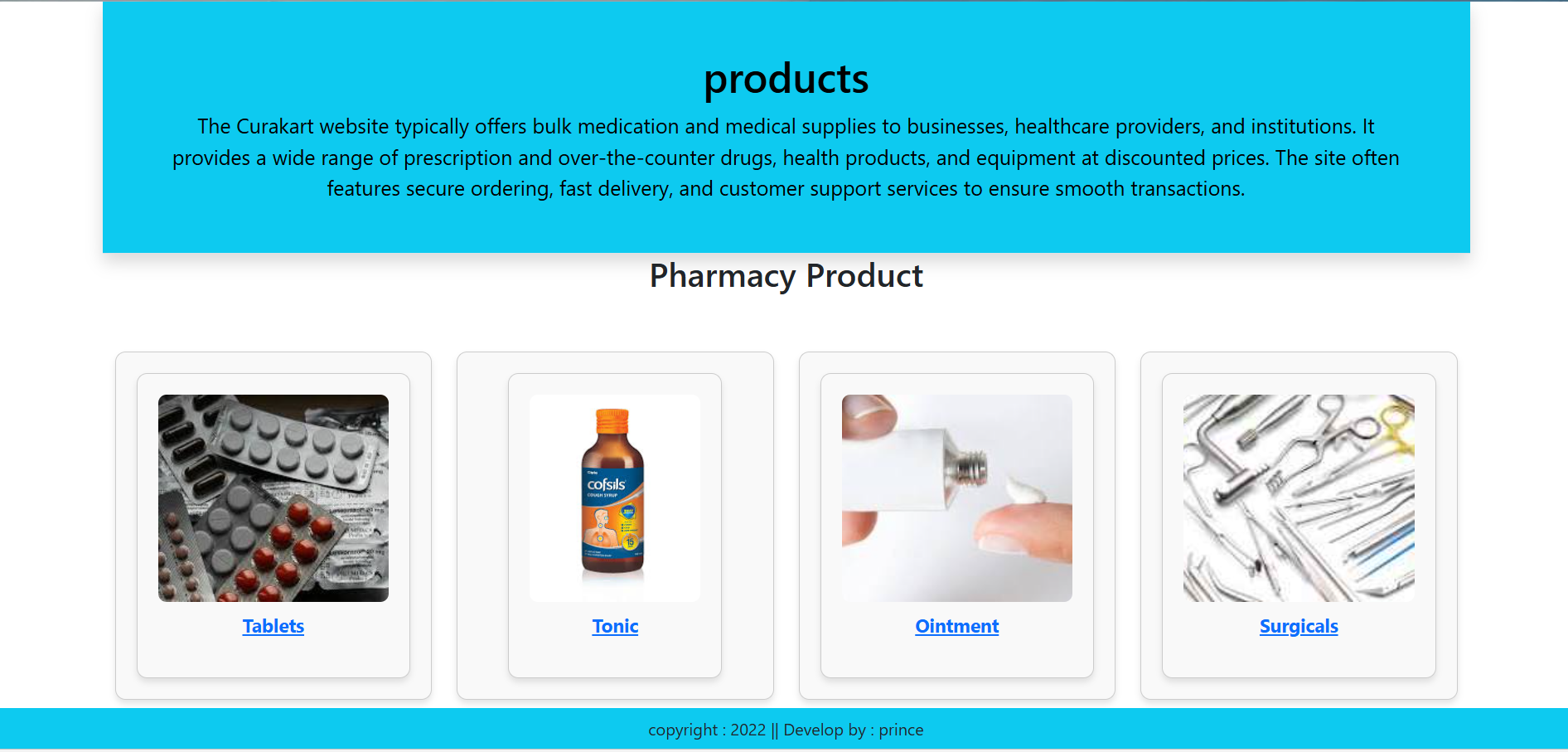
**7.Software Testing Principles**

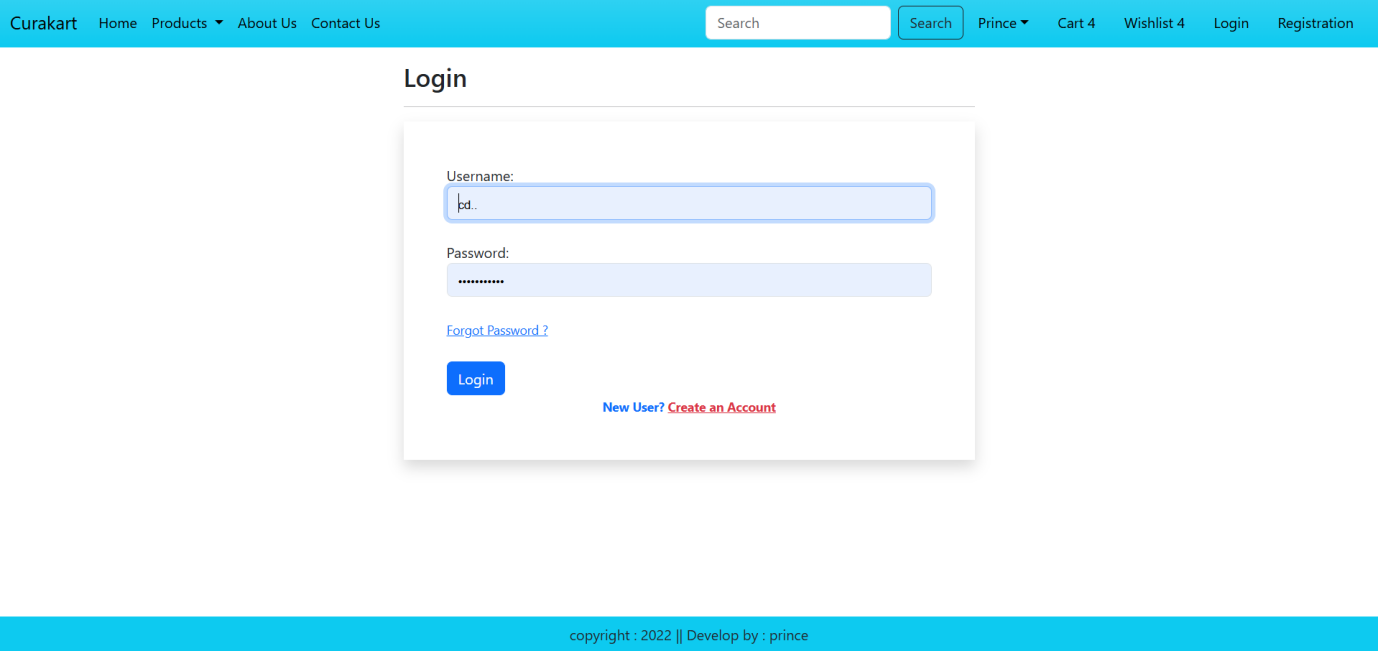
* Pressman, R. S. (2014). Software Engineering: A Practitioner’s Approach. McGraw-Hill.
* Used for designing testing strategies, including unit and integration testing.

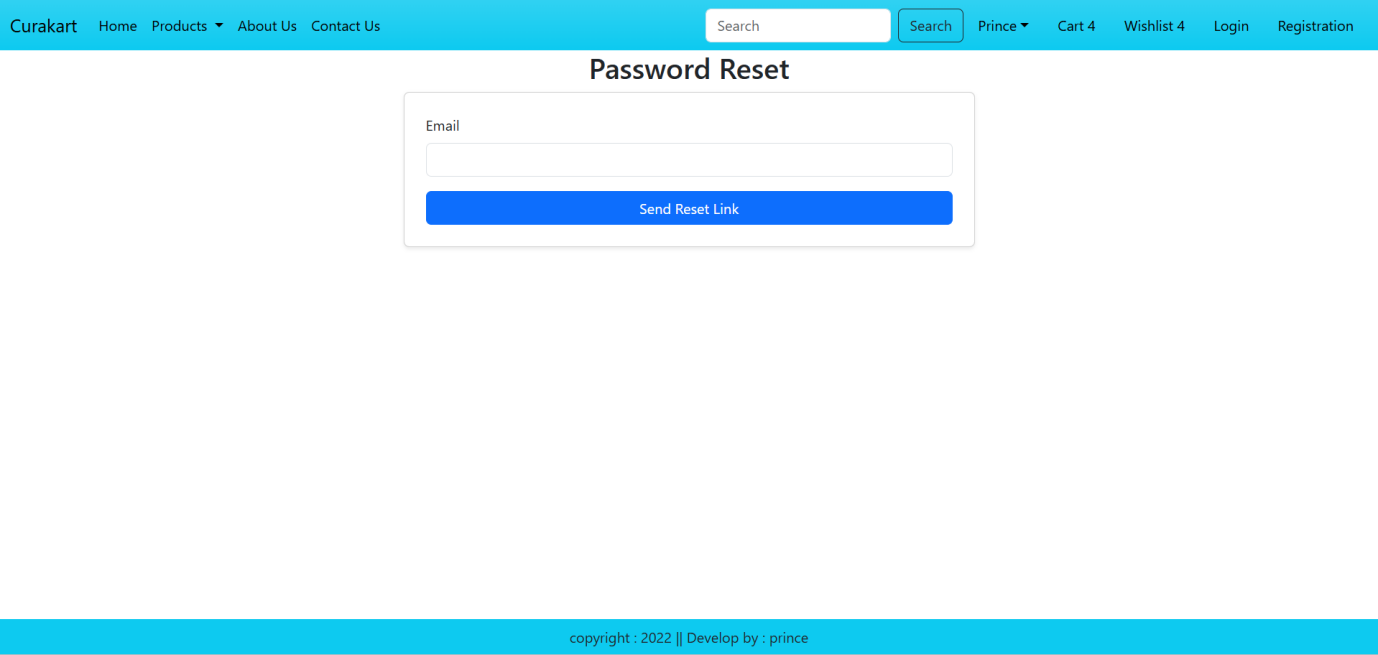
**Appendix**

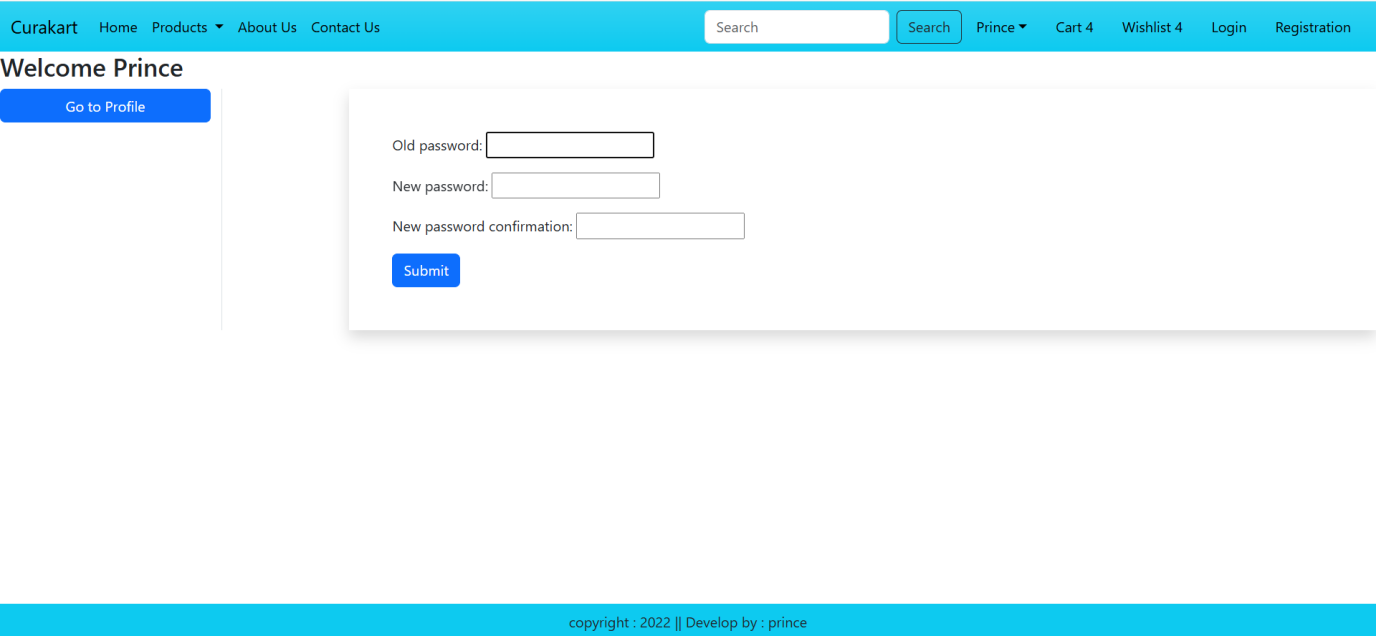
**11.1 Sample User Interface screen**

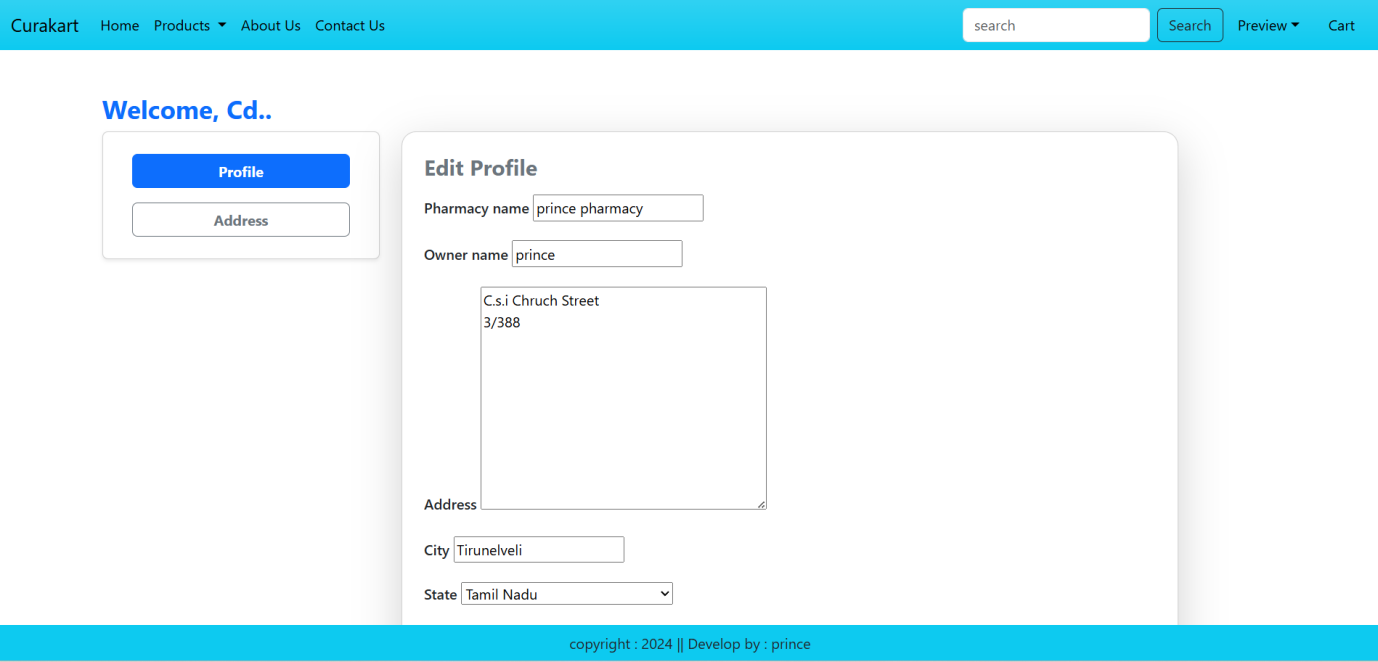


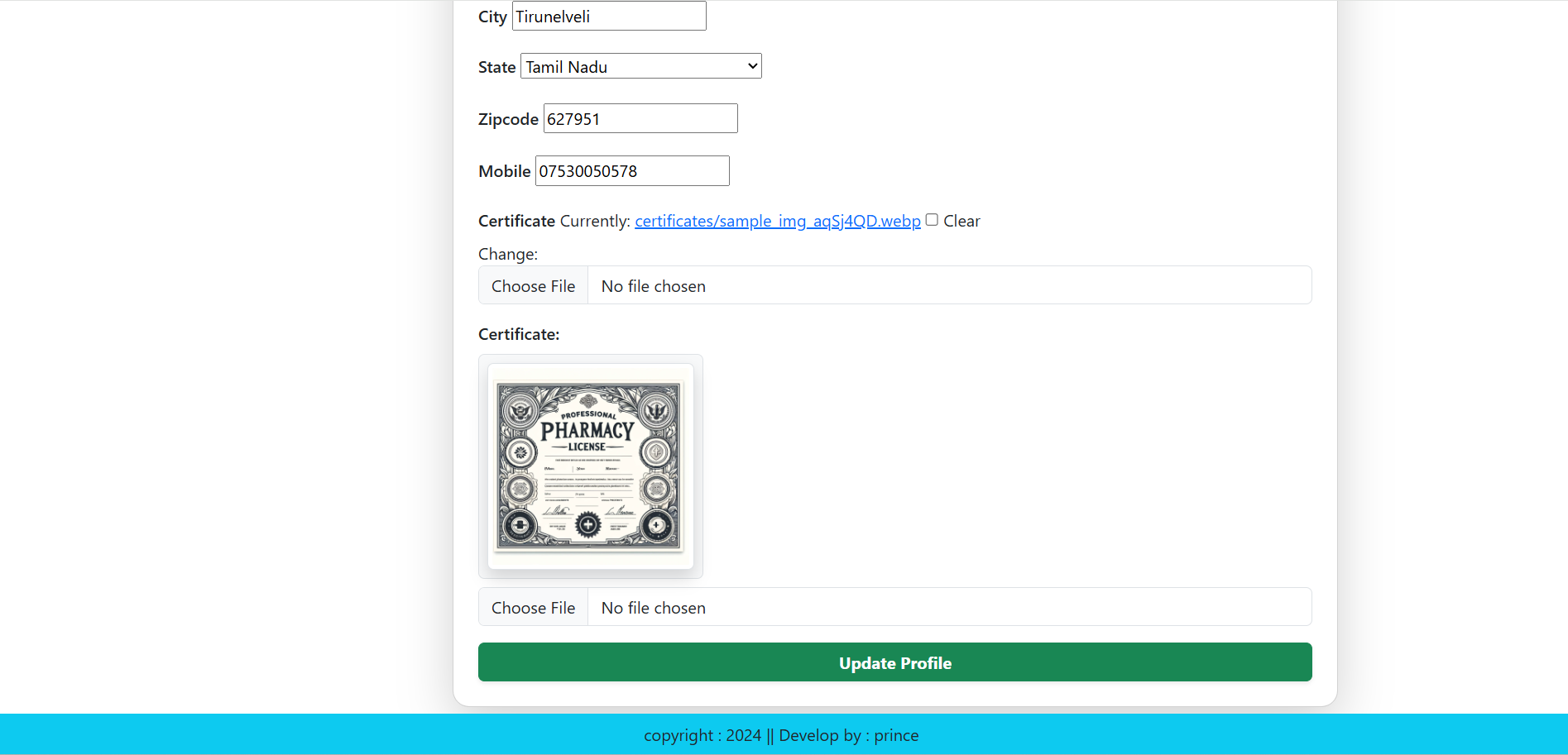


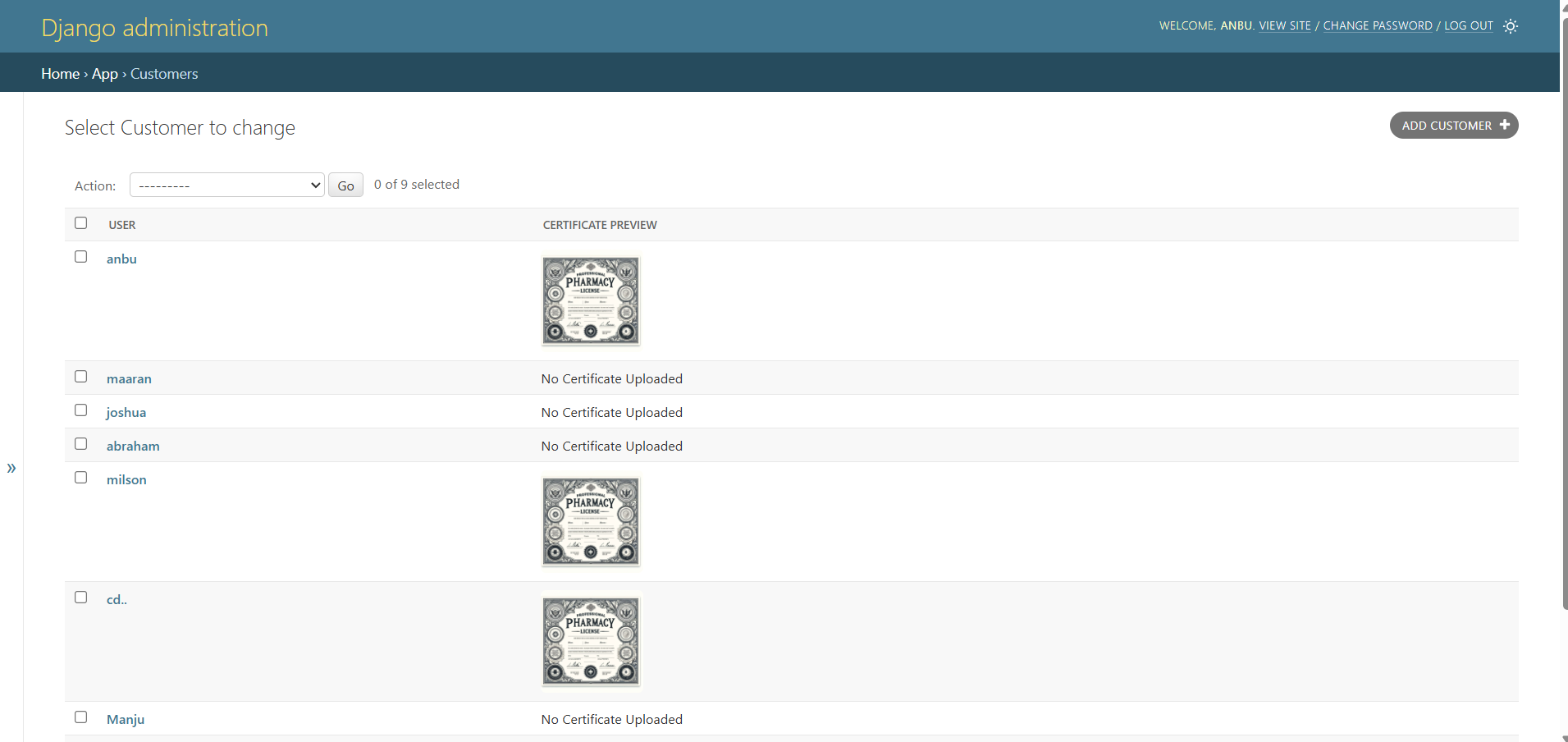
****

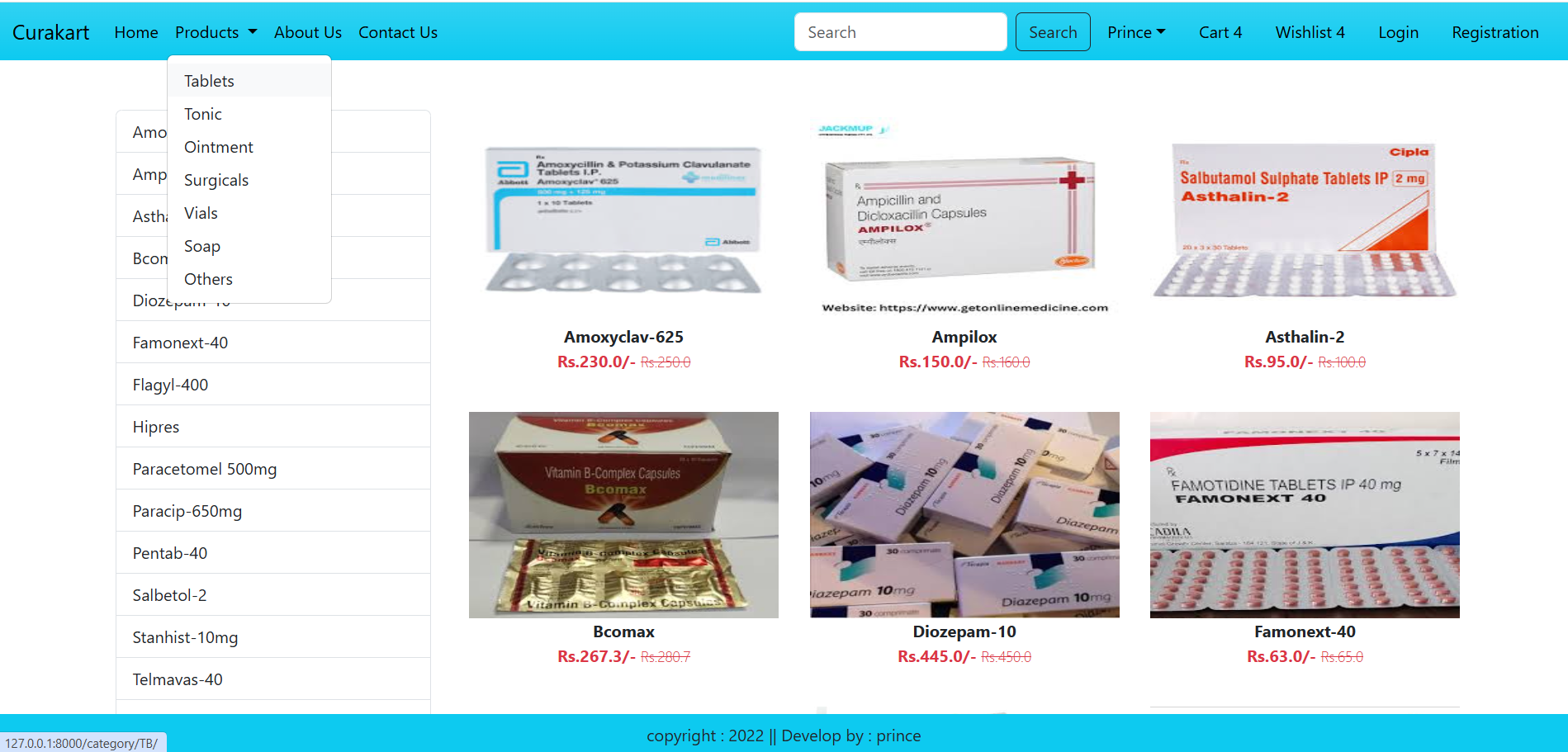
****

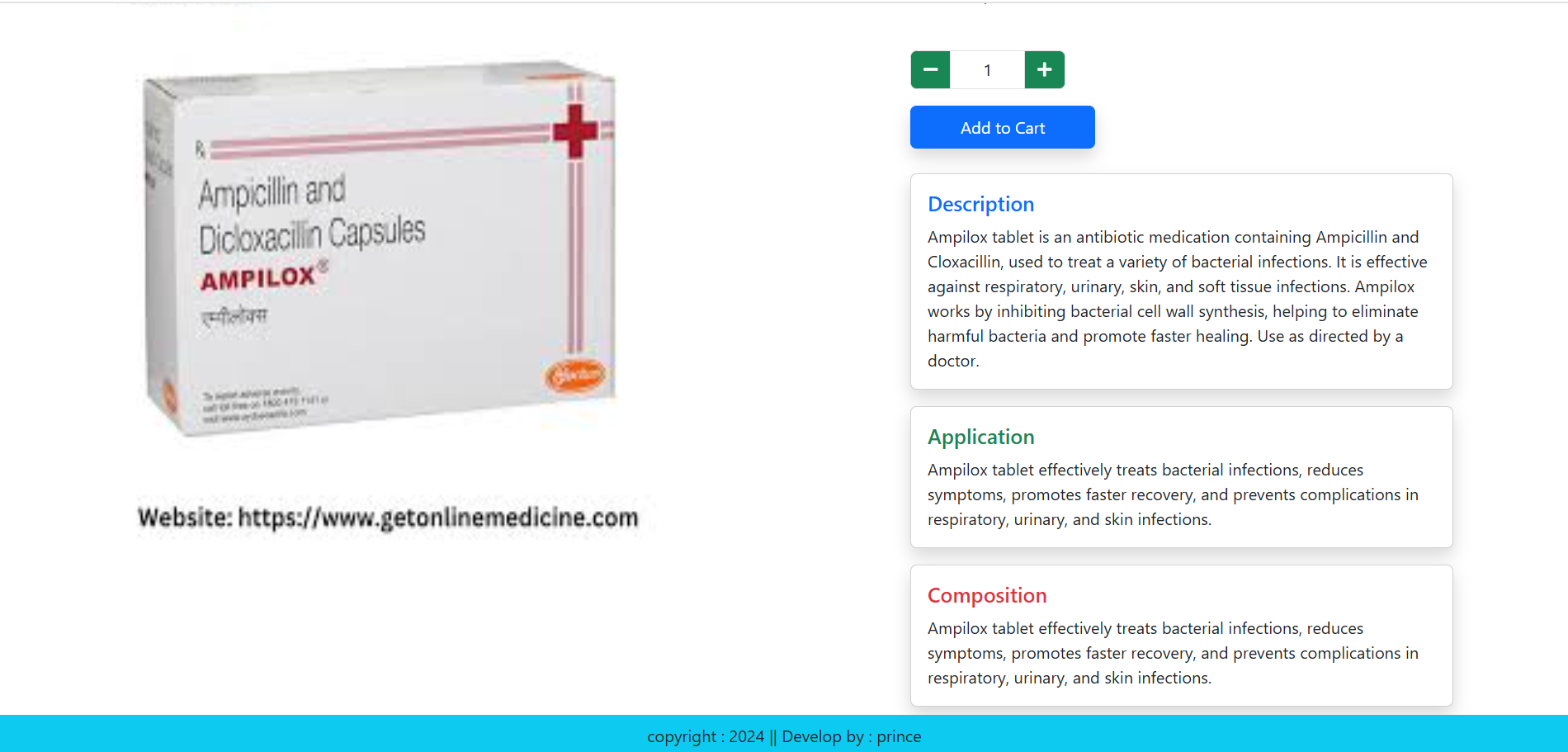
****

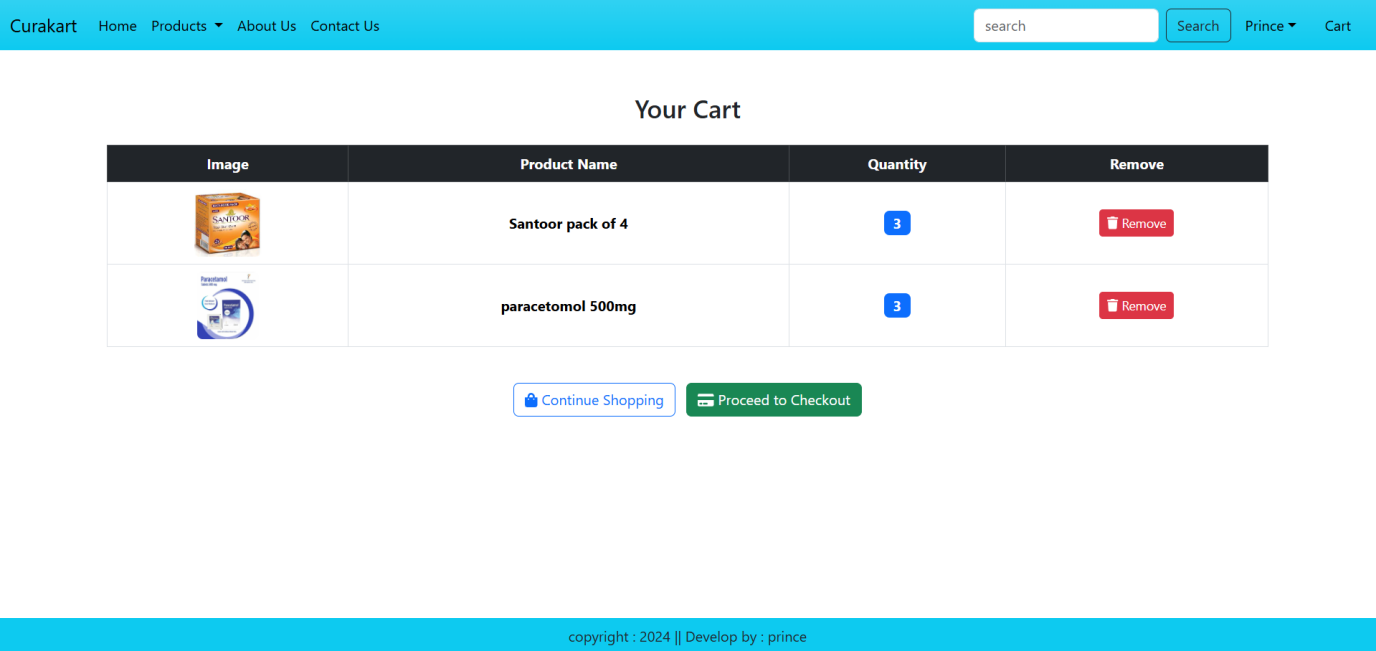
****

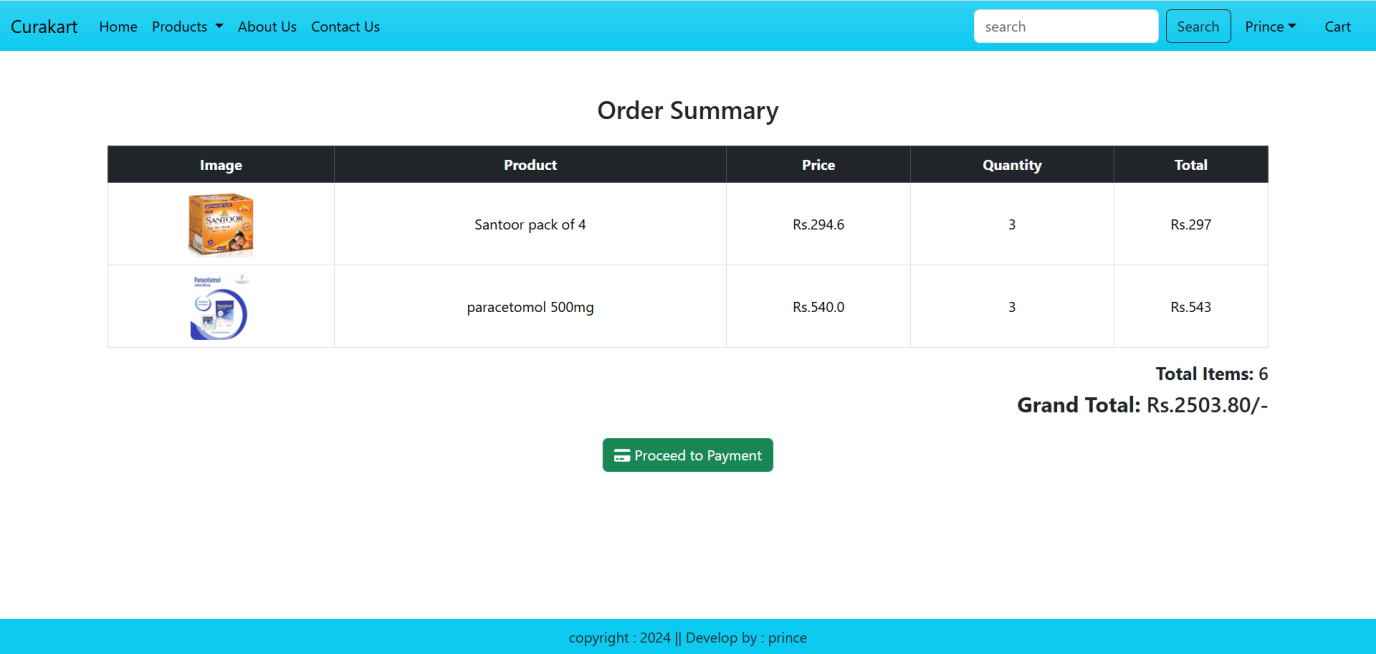
****

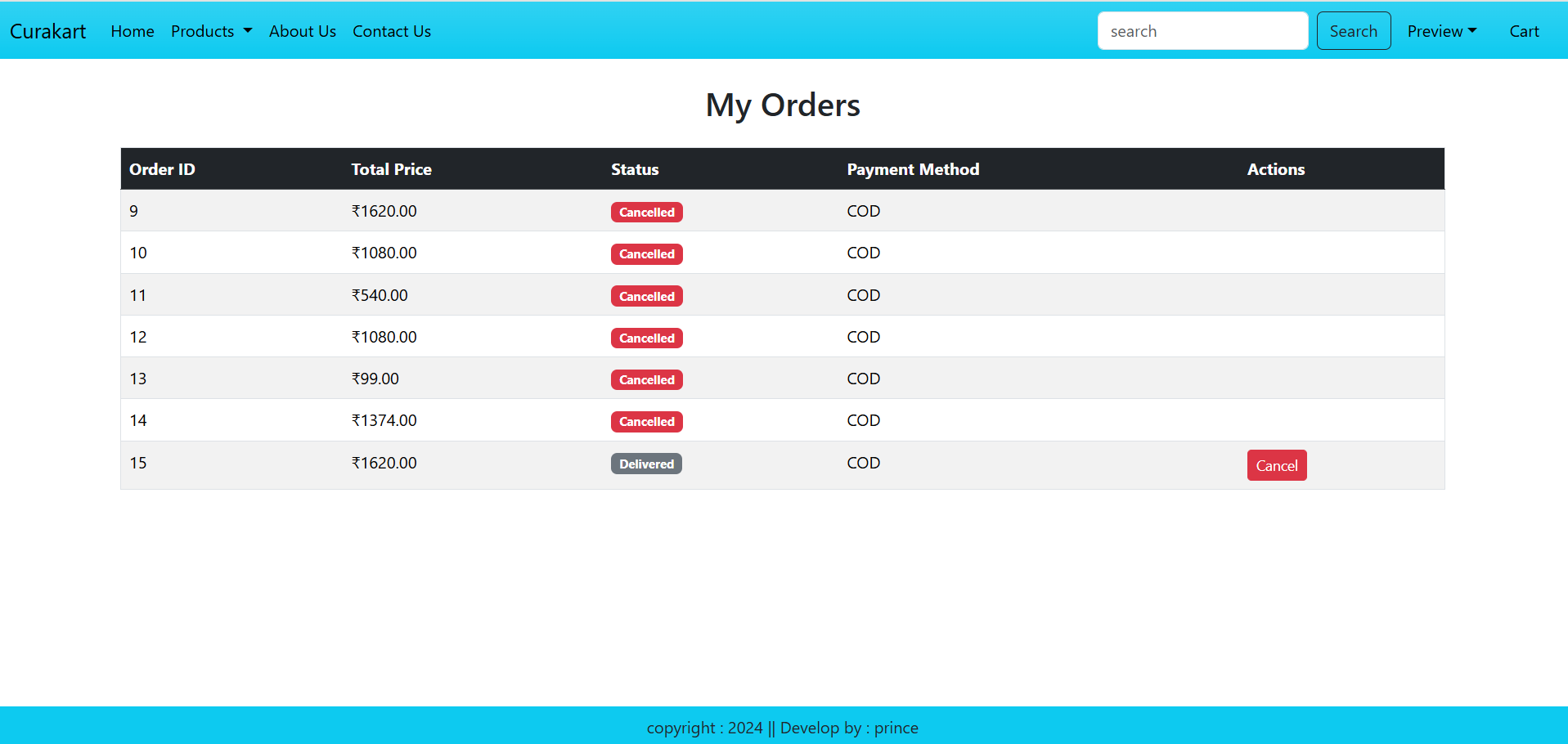
****

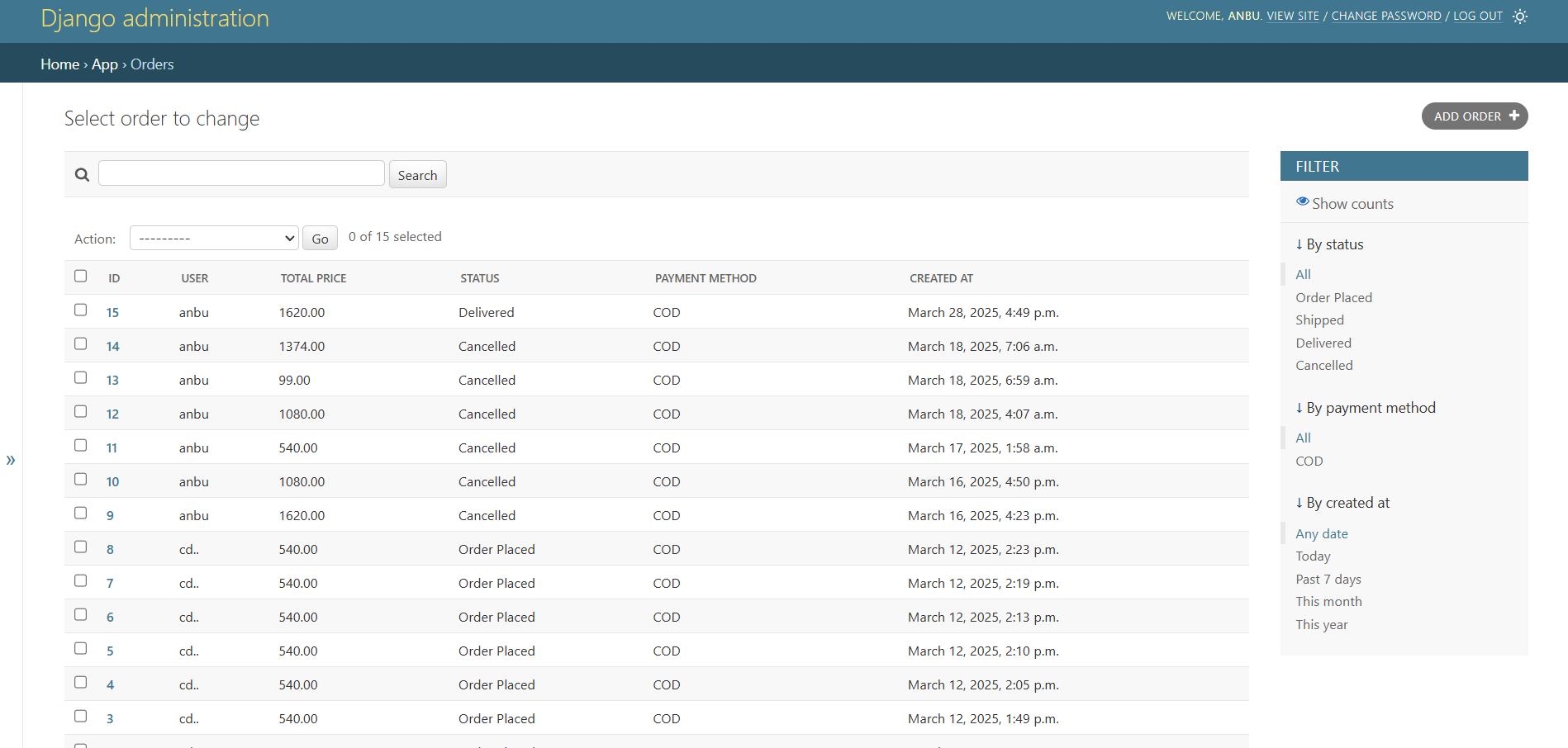
****

****

****

****

****

****

**11.2 Sample source code**

**Base.html**

<!DOCTYPE html>

{% load static %}

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- External CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

<!-- Custom CSS Files -->

<link rel="stylesheet" href="{% static 'app/css/owl.carousel.main.css' %}">

<link rel="stylesheet" href="{% static 'app/css/all.main.css' %}">

<link rel="stylesheet" href="{% static 'app/css/style.css' %}">

<title>pharmacy product | {% block title %}{% endblock title %}</title>

</head>

<body>

<!-- Navbar -->

<nav class="navbar navbar-expand-lg navbar-dark bg-info bg-gradient">

<div class="container-fluid">

<a class="navbar-brand text-black" href="#" >Curakart</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">

{% if request.user.is\_authenticated %}

<li class="nav-item">

<a class="nav-link active text-black" aria-current="page" href="{%url 'home' %}">Home</a>

</li>

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle text-black" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

Products

</a>

<ul class="dropdown-menu">

<li><a class="dropdown-item" href="{%url 'category\_detail' 'TB'%}">Tablets</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'TO'%}">Tonic</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'OI'%}">Ointment</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'SU'%}">Surgicals</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'VI'%}">Vials</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'SO'%}">Soap</a></li>

<li><a class="dropdown-item" href="{%url 'category\_detail' 'OT'%}">Others</a></li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link text-black" href="{%url 'about'%}">About Us</a>

</li>

<li class="nav-item">

<a class="nav-link text-black" href="{%url 'contact'%}">Contact Us</a>

</li>

</ul>

<form class="d-flex" role="search" action="/search">

<input class="form-control me-2" type="search"placeholder="search" name="search" aria-label="search">

<button class="btn btn-outline-dark"type="submit">Search</button>

</form>

</div>

<ul class="navbar-nav me-auto mb-2 mb-lg-0">

<li class="nav-item dropdown mx-2">

<a class="nav-link dropdown-toggle text-black" href="#" id="profiledropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">Preview</a>

<ul class="dropdown-menu" aria-labelledby="profiledropdown">

<li><a class="dropdown-item" href="{% url 'profile' %}">Profile</a></li>

<li><a class="dropdown-item" href="{% url 'my\_orders' %}">Orders</a></li>

<li><a class="dropdown-item" href="{% url 'password\_change'%}">Change Password</a></li>

<li><a class="dropdown-item" href="{% url 'logout'%}"> Logout</a></li>

</ul>

</li>

<li class="nav-item mx-2">

<a href="{% url 'cart' %}" class="nav-link text-black">Cart </a>

</li>

{%else%}

<li class="nav-item mx-2">

<a href="{%url 'login' %}" class="nav-link text-black">Login</a>

</li>

<li class="nav-item mx-2">

<a href="{% url 'customer\_registration' %}" class="nav-link text-black">Registration</a>

</li>

{%endif%}

</ul>

</div>

</nav>

{% if messages %}

<div class="container mt-3">

{% for message in messages %}

<div class="alert alert-{{ message.tags }}">{{ message }}</div>

{% endfor %}

</div>

{% endif %}

<!-- Main Content Blocks -->

{% block banner\_slider %}{% endblock banner\_slider %}

{% block information %}{% endblock information %}

{% block main-content %}{% endblock main-content %}

<!-- Footer -->

<footer class="container-fluid fixed-bottom bg-info text-center p-2 mt-5">

copyright : 2024 || Develop by : prince

</footer>

<!-- External JS Scripts -->

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js" integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.min.js" integrity="sha384-0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCsOGabYfZo0T0to5eqruptLy" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/jquery.min.js" integrity="sha512-v2CJ7UaYy4JwqLDIrZUI/4hqeoQieOmAZNXBeQyjo21dadnwR+8ZaIJVT8EE2iyI61OV8e6M8PP2/4hpQINQ/g==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>

<!-- Custom JS Files -->

<script src="{% static 'app/js/owl.carousel.main.js' %}"></script>

<script src="{% static 'app/js/all.main.js' %}"></script>

<script src="{% static 'app/js/myscript.js' %}"></script>

</body>

</html>

**Url.py**

from django.urls import path

from . import views

from .views import CustomerRegistrationView

from django.contrib.auth import views as auth\_view

from .forms import LoginForm, MyPasswordResetForm, PasswordChangeForm, MySetPasswordForm

from .views import custom\_logout\_view

from .views import remove\_cart\_item

from django.conf.urls.static import static

from django.conf import settings

from .views import custom\_admin\_login # Import the function

from .views import my\_orders,cancel\_order

from django.contrib import admin

import logging

# Create a logger

logger = logging.getLogger(\_\_name\_\_)

# Set the logging level

logger.setLevel(logging.DEBUG)

# Create a file handler and a stream handler

file\_handler = logging.FileHandler('app.log')

stream\_handler = logging.StreamHandler()

# Create a formatter and set it for the handlers

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

file\_handler.setFormatter(formatter)

stream\_handler.setFormatter(formatter)

# Add the handlers to the logger

logger.addHandler(file\_handler)

logger.addHandler(stream\_handler)

# Add logs to the views

from django.views.decorators.http import require\_http\_methods

@require\_http\_methods(["GET"])

def home(request):

logger.info('Home page accessed')

return render(request, 'home.html')

@require\_http\_methods(["GET"])

def about(request):

logger.info('About page accessed')

return render(request, 'about.html')

@require\_http\_methods(["GET"])

def contact(request):

logger.info('Contact page accessed')

return render(request, 'contact.html')

# Add logs to the views in the views.py file

from django.views import View

from django.shortcuts import render

class CategoryView(View):

def get(self, request, \*args, \*\*kwargs):

logger.info('Category page accessed')

return render(request, 'category.html')

class ProductDetail(View):

def get(self, request, pk, \*args, \*\*kwargs):

logger.info('Product detail page accessed')

return render(request, 'product\_detail.html')

class ProfileView(View):

def get(self, request, \*args, \*\*kwargs):

logger.info('Profile page accessed')

return render(request, 'profile.html')

class CustomerRegistrationView(View):

def get(self, request, \*args, \*\*kwargs):

logger.info('Registration page accessed')

return render(request, 'registration.html')

def post(self, request, \*args, \*\*kwargs):

logger.info('Registration form submitted')

return render(request, 'registration.html')

# Add logs to the authentication views

from django.contrib.auth.views import LoginView

class CustomLoginView(LoginView):

def get(self, request, \*args, \*\*kwargs):

logger.info('Login page accessed')

return render(request, 'login.html')

def post(self, request, \*args, \*\*kwargs):

logger.info('Login form submitted')

return render(request, 'login.html')

# Add logs to the password reset views

from django.contrib.auth.views import PasswordResetView

class CustomPasswordResetView(PasswordResetView):

def get(self, request, \*args, \*\*kwargs):

logger.info('Password reset page accessed')

return render(request, 'password\_reset.html')

def post(self, request, \*args, \*\*kwargs):

logger.info('Password reset form submitted')

return render(request, 'password\_reset.html')

# Add logs to the password change views

from django.contrib.auth.views import PasswordChangeView

class CustomPasswordChangeView(PasswordChangeView):

def get(self, request, \*args, \*\*kwargs):

logger.info('Password change page accessed')

return render(request, 'password\_change.html')

def post(self, request, \*args, \*\*kwargs):

logger.info('Password change form submitted')

return render(request, 'password\_change.html')

# Add logs to the logout view

from django.contrib.auth.views import LogoutView

class CustomLogoutView(LogoutView):

def get(self, request, \*args, \*\*kwargs):

logger.info('Logout page accessed')

return render(request, 'logout.html')

def post(self, request, \*args, \*\*kwargs):

logger.info('Logout form submitted')

return render(request, 'logout.html')

urlpatterns = [

# General pages

path('', views.home, name='home'),

path('about/', views.about, name='about'),

path('contact/', views.contact, name='contact'),

path('admin/', admin.site.urls),

# Category and Product Views

path('category/', views.CategoryView.as\_view(), name='category\_list'),

path('category/<slug:val>/', views.CategoryView.as\_view(), name='category\_detail'),

path('product-detail/<int:pk>/', views.ProductDetail.as\_view(), name='product\_detail'),

path('category-title/<str:val>/', views.CategoryTitle.as\_view(), name='category\_title'),

# Customer Profile and Address Views

path('profile/', views.ProfileView.as\_view(), name='profile'),

path('address/', views.address, name='address'),

path('updateaddress/<int:pk>/', views.UpdateAddress.as\_view(), name='updateaddress'),

# Authentication Views

path('registration/', CustomerRegistrationView.as\_view(), name='customer\_registration'),

path('accounts/login/', auth\_view.LoginView.as\_view(template\_name='app/login.html', authentication\_form=LoginForm), name='login'),

# Password Reset Views

path('password-change/', auth\_view.PasswordChangeView.as\_view(

template\_name='app/changepassword.html', form\_class=PasswordChangeForm, success\_url='/password-changed-done'), name='password\_change'),

path('password-changed-done/', auth\_view.PasswordChangeDoneView.as\_view(template\_name='app/passwordchangedone.html'), name='password\_change\_done'),

path('logout/', custom\_logout\_view, name='logout'),

path('password-reset/', auth\_view.PasswordResetView.as\_view(

template\_name='app/password\_reset.html', form\_class=MyPasswordResetForm

), name='password\_reset'),

path('password-reset-done/', auth\_view.PasswordResetDoneView.as\_view(

template\_name='app/password\_reset\_done.html'

), name='password\_reset\_done'),

path('password-reset-confirm/<uidb64>/<token>/', auth\_view.PasswordResetConfirmView.as\_view(

template\_name='app/password\_reset\_confirm.html', form\_class=MySetPasswordForm

), name='password\_reset\_confirm'),

path('password-reset-complete/', auth\_view.PasswordResetCompleteView.as\_view(

template\_name='app/password\_reset\_complete.html'

), name='password\_reset\_complete'),

path('addtocart', views.add\_to\_cart, name='addtocart'),

path('cart/', views.cartpage, name='cart'),

path('remove/<int:item\_id>/', remove\_cart\_item, name='remove\_cart\_item'),

path('checkout/', views.checkout, name='checkout'),

path('products/',views.product\_list, name='product\_list'),

path('search/', views.search, name='search'),

path('orders/', views.order\_summary, name='order-summary'),

path('checkout/',views.checkout, name='checkout'),

path("my-orders/", views.my\_orders, name="my\_orders"),

path("confirm-order/", views.confirm\_order, name="confirm\_order"),

path("admin/login/", custom\_admin\_login, name="custom\_admin\_login"),

path("my-orders/", my\_orders, name="my\_orders"),

path("cancel-order/<int:order\_id>/",cancel\_order, name="cancel\_order"),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Modules.py**

from django.db import models

from django.contrib.auth.models import User

from decimal import Decimal

from django.utils.html import mark\_safe

STATE\_CHOICES = [

('AN', 'Andaman and Nicobar Islands'),

('AP', 'Andhra Pradesh'),

('AR', 'Arunachal Pradesh'),

('AS', 'Assam'),

('BR', 'Bihar'),

('CG', 'Chhattisgarh'),

('DL', 'Delhi'),

('GA', 'Goa'),

('GJ', 'Gujarat'),

('HR', 'Haryana'),

('HP', 'Himachal Pradesh'),

('JK', 'Jammu and Kashmir'),

('JH', 'Jharkhand'),

('KA', 'Karnataka'),

('KL', 'Kerala'),

('MP', 'Madhya Pradesh'),

('MH', 'Maharashtra'),

('MN', 'Manipur'),

('ML', 'Meghalaya'),

('MZ', 'Mizoram'),

('NL', 'Nagaland'),

('OD', 'Odisha'),

('PB', 'Punjab'),

('RJ', 'Rajasthan'),

('SK', 'Sikkim'),

('TN', 'Tamil Nadu'),

('TG', 'Telangana'),

('UP', 'Uttar Pradesh'),

('UK', 'Uttarakhand'),

('WB', 'West Bengal'),

('AR', 'Andhra Pradesh'),

('BR', 'Bihar'),

('KA', 'Karnataka'),

('OD', 'Odisha'),

('KL', 'Kerala'),

('UP', 'Uttar Pradesh'),

]

CATEGORY\_CHOICES = (

('TB', 'Tablet'),

('TO', 'Tonic'),

('OI', 'Ointment'),

('SU', 'Surgicals'),

('VI', 'Vials'),

('SO', 'Soap'),

('OT', 'Others'),

)

class Product(models.Model):

title = models.CharField(max\_length=100)

selling\_price = models.FloatField()

discount\_price = models.FloatField()

quantity = models.IntegerField(default=1)

description = models.TextField()

composition = models.TextField(default='')

prodapp = models.TextField(default='') # Fixed the extra text here

category = models.CharField(choices=CATEGORY\_CHOICES, max\_length=2)

product\_image = models.ImageField(upload\_to='products/')

def \_\_str\_\_(self):

return self.title

class Customer(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE, unique=True) # Ensure one profile per user

pharmacy\_name = models.CharField(max\_length=255, default="Unknown Pharmacy")

owner\_name = models.CharField(max\_length=200, default="Not Provided")

address = models.TextField(default="Not Provided")

city = models.CharField(max\_length=100, null=True, blank=True, default="")

state = models.CharField(choices=STATE\_CHOICES, max\_length=100, default="") # Ensure STATE\_CHOICES is defined

zipcode = models.CharField(max\_length=10, null=True, blank=True, default="")

mobile = models.CharField(max\_length=15, null=True, blank=True, default="")

certificate = models.ImageField(upload\_to='certificates/', null=True, blank=True)

def certificate\_preview(self):

"""Display certificate image preview in Django Admin."""

if self.certificate and hasattr(self.certificate, 'url'):

return mark\_safe(f'<img src="{self.certificate.url}" width="100" height="100" style="border-radius: 5px;" />')

return "No Certificate Uploaded"

certificate\_preview.short\_description = "Certificate Preview" # Rename column in admin panel

class Meta:

verbose\_name = "Customer"

verbose\_name\_plural = "Customers"

def \_\_str\_\_(self):

return f"{self.pharmacy\_name} ({self.owner\_name})"

class Cart(models.Model):

user = models.ForeignKey(User,on\_delete=models.CASCADE)

product = models.ForeignKey(Product,on\_delete=models.CASCADE)

product\_qty = models.IntegerField(null=False,blank=False)

created\_at = models.DateTimeField(auto\_now\_add=True)

class Order(models.Model):

STATUS\_CHOICES = (

("Order Placed", "Order Placed"),

("Shipped", "Shipped"),

("Delivered", "Delivered"),

("Cancelled", "Cancelled")

)

user = models.ForeignKey(User, on\_delete=models.CASCADE)

total\_price = models.DecimalField(max\_digits=10, decimal\_places=2, default=Decimal("0.00"))

payment\_method = models.CharField(max\_length=20)

payment\_status = models.CharField(max\_length=20, default="Pending")

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default="Order Placed")

created\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"Order {self.id} by {self.user.username}"

class OrderItem(models.Model):

order = models.ForeignKey(Order, on\_delete=models.CASCADE, related\_name="items")

product = models.ForeignKey("Product", on\_delete=models.CASCADE)

quantity = models.PositiveIntegerField()

price = models.DecimalField(max\_digits=10, decimal\_places=2)

def \_\_str\_\_(self):

return f"{self.product.title} (x{self.quantity})"

**Forms.py**

from django import forms

from django.contrib.auth.forms import UserCreationForm, AuthenticationForm, UsernameField, PasswordResetForm, SetPasswordForm, PasswordChangeForm

from django.contrib.auth.models import User

from .models import Customer

# Login form

class LoginForm(AuthenticationForm):

username = UsernameField(widget=forms.TextInput(attrs={'autofocus': 'true', 'class': 'form-control'}))

password = forms.CharField(widget=forms.PasswordInput(attrs={'autocomplete': 'current-password', 'class': 'form-control'}))

# Registration form

class CustomerRegistrationForm(UserCreationForm):

username = forms.CharField(widget=forms.TextInput(attrs={'autofocus': 'True', 'class': 'form-control'}))

email = forms.EmailField(widget=forms.EmailInput(attrs={'class': 'form-control'}))

password1 = forms.CharField(label='Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

password2 = forms.CharField(label='Confirm Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

class Meta:

model = User

fields = ['username', 'email', 'password1', 'password2']

# Password reset form

class MyPasswordResetForm(PasswordResetForm):

email = forms.EmailField(widget=forms.EmailInput(attrs={'class': 'form-control'}))

# Set password form (after resetting password)

class MySetPasswordForm(SetPasswordForm):

new\_password1 = forms.CharField(label='New Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

new\_password2 = forms.CharField(label='Confirm New Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

# Password change form (if needed for changing password)

class MyPasswordChangeForm(PasswordChangeForm):

old\_password = forms.CharField(label='Old Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

new\_password1 = forms.CharField(label='New Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

new\_password2 = forms.CharField(label='Confirm New Password', widget=forms.PasswordInput(attrs={'class': 'form-control'}))

# Profile form for updating customer details

class CustomerProfileForm(forms.ModelForm):

class Meta:

model = Customer

fields = ['pharmacy\_name', 'owner\_name', 'address', 'city', 'state', 'zipcode', 'mobile', 'certificate']

widgets = {

'certificate': forms.ClearableFileInput(attrs={'class': 'form-control'}),

}

**Views.py**

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.views import View

from .models import \*

from .forms import CustomerRegistrationForm, CustomerProfileForm

from django.contrib import messages

from django.contrib.auth import logout

from django.contrib.auth.forms import PasswordResetForm

from django.contrib.auth.views import PasswordResetView

from django.contrib.auth.decorators import login\_required

from django.http import JsonResponse

import json

from django.db import IntegrityError

from django.contrib import messages

from .models import Product

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

from .models import Order,OrderItem

from decimal import Decimal

from django.contrib.admin.views.decorators import staff\_member\_required

from django.contrib.auth.views import LoginView

from django.contrib.auth import login,authenticate

# Home View

def home(request):

return render(request, "app/home.html")

def about(request):

return render(request, "app/about.html")

def contact(request):

return render(request, "app/contact.html")

class CustomAdminLoginView(LoginView):

template\_name = "admin/login.html"

def form\_valid(self, form):

user = form.get\_user()

if user.is\_staff: # Ensure only staff users can log in

login(self.request, user)

self.request.session['is\_admin'] = True # Set admin session flag

self.request.session.set\_expiry(0) # Expire session when the browser closes

self.request.session.modified = True # Ensure the session is updated

return redirect('/admin/')

else:

return redirect('login')

def user\_login(request):

if request.method == "POST":

username = request.POST["username"]

password = request.POST["password"]

user = authenticate(request, username=username, password=password)

if user is not None:

if user.is\_staff:

return redirect('/admin/login/') # Redirect admin users to admin login

else:

login(request, user)

request.session.set\_expiry(0) # Expire session when the browser closes

return redirect('user\_dashboard')

else:

return render(request, 'login.html', {"error": "Invalid credentials"})

return render(request, "login.html")

def custom\_logout\_view(request):

if request.user.is\_staff: # Only log out admin if they are logging out

logout(request)

return redirect('login')

# Category View

class CategoryView(View):

def get(self, request, val=None):

products = Product.objects.filter(category=val)

titles = Product.objects.filter(category=val).values\_list('title', flat=True).distinct()

context = {

"products": products,

"titles": titles,

"slug\_value": val,

}

return render(request, "app/category.html", context)

# Product Detail View

class ProductDetail(View):

def get(self, request, pk):

product = get\_object\_or\_404(Product, pk=pk)

return render(request, "app/productdetail.html", {"product": product})

# Category Title View

class CategoryTitle(View):

def get(self, request, val):

product = Product.objects.filter(title=val)

if product.exists():

titles = Product.objects.filter(category=product[0].category).values('title')

context = {

"titles": titles,

"products": product,

}

return render(request, "app/category.html", context)

else:

messages.error(request, "No products found with the specified title.")

return redirect('home')

# Customer Registration View

class CustomerRegistrationView(View):

def get(self, request):

form = CustomerRegistrationForm()

return render(request, 'app/customerregistration.html', {'form': form})

def post(self, request):

form = CustomerRegistrationForm(request.POST)

if form.is\_valid():

form.save()

messages.success(request, "Congratulations! User registered successfully.")

return redirect('home')

else:

messages.error(request, "Invalid input data. Please check the form.")

return render(request, 'app/customerregistration.html', {'form': form})

def custom\_admin\_login(request):

if request.method == "POST":

username = request.POST.get("username")

password = request.POST.get("password")

user = authenticate(request, username=username, password=password) # Authenticate user

if user is not None and user.is\_staff: # Ensure the user exists and is an admin

login(request, user) # Log in the admin user

return redirect("/admin/") # Redirect to Django admin panel

else:

messages.error(request, "Invalid credentials or not an admin") # Show error message

return render(request, "admin/login.html")

# Profile View

class ProfileView(View):

def get(self, request):

customer, created = Customer.objects.get\_or\_create(user=request.user)

form = CustomerProfileForm(instance=customer)

return render(request, 'app/profile.html', {'form': form, 'customer': customer})

def post(self, request):

customer, created = Customer.objects.get\_or\_create(user=request.user)

form = CustomerProfileForm(request.POST, request.FILES, instance=customer) # Ensure request.FILES is included

if form.is\_valid():

form.save() # Save the form correctly

messages.success(request, "Profile updated successfully.")

return redirect('profile')

return render(request, 'app/profile.html', {'form': form, 'customer': customer})

# Address View

def address(request):

# Fetch the addresses for the logged-in user

addresses = Customer.objects.filter(user=request.user)

context = {'addresses': addresses}

return render(request, 'app/address.html', context)

# Update Address View

class UpdateAddress(View):

def get(self, request, pk):

# Fetch the existing address object for the given pk

address = get\_object\_or\_404(Customer, pk=pk, user=request.user)

form = CustomerProfileForm(instance=address)

return render(request, 'app/updateaddress.html', {'form': form, 'address': address})

def post(self, request, pk):

# Fetch the existing address object for the given pk

address = get\_object\_or\_404(Customer, pk=pk, user=request.user)

form = CustomerProfileForm(request.POST, instance=address)

if form.is\_valid():

form.save()

messages.success(request, "Address updated successfully.")

return redirect('address')

else:

messages.error(request, "Error updating the address. Please correct the errors below.")

return render(request, 'app/updateaddress.html', {'form': form, 'address': address})

def custom\_logout\_view(request):

if not request.user.is\_staff: # Prevents admin users from being logged out

logout(request)

return redirect('login')

class MyPasswordResetView(PasswordResetView):

form\_class = PasswordResetForm

template\_name = 'app/password\_reset.html'

success\_url = '/password-reset-done/'

def add\_to\_cart(request):

if request.headers.get('X-Requested-With') == 'XMLHttpRequest':

if request.user.is\_authenticated:

try:

data = json.loads(request.body)

product\_qty = int(data.get('product\_qty', 0))

product\_id = int(data.get('pid', 0))

if product\_qty <= 0:

return JsonResponse({'status': 'Invalid quantity'}, status=400)

try:

product = Product.objects.get(id=product\_id)

except Product.DoesNotExist:

return JsonResponse({'status': 'Product not found'}, status=404)

if Cart.objects.filter(user=request.user, product=product).exists():

return JsonResponse({'status': 'Product already in cart'}, status=200)

if product.quantity >= product\_qty:

try:

Cart.objects.create(user=request.user, product=product, product\_qty=product\_qty)

return JsonResponse({'status': 'Product added to cart'}, status=200)

except IntegrityError as e:

print(f"IntegrityError: {e}")

return JsonResponse({'status': 'Database error'}, status=500)

else:

return JsonResponse({'status': 'Product stock not available'}, status=400)

except json.JSONDecodeError:

return JsonResponse({'status': 'Invalid JSON data'}, status=400)

except ValueError:

return JsonResponse({'status': 'Invalid data format'}, status=400)

else:

return JsonResponse({'status': 'Please log in to add items to the cart'}, status=401)

return JsonResponse({'status': 'Invalid access'}, status=403)

def cartpage(request):

if request.user.is\_authenticated:

cart = Cart.objects.filter(user=request.user)

return render(request, 'app/cart.html', {'cart': cart})

else:

return redirect('/')

def remove\_cart\_item(request, item\_id):

if request.method == "POST": # Indentation is fixed

cart\_item = get\_object\_or\_404(Cart, id=item\_id)

cart\_item.delete()

messages.success(request, "Item removed from cart successfully.")

return redirect('cart')

def checkout(request):

return render(request, 'app/addtocart.html')

def search(request):

query = request.GET.get('search', '') # Prevent KeyError

products = Product.objects.filter(title\_\_icontains=query) if query else []

return render(request, "app/search.html", {"products": products, "query": query})

def product\_list(request):

products = Product.objects.all() # Fetch all products

return render(request, "app/product\_list.html", {"products": products})

def order\_summary(request):

cart\_items = Cart.objects.filter(user=request.user)

total\_quantity = sum(item.product\_qty for item in cart\_items)

total\_amount = "{:.2f}".format(sum(item.product.discount\_price \* item.product\_qty for item in cart\_items))

context = {

"cart\_items": cart\_items,

"total\_quantity": total\_quantity,

"total\_amount": total\_amount,

}

return render(request, "app/order.html", context)

@login\_required

@csrf\_exempt

def confirm\_order(request):

if request.method == "POST":

try:

data = json.loads(request.body)

user = request.user

payment\_method = data.get("payment\_method")

total\_price = Decimal(data.get("total\_price", "0.00"))

# Create a new order

order = Order.objects.create(

user=user,

payment\_method=payment\_method,

payment\_status="Pending",

total\_price=total\_price,

status="Order Placed"

)

# Move items from cart to order

cart\_items = Cart.objects.filter(user=user)

for item in cart\_items:

OrderItem.objects.create(

order=order,

product=item.product,

quantity=item.product\_qty,

price=item.product.discount\_price if hasattr(item.product, 'discount\_price') else item.product.price

)

# Clear the user's cart after order placement

cart\_items.delete()

return JsonResponse({"message": "Order Confirmed Successfully!", "order\_id": order.id})

except Exception as e:

return JsonResponse({"error": str(e)}, status=400)

@login\_required

def my\_orders(request):

orders = Order.objects.filter(user=request.user).prefetch\_related("items\_\_product")

return render(request, "app/my\_orders.html", {"orders": orders}) # Ensure this file exists in templates

def cancel\_order(request, order\_id):

order = get\_object\_or\_404(Order, id=order\_id)

if order.status == "Cancelled":

messages.warning(request, "Order is already cancelled.")

elif order.status == "Completed":

messages.error(request, "Completed orders cannot be cancelled.")

else:

order.status = "Cancelled"

order.save()

messages.success(request, "Order cancelled successfully.")

return redirect("my\_orders")