A Project Report on

OGANI E-COMMERCE

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

COMPUTER ENGINEERING

Submitted By

Smeet Ashokbhai Patel (12399)

Guided By

Dr. Pankti Bhatt

School of Computing and Technology

Institute of Advanced Research

Gandhinagar - 382426

2024-2025

Acknowledgement

I have put effort into this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Dr. Pankti Bhatt for their guidance and constant supervision as well as for providing necessary information regarding the project. I take this opportunity to thank all my friends and colleagues who started me out on the topic and provided extremely useful review feedback and for their all-time support and help in each and every aspect of the course of my project preparation. I am grateful to my college Institute of Advanced Research, Gandhinagar, for providing me with all the required resources and a good working environment.

I would like to express my gratitude towards the Head of the Department, Dr. Brijesh Jajal, and the Director, Brig. P.C. Vyas for their kind cooperation and encouragement, which helped me in this project.

Smeet Patel (12399)

Abstract

The OGANI E-Commerce Web Application project is designed to streamline and automate online grocery shopping with a focus on backend development using the Django framework. This project facilitates seamless product browsing, user authentication, cart management, and billing functionality while ensuring secure and efficient data processing. It emphasizes modular design, clean URL routing, and integration of AJAX for real-time updates without page reloads.

The system architecture supports multiple user roles and enables dynamic interaction between the front-end and back-end components through RESTful practices. Features such as category filtering, product details retrieval, and checkout management are handled efficiently using Django views and templates. The project also includes robust database modeling through ORM and effective testing strategies, including both white-box and black-box testing methodologies.

The OGANI system not only addresses the functional requirements of an online shopping portal but also adheres to best practices in code optimization, maintainability, and scalability. The project demonstrates the practical implementation of web development principles in a real-world internship setting, fostering valuable hands-on experience in full-stack development.

Institute of Advanced Research
The University for Innovation
IAR University
Established under the Gujarat Private Universities Amendment Act 2011 and recognized under section 22 and 2(f) of UGC

Date: 09/04/2025

CERTIFICATE

This is to certify that **Mr. Smeet Ashokbhai Patel** with **IAR RegNo.12399,** Student of Bachelor of Technology – School of Computing and Technology, Semester-VIII, has satisfactorily completed their project work entitled

"OGANI E-Commerce"

In partial fulfilment of the requirement for the Bachelor of Technology degree of the Institute of Advanced Research (The University for Innovation), Gandhinagar, in the year 2024-25.

Dr. Pankti Bhatt
[Project Supervisor]

Dr. Brijesh Jajal
[HoD – SCT]

Date: 12 Apr 2025

CERTIFICATE

This is to certify that Mr. Smeet Ashokbhai Patel has successfully completed project work in the company under the esteemed guidance and technical support of Sr. Developers toward the fulfillment of his educational semester.

Technology:- Python Web Development
From – 23rd Dec 2024 to 12th Apr 2025

He has successfully completed his Internship. During the period he was sincere, hardworking & fully devoted to project.

We wish him all the best in their future endeavors.

Maxgen Technologies Pvt. Ltd.

Sincerely,
Mamta Yerawar (HR Manager)
Maxgen Technologies Pvt. Ltd.

INSTITUTE OF ADVANCED RESEARCH

[UNDERTAKING ABOUT ORIGINALITY OF WORK]

I, **Patel Smeet Ashokbhai, IAR 12399**, a final-year B.Tech student of Computer Engineering at the Institute of Advanced Research, Gandhinagar, solemnly affirm that the project report titled **"OGANI E-Commerce"** is entirely my original work and has not been submitted elsewhere for any purpose. I take full responsibility for the accuracy and authenticity of the information presented in this report.

Throughout the course of this project, I was guided by Dr. Pankti Bhatt, DCSE, Institute of Advanced Research, Gandhinagar. I assure that all sources used for this project have been duly acknowledged and referenced in the report. I affirm that this project report is free from any form of plagiarism or academic misconduct.

I also acknowledge that any assistance received in the preparation of this project report has been properly cited. I am aware that any act of plagiarism or academic misconduct will result in serious disciplinary action in accordance with the institute's rules and regulations.

Date: 15th April 2025
Place: Gandhinagar

Signature of Student: **Signature of Guide:**

Patel Smeet Ashokbhai **Dr. Pankti Bhatt**

PROJECT INDEX

PROJECT INDEX

LIST OF FIGURES

LIST OF TABLES

1. INTERNSHIP ORGANISATION ESSENTIALS

1.1 Organization Profile

Maxgen Technologies Pvt. Ltd.

Founded	:	2013
Type	:	Privately Held
Industry	:	IT Services and Consulting
Headquarters	:	Pune, Maharashtra, India
Additional Offices	:	Ahmedabad (Corporate Office), Navi Mumbai (Branch Office-1), USA (Branch Office-2)
Company Size	:	51–200 employees
Specialties	:	Web Designing, Web Development, E-Commerce, Digital Marketing, SEO Services, Logo & Graphic Designing, Game Development, and Corporate Training

1.2 Organization Background

Maxgen Technologies Pvt. Ltd. (MNC), with locations in Pune, Navi Mumbai, Ahmedabad, and the United States, is your top choice for professional WordPress development, website design, SEO, and digital marketing services. With years of experience in the IT industry, we have gained deep insights into the Indian market, setting us apart from the competition. We also offer specialized Android application development services in Ahmedabad, Navi Mumbai, and Pune, catering to diverse business needs.

WordPress, web design and development, mobile app development, E-Commerce development, logo and graphic design, web support and maintenance, etc., are all services we provide. Every company needs a website that will serve as a salesperson. A well-designed website pulls in new customers and specialty visitors. Your website should draw visitors if you want to help your business grow. Use our specialized web development services to create a stunning website that attracts potential customers.

Additionally, we provide internship programs and services. Internship programs help you develop your skills while working with us. You'll get the chance to work on various projects and gain invaluable experience that will undoubtedly benefit your future career.

We are a full-service web design and development company that provides services for creating websites, logos, graphics, and software. Designing E-Commerce websites and boosting conversion rates for online merchants are areas in which we specialize. We also offer hosting, domain administration, and support services in India.

Our ability to combine a quality-driven delivery process with our in-depth knowledge of current market trends, cutting-edge technology, and extensive business domain expertise gives us an edge. When we say that we will build your website and brand for success, we mean it. We are a group of web development and design professionals who value business

growth and strive to increase visitor numbers through various digital marketing channels while providing a user-friendly experience.

1.3 Organization Activities

Maxgen Technologies Pvt. Ltd. engages in a wide spectrum of IT-related activities that cater to both corporate clients and aspiring IT professionals. Their operations are broadly divided into two key areas: IT Services and Industrial Training & Internships.

A. IT Services and Solutions

1. Custom Software Development

Maxgen builds scalable, secure, and robust software solutions tailored to business needs, leveraging technologies like Java, Python, PHP, .NET, and more.

2. Web Development

They design and develop responsive, feature-rich websites and web applications using modern frameworks and CMS platforms.

3. Mobile App Development

Maxgen develops mobile applications for Android and iOS platforms, focusing on performance, UI/UX, and integration.

4. Digital Marketing

Their digital marketing team offers services like SEO, SEM, social media marketing, email campaigns, and brand visibility strategies.

5. UI/UX Design

The company crafts intuitive and engaging user interfaces and experiences for web and mobile applications.

6. Software Testing

They provide manual and automated testing services to ensure the quality, reliability, and performance of software applications.

B. Industrial Training and Internships

1. Technology Training Programs

Maxgen offers hands-on training in cutting-edge technologies such as:

- Python

- Artificial Intelligence & Machine Learning

- Data Science
- Java / Core Java / Advanced Java
- PHP / Laravel
- Android Development
- .NET Technologies
- Manual & Automation Testing (Selenium, QTP)

2. Live Project Exposure

Interns and trainees work on **live client-based projects** under the guidance of experienced mentors, helping them understand the real-world software development cycle.

3. Career Development Support

The company provides resume building, interview preparation, and placement assistance for students and freshers.

4. Workshops and Seminars

Maxgen organizes sessions to keep students updated with emerging tech trends and industry practices.

1.4 Organization Services

E-commerce Development

We assist you in creating a flawless E-Commerce experience for your clients in your stores using our knowledge in E-Commerce platforms like Magento and WooCommerce.

Flutter App Development

We assist you with adopting Flutter to transfer your Android and iOS app development. Your current application may be ported across platforms and operating systems thanks to the expertise of our team of Flutter Android app development and deployment professionals. We are the platform-neutral app development firm for Flutter that the world relies on.

Custom Application Development

We can do it quickly & cost-effectively, whether you need to construct a bespoke web app from the start, move your old backend, or simplify current front-end functionality.

We provide document management solutions, secure intranet, extranet, portal setup, application conversion to the cloud, collaboration and portal creation, corporate intelligence solutions, and maintenance, support, and upgrade services.

Web Application Support and Maintenance

We collaborate with you to manage web apps created by outside suppliers or by us. Our skilled engineers can easily handle any task, whether repairing problems or introducing new capabilities. We also track all of your future requirements and offer assistance.

We will create and deploy web apps from our team of committed web application developers to help you increase operational effectiveness, hasten decision-making, and acquire a competitive edge. We constantly prioritize the scalability and security of the application and modify Micro Services to follow the security standards.

We have created a benchmark in the web app development industry by utilizing the most recent web technologies to meet various project needs. Additionally, we devote ourselves entirely to the project's technical and commercial aspects. We often offer the finest inside the expected period by keeping communication open and adhering to the Agile strategy.

2. ABOUT THE SYSTEM

2.1 Introduction

The OGANI E-commerce System is a comprehensive, backend-driven web application developed as part of a real-world internship project at Maxgen Technologies Pvt. Ltd. It is designed to simulate the core functionalities of an online shopping platform, focusing exclusively on the backend logic and business operations required to manage an online store efficiently.

In today's digital landscape, e-commerce platforms are at the heart of the global economy, offering consumers a seamless way to browse, select, and purchase products from the comfort of their homes. However, the backbone of any e-commerce system lies not in its visual interface but in the strength and efficiency of its backend processes. OGANI serves as a robust example of this concept by prioritizing functionality, data flow, and logical architecture over front-end aesthetics.

The primary objective of this project is to build a backend system that supports dynamic product listing, category-based filtering, cart operations, checkout mechanisms, and order processing. The platform allows users to manage shopping cart items and proceed through a simplified checkout workflow—all backed by a secure and logically structured Django framework.

The project emphasizes database interaction using Django ORM (Object-Relational Mapping), secure and session-based user handling, AJAX-based dynamic functionality to reduce page reloads, and cart manipulation features like item addition, removal, quantity updates, and total price calculation. Each function and view is designed to interact with corresponding models that reflect real-time changes in the system database.

Through this system, special attention is given to:

- Efficient handling of user sessions and cart data
- Accurate representation of product details and categories
- Seamless filtering and searching of items
- Automated calculation of total price and bill generation
- Modular architecture and maintainable code structure

The project also simulates the kind of work done in professional settings by incorporating coding practices that are scalable, maintainable, and reusable. The use of Django, Python, and structured project modules ensures that the backend is capable of supporting a real-time production application if needed.

Ultimately, OGANI reflects not just a training exercise but a practical implementation of modern backend development methodologies aimed at solving real-world business problems by enabling efficient e-commerce operations.

2.2. Tools and Technology Used for the System

The development of the OGANI e-commerce system is supported by a suite of modern and industry-standard technologies. Each tool has been carefully selected to ensure optimal performance, maintainability, scalability, and integration with real-world development environments. The system's architecture is based on the Model-View-Template (MVT) design pattern provided by Django, a high-level Python web framework known for its simplicity and efficiency.

1. Programming Language

- **Python**: Used for backend development, Python provides a clean syntax and robust support for web development through frameworks like Django. It ensures quick development cycles and supports integration with databases and other libraries.

2. Web Framework

- **Django (Version 5.x)**: A powerful web framework that follows the MVT pattern, Django facilitates rapid development of secure and maintainable web applications. It handles URL routing, data modeling, and rendering logic efficiently.

3. Database

- **SQLite (Development)**: Used during the development phase for simplicity and ease of configuration.
- **MySQL/PostgreSQL (Optional for Production)**: Scalable relational databases that can be adopted in production for handling larger datasets.

4. Development Tools

- **Visual Studio Code (VS Code)**: Primary code editor used for development, offering extensive plugin support for Django and Python.
- **Git**: A Version control system used for tracking changes in code and collaborating efficiently.
- **Postman**: Used for testing API endpoints and simulating data interactions.

5. Web Server (Development)

- **Django's Built-in Development Server**: Used for local testing and development. It handles HTTP requests and serves the application during the development phase.

6. Browser Tools

- **Chrome Developer Tools**: Used for debugging AJAX requests, DOM inspection, and testing the responsive behavior of the web pages.

These tools collectively contribute to the stability, security, and performance of the OGANI system. By using Django and Python, the system benefits from rapid prototyping, secure authentication, and robust database interaction, aligning with the professional standards followed in enterprise-level web development.

2.3. Requirements

The primary objective of the OGANI e-commerce backend system is to design and implement a scalable and secure platform that efficiently handles the core functionalities of an online shopping environment. This includes managing products, departments, user interactions, cart and checkout mechanisms, billing operations, and administrative control—all from a backend logic perspective.

This section elaborates on the major functional requirements that were provided for the development of the system.

1. User Management

- **Registration and Login:** The system facilitates user registration with validation mechanisms. Secure login is implemented using Django's authentication system.
- **Session Management:** User sessions are maintained to personalize the cart and order experience. Sessions are terminated upon logout or inactivity.

2. Product and Department Management

- **Product Catalog:** The system manages a wide catalog of products, each associated with specific departments/categories. Every product contains attributes such as name, image, price, quantity, size, color, and description.
- **Department Classification:** Products are classified under departments (e.g., Fruits, Vegetables, Beverages) to facilitate organized browsing and filtering.

3. Shopping Cart Functionality

- **Add to Cart:** Users can add products to their cart with a single click. Each item's quantity can be managed dynamically.
- **Cart Operations:** Functions such as incrementing/decrementing quantities, updating cart totals, and removing items are performed asynchronously using AJAX.
- **Cart Summary:** The total price is calculated in real-time and displayed, reflecting any changes to item quantities or removals.

4. Checkout and Billing

- **Checkout Process:** A secure checkout system enables users to proceed to payment after verifying the items in their cart.
- **Billing System:** A billing module generates a detailed summary of the items purchased, their quantities, individual prices, total cost, and order ID for record-keeping.

5. Product Filtering and Search (AJAX-Based)

- **Search Bar:** Implemented to search products by name or keyword.
- Category Filter: Users can filter products based on departments, price range, sizes, or colors.

- AJAX Requests: Filtering is handled without reloading the page, improving performance and enhancing user experience.

6. Wishlist Functionality

- Users can mark certain products as favorites to add them to a wishlist for future reference or purchase.

7. Admin and Backend Logic

- Although the project is centered on backend logic, administrative functionalities such as product management, order monitoring, and billing logs are handled within the Django admin interface.
- Database Modeling: Entities like User_Detail, Product, Department, Cart, and BillingDetail are designed with proper relationships (foreign keys, one-to-many) to ensure data integrity and normalization.

8. Error Handling and Validations

- The system includes proper validation at all data entry points to prevent invalid data from being processed.
- Exception handling mechanisms are in place to manage errors such as missing product IDs or cart access without login.

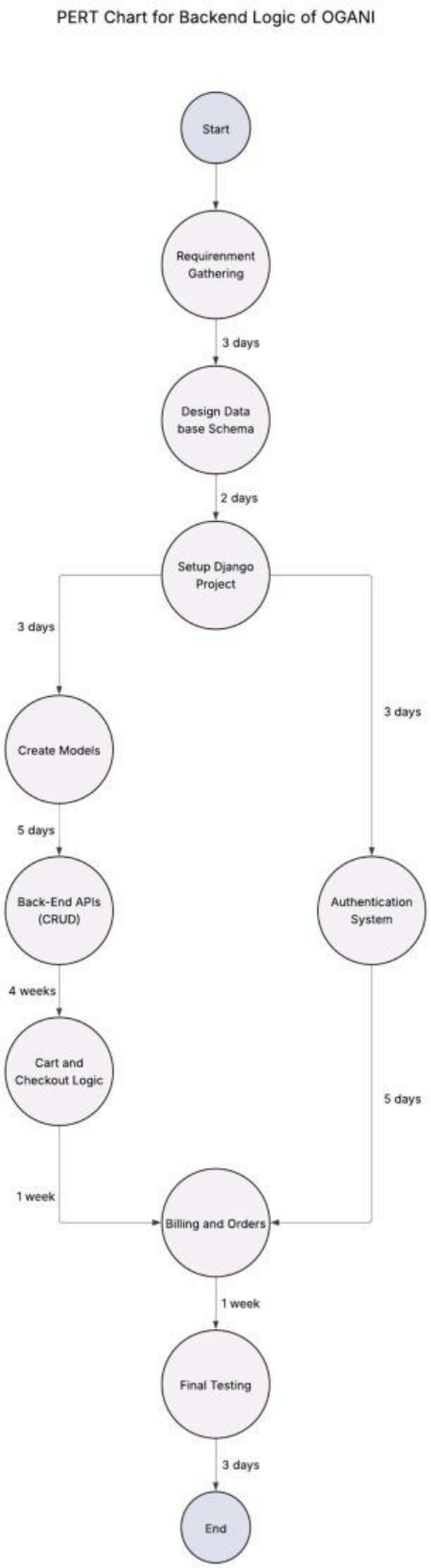
9. Security Measures

- Django's built-in protections against CSRF, SQL injection, and session hijacking are utilized to safeguard user data and transactions.

10. Modular and Maintainable Architecture

- The application is structured using reusable views, templates, and models to ensure long-term maintainability and scalability. The separation of logic into views and models enhances code readability and debugging efficiency.

2.4. Project planning using PERT Analysis



[Figure 2.4.1 PERT Chart]

1. Start: The project initiates from this node, marking the beginning of the backend development process.

2. Requirement Gathering (3 days): The first step is to collect detailed requirements regarding the backend functionality. This includes features like product management, user handling, authentication, cart logic, and billing workflow. It sets the groundwork for designing an efficient backend structure.

3. Design Database Schema (2 days): Based on the gathered requirements, the next step is to design the database schema. This task includes defining tables, attributes, primary-foreign key relationships, and normalization. The design must support operations such as CRUD (create, read, update, delete), transactions, and relationships between users, products, carts, and orders.

4. Setup Django Project (2 days): A foundational setup of the Django framework is done at this stage. This includes creating the Django project, defining installed apps, initial migrations, setting up the admin panel, and basic configurations. It acts as the base for model and logic implementation.

5. Create Models (3 days): Django models are created for entities like User, Product, Cart, Order, BillingDetail, etc. These models form the ORM layer and directly reflect the database design. Relationships using ForeignKey, OneToMany, and ManyToMany are implemented.

6. Back-End APIs (CRUD) (5 days): RESTful API endpoints are developed for each model. These APIs handle CRUD operations such as adding products, registering users, updating cart contents, and processing orders. Django REST Framework or Django's native views are utilized.

7. Cart and Checkout Logic (4 weeks): The core business logic for the cart and checkout process is implemented. It handles real-time cart updates, session management, quantity calculations, and transition to checkout. Secure billing workflows and order processing are included here.

8. Authentication System (3 days): User authentication and session management are implemented. This includes login, registration, password reset, token-based access (if needed), and user role management. It runs parallel to other development but is dependent on project setup.

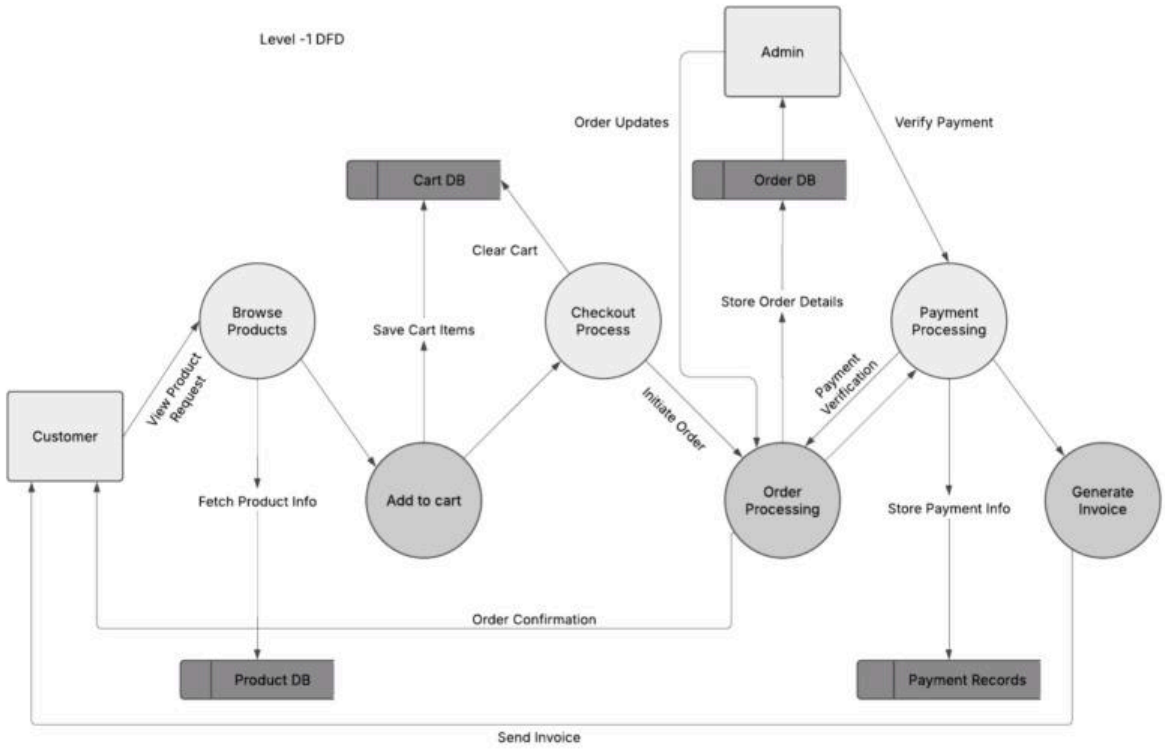
9. Billing and Orders (5 days): Integration of billing logic with cart and checkout processes. Handles invoice generation, payment method capturing (test/dummy logic for backend), and order status tracking. Ensures data integrity for every order placed by a customer.

10. Final Testing (1 week): Full backend testing using unit tests, integration tests, and manual validation. Ensures APIs work as intended, business logic is accurate, and data handling is secure. Prepares the backend system for deployment and integration with the frontend.

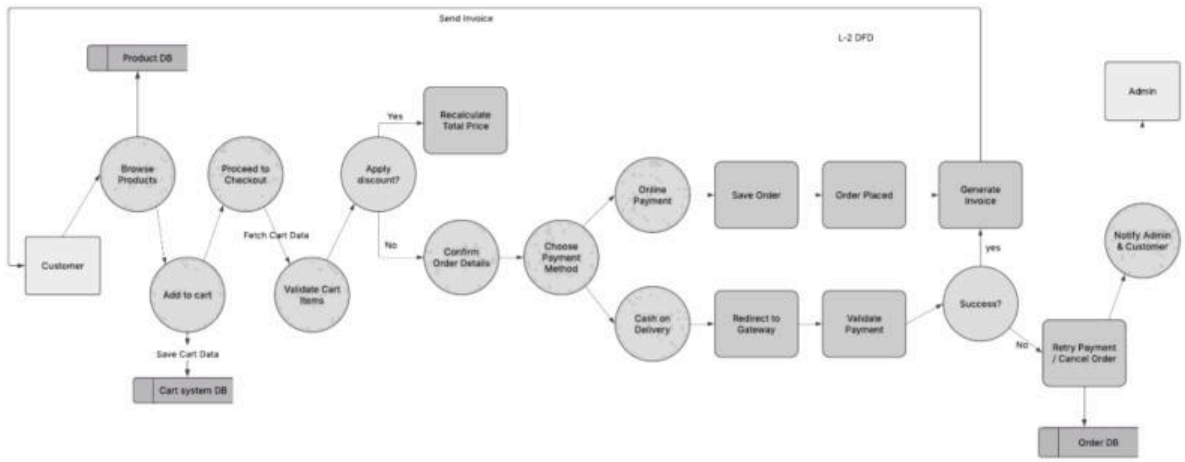
11. End (3 days buffer): A wrap-up stage providing buffer time for any leftover testing, debugging, or adjustments before final submission or deployment. Marks the formal closure of backend development.

3. SYSTEM DESIGNES USING UML

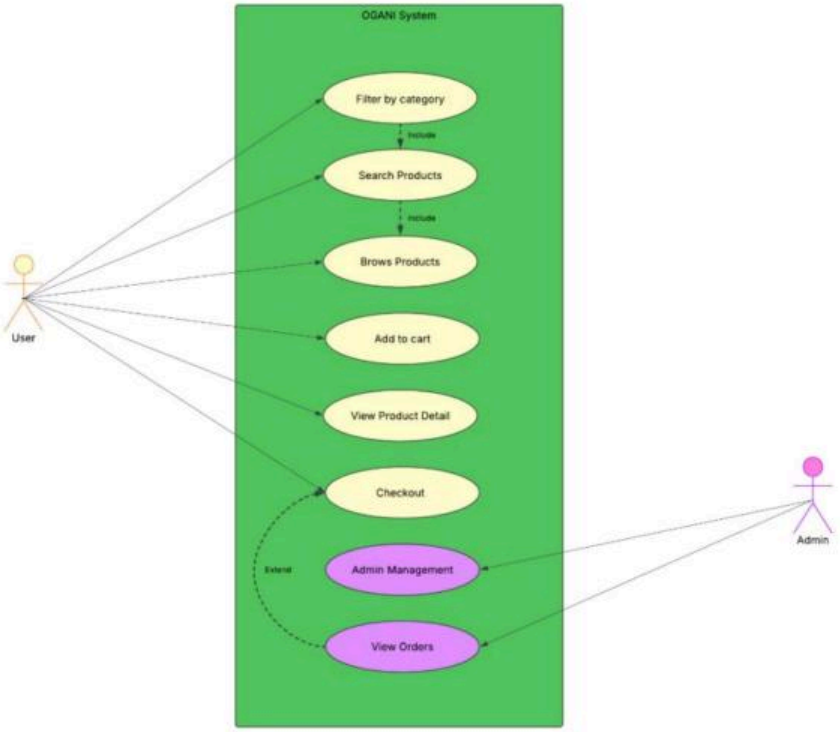
3.1.1. DFD Level-1



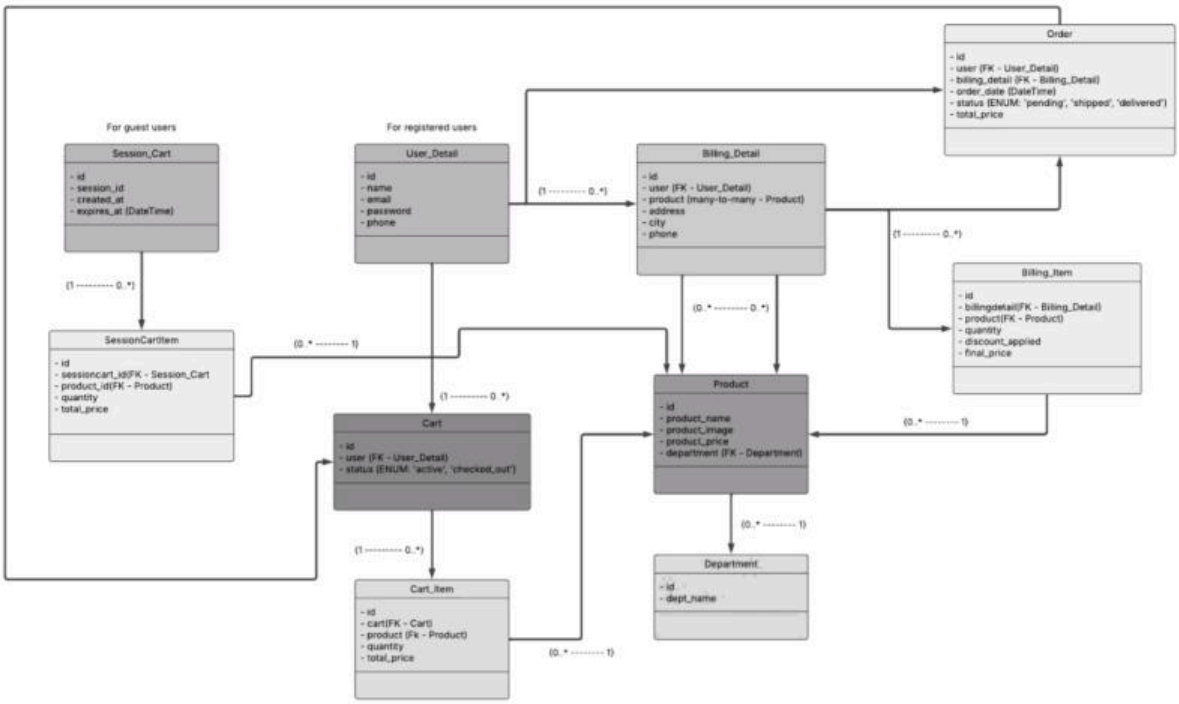
3.1.2. DFD Level-2



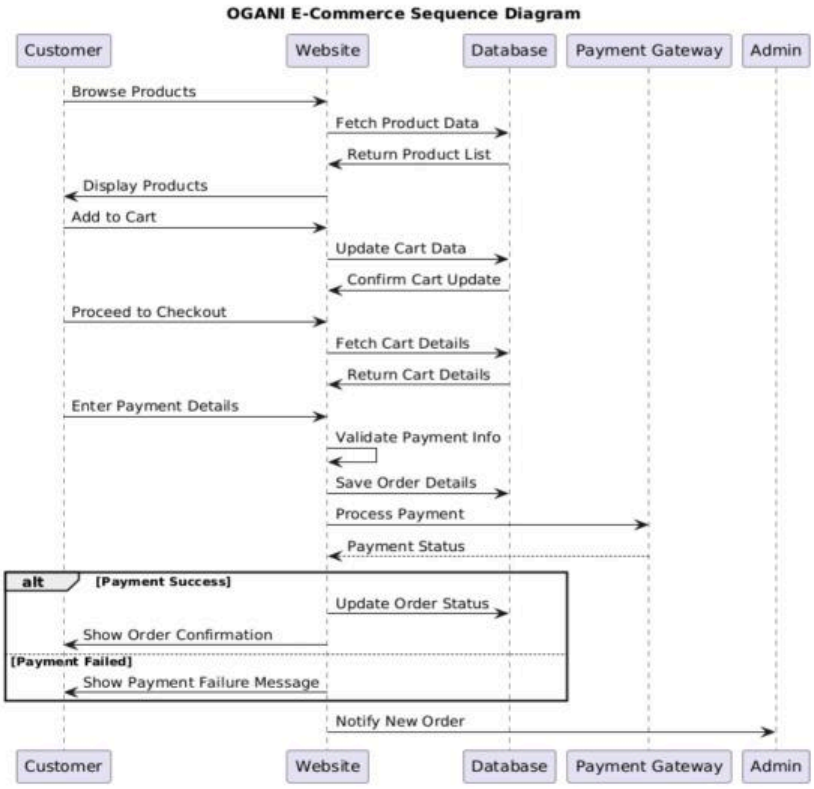
3.2. Use Case Diagram



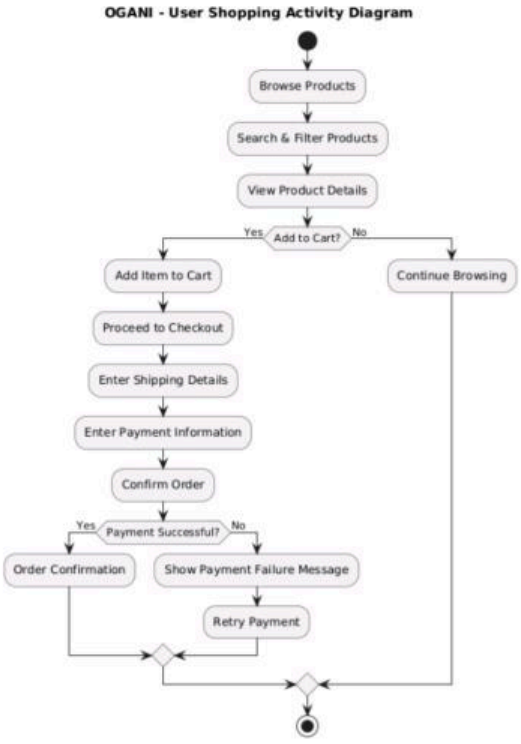
3.3. Class Diagram

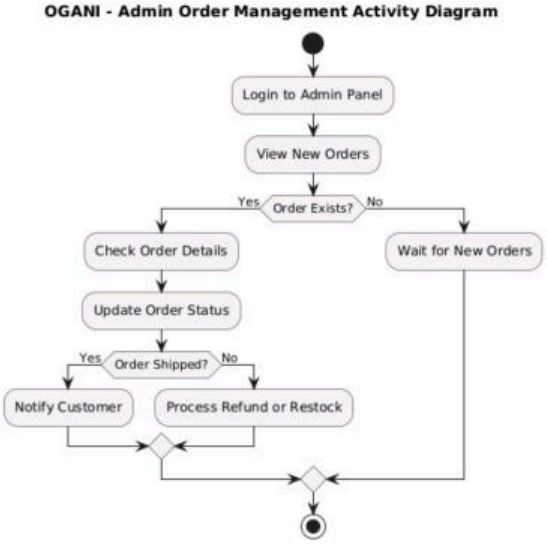


3.4. Sequence Diagram

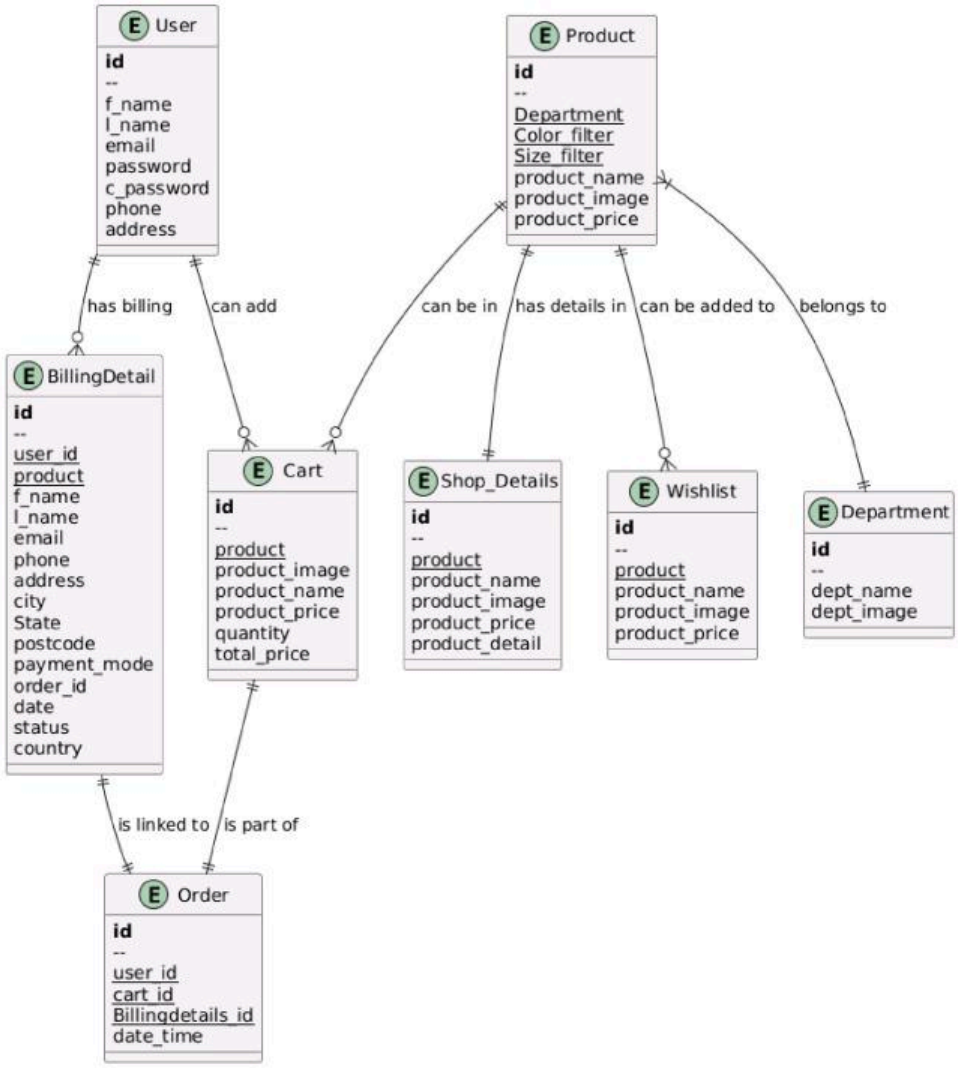


3.5. Activity Diagrams





3.6. Entity Relationship Diagram (ERD)



3.7. Data Dictionaries

3.7.1. User

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique identifier for each user
f_name	Varchar(50)	Not Null	First name of the user
l_name	Varchar(50)	Not Null	Last name of the user
email	Varchar(100)	Unique, Not Null	User email address
password	Varchar(100)	Not Null	User login password
c_password	Varchar(100)	Not Null	Password confirmation
phone	Varchar(15)	Not Null	Contact number
address	Text	Nullable	Residential/shipping address

3.7.2. Product

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique identifier for each product
Department	ForeignKey	Linked to Department(id)	Product's department
Color_filter	Varchar(50)	Nullable	Color filtering option
Size_filter	Varchar(50)	Nullable	Size filtering option
product_name	Varchar(100)	Not Null	Name of the product
product_image	Image	Nullable	Product image
product_price	Decimal(10, 2)	Not Null	Price of the product

3.7.3. Department

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique ID for each department
dept_name	Varchar(100)	Not Null	Name of the department/category
dept_image	Image	Nullable	Image representing the department

3.7.4. Cart

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique cart ID
product	ForeignKey	Linked to Product(id)	Product in the cart
product_image	Image	Nullable	Image of the product
product_name	Varchar(100)	Not Null	Product name
product_price	Decimal(10,2)	Not Null	Price per unit
quantity	Integer	Default = 1	Quantity added
total_price	Decimal(10,2)	Computed (product × quantity)	Total price for this product

3.7.5. Wishlist

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique wishlist ID
product	ForeignKey	Linked to Product(id)	Product added to wishlist
product_name	Varchar(100)	Not Null	Name of the product
product_image	Image	Nullable	Image of the product
product_price	Decimal(10,2)	Not Null	Price of the product

3.7.6. BillingDetail

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique billing ID
user_id	ForeignKey	Linked to User(id)	Billing belongs to this user
product	ForeignKey	Linked to Product(id)	Billed product
f_name	Varchar(50)	Not Null	First name of the recipient
l_name	Varchar(50)	Not Null	Last name of the recipient
email	Varchar(100)	Not Null	Email of the customer
phone	Varchar(15)	Not Null	Customer phone number
address	Text	Not Null	Shipping/billing address
city	Varchar(50)	Not Null	City
state	Varchar(50)	Not Null	State
postcode	Varchar(10)	Not Null	ZIP/postal code
country	Varchar(50)	Not Null	Country
payment_mode	Varchar(50)	Not Null	Payment method (e.g., COD, UPI)
order_id	Integer	ForeignKey(Order)	Related order
date	DateTime	Auto Now Add	Date of billing
status	Varchar(50)	Default = "Pending"	Billing status (Pending, Paid, Cancelled)

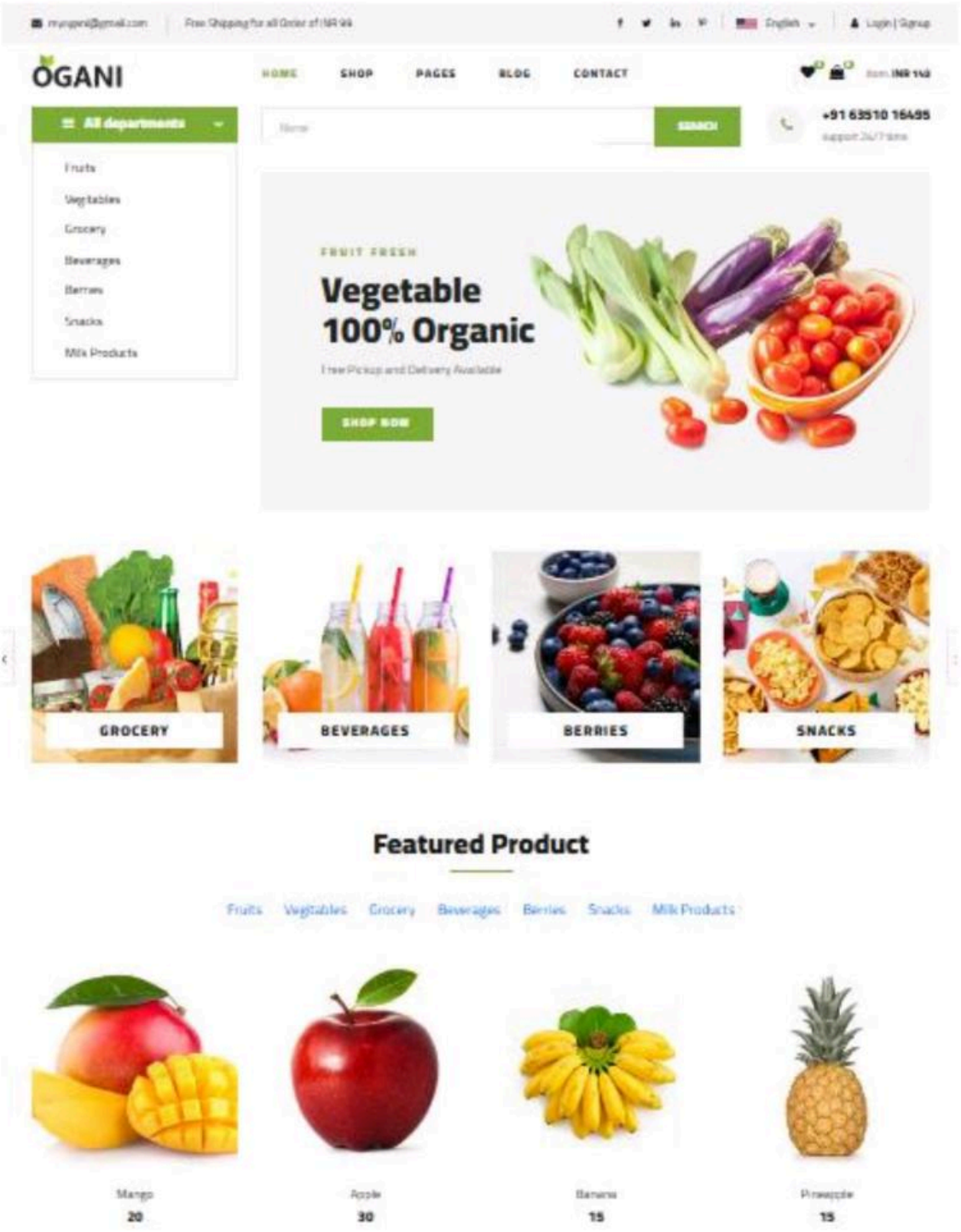
3.7.7. Order

Field Name	Data Type	Constraints	Description
id	Integer	Primary Key, Auto Increment	Unique order ID
user_id	ForeignKey	Linked to User(id)	The user who placed the order
cart_id	ForeignKey	Linked to Cart(id)	Cart items included in the order

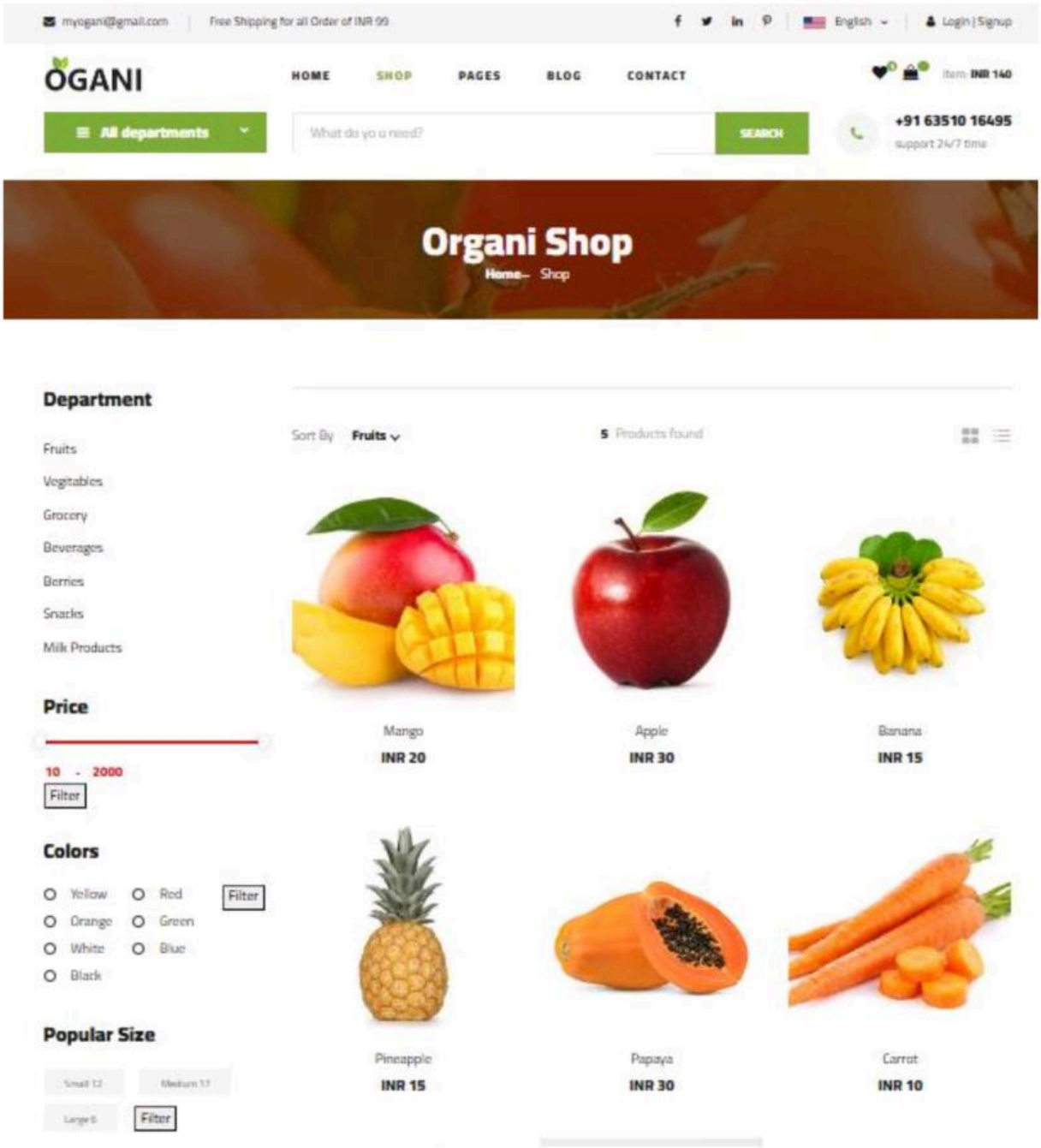
Field Name	Data Type	Constraints	Description
Billingdetails_id	ForeignKey	Linked to BillingDetail(id)	Billing information for the order
date_time	DateTime	Auto Now Add	Date and time the order was placed

4. IMPLEMENTATION

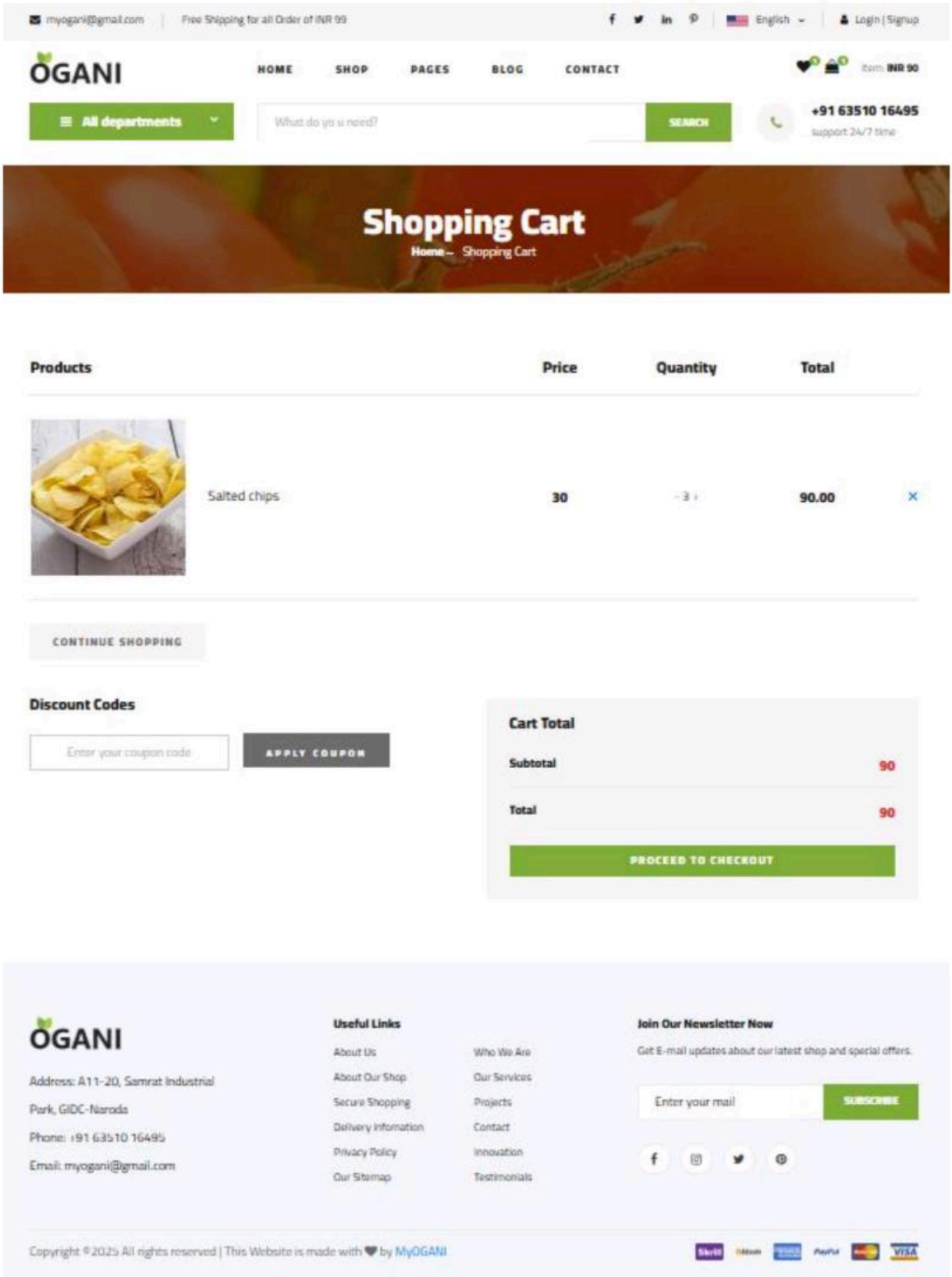
4.1 Screen Layouts and Validations



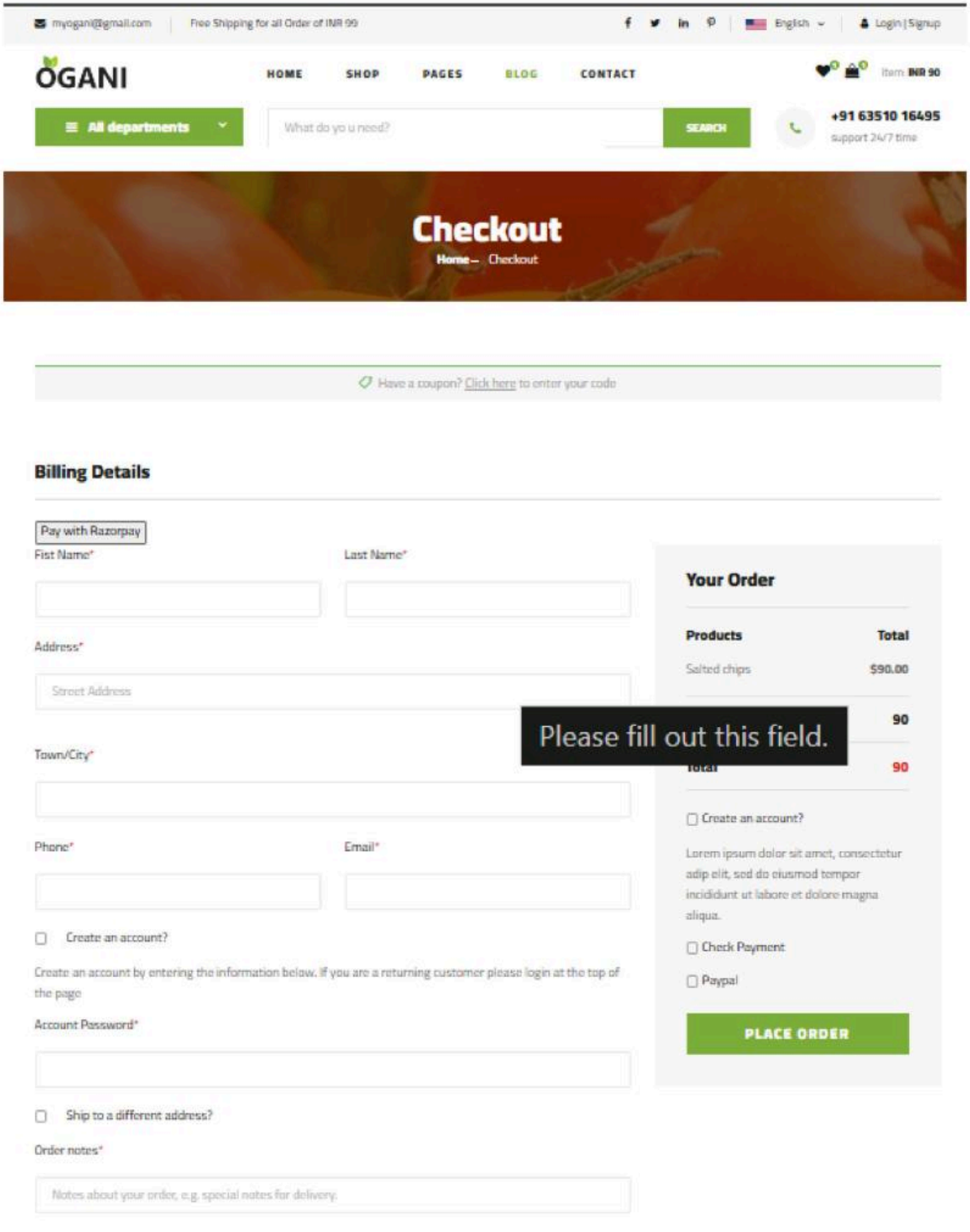
[Figure 4.1.1 Home Page]



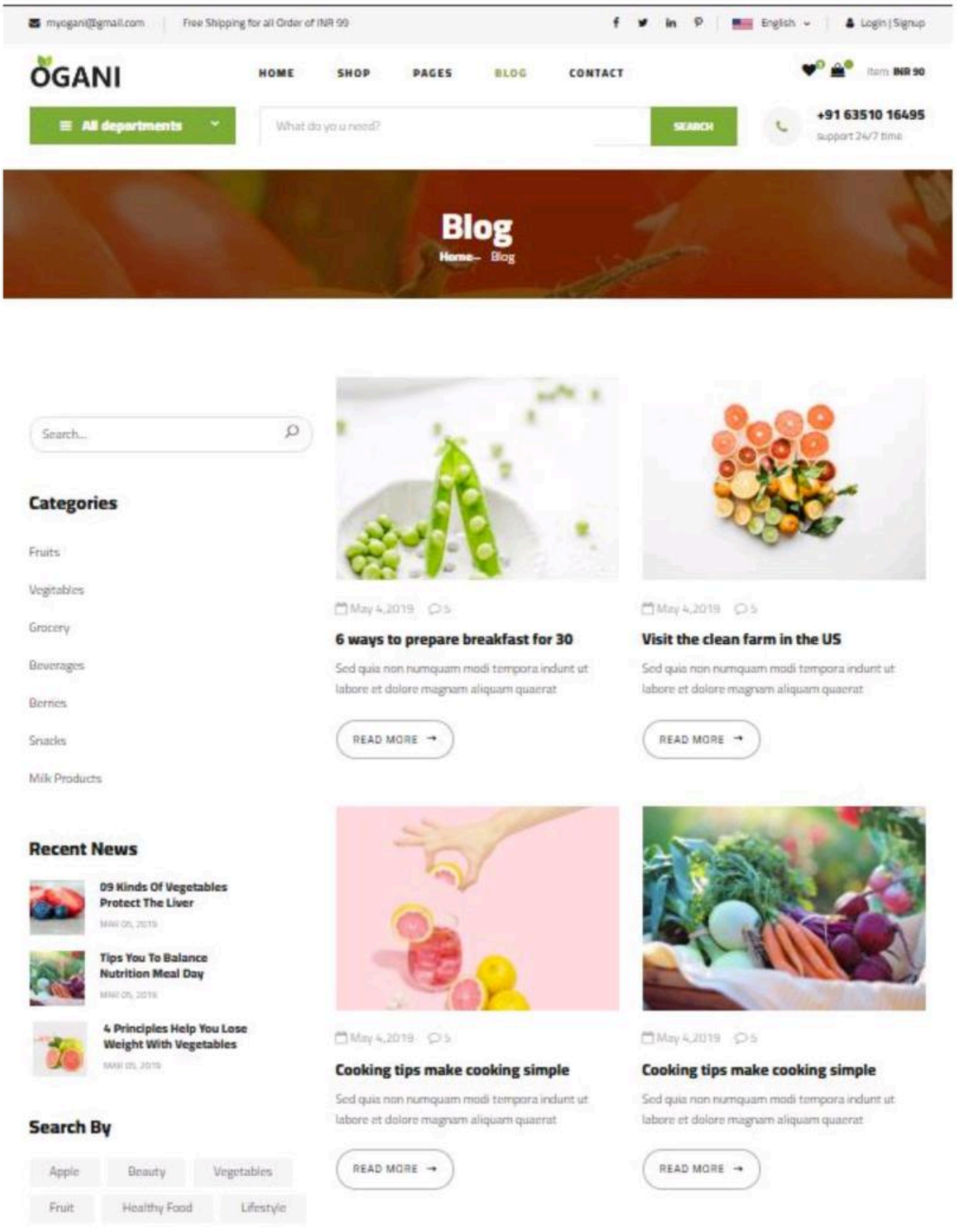
[Figure 4.1.2 Shop Page]



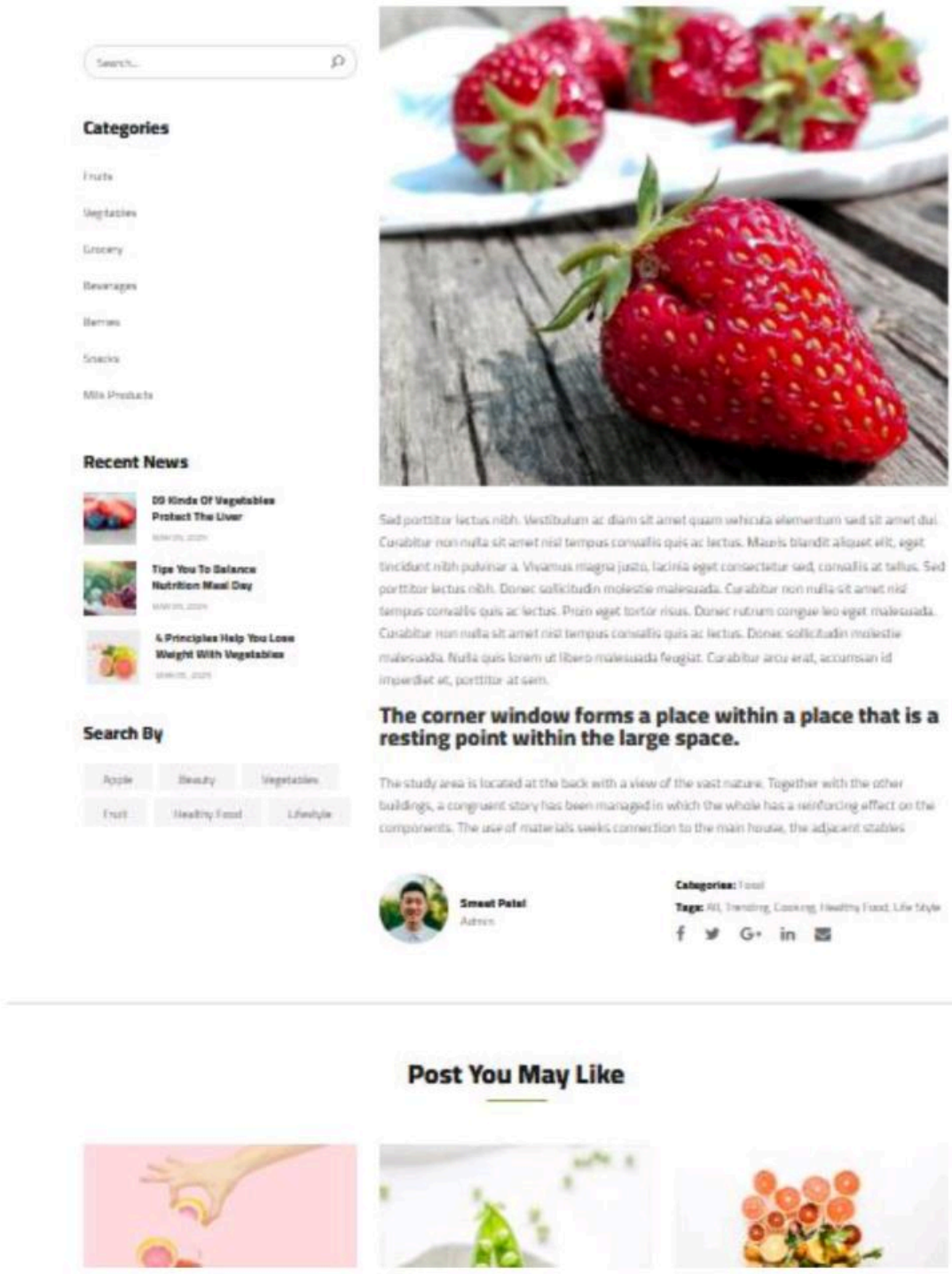
[Figure 4.1.3 Shopping Cart]



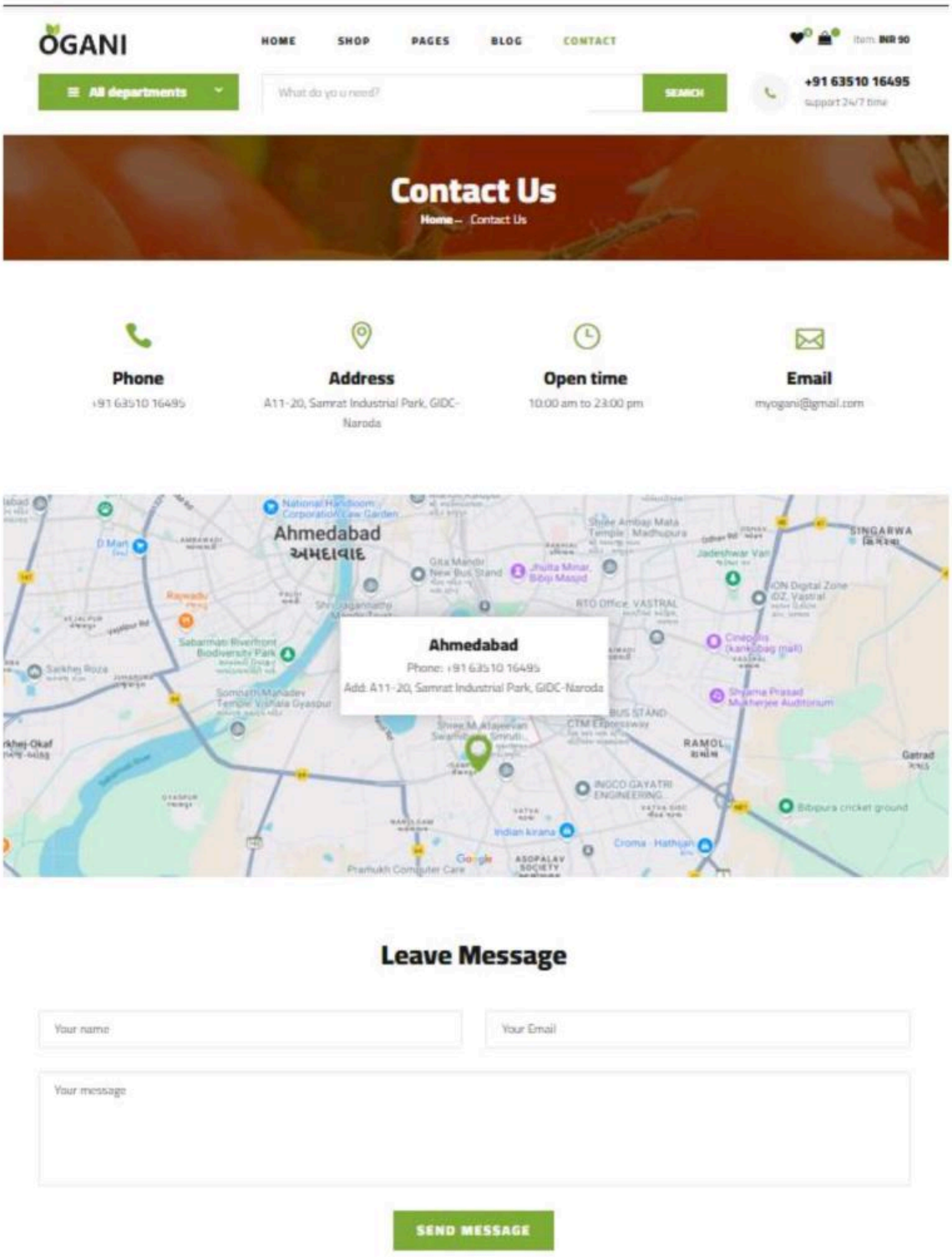
[Figure 4.1.4 Checkout Page]



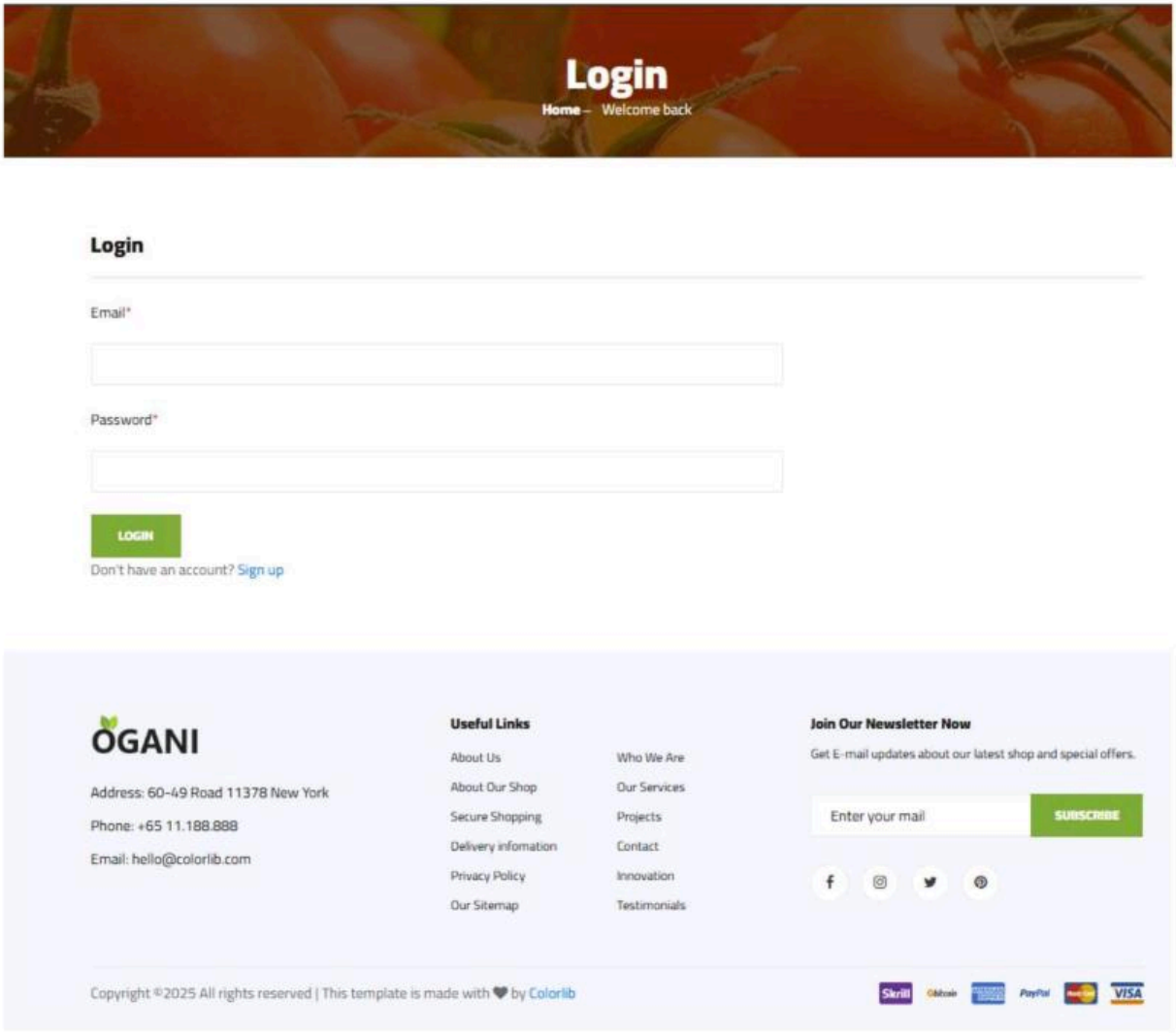
[Figure 4.1.5 Blog Page]



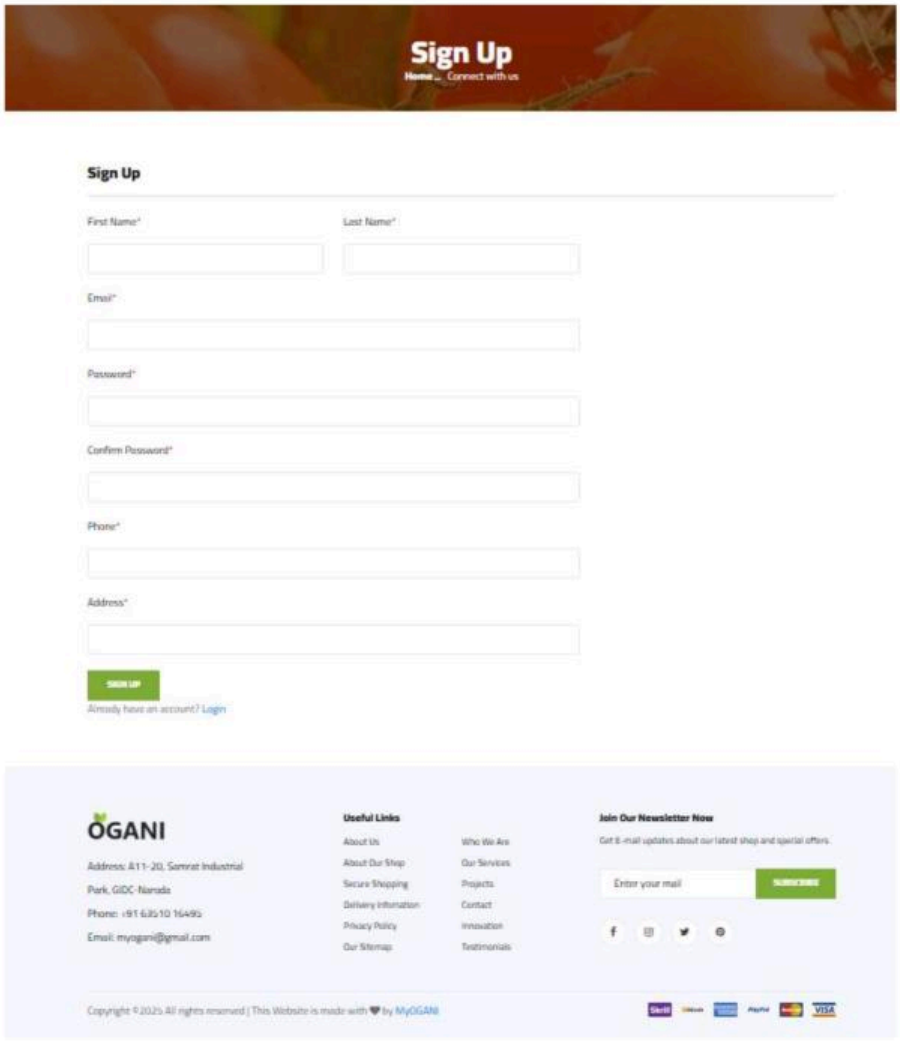
[Figure 4.1.6 Blog Details Page]



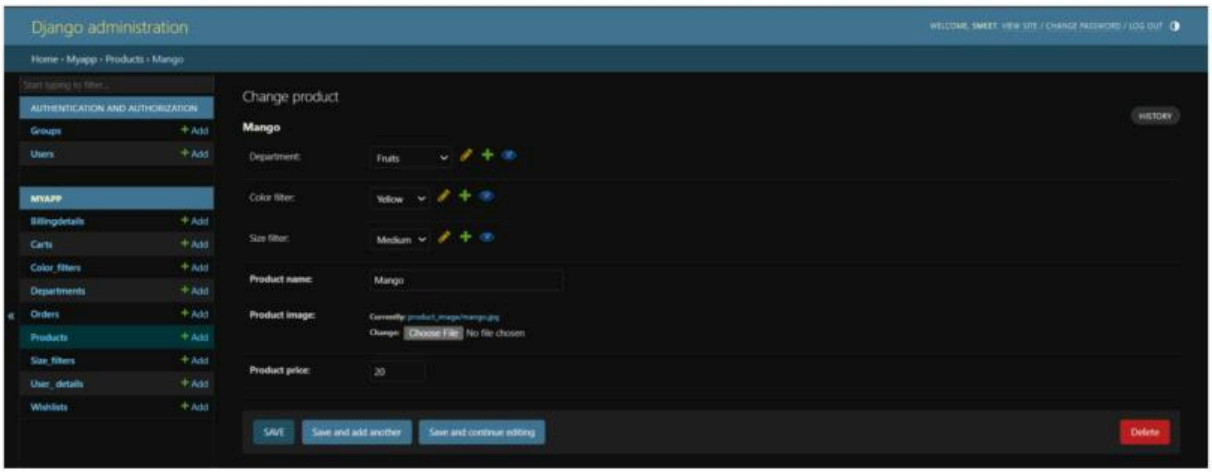
[Figure 4.1.7 Contact Page]



[Figure 4.1.8 Login Page]



[Figure 4.1.9 Sign-Up Page]



[Figure 4.1.10 Admin Panel]

4.2 Sample Coding

4.2.1. Models.py :

```
from django.db import models
from .models import*
from django.db.models import Sum


class Department(models.Model):

    dept_name = models.CharField(max_length=100)
    dept_image = models.ImageField(upload_to='dept_image')

    def __str__(self):
        return self.dept_name

class Color_filter(models.Model):

    p_color = models.CharField(max_length=20)

    def __str__(self):
        return self.p_color

class Size_filter(models.Model):

    p_size = models.CharField(max_length=20)

    def __str__(self):
        return self.p_size

class User_Detail(models.Model):

    f_name = models.CharField(max_length=50)
    l_name = models.CharField(max_length=50)
    email = models.EmailField(max_length=254)
    password = models.CharField(max_length=8)
    c_password = models.CharField(max_length=8)
    phone = models.IntegerField()
    address = models.CharField(max_length=100)

    def __str__(self):
        return self.f_name


class Product(models.Model):

    Department = models.ForeignKey(Department, on_delete=models.CASCADE, blank=True, null=True)
    Color_filter = models.ForeignKey(Color_filter, on_delete=models.CASCADE, blank=True, null=True)
    Size_filter = models.ForeignKey(Size_filter, on_delete=models.CASCADE, blank=True, null=True)
    product_name = models.CharField(max_length=100)
    product_image = models.ImageField(upload_to='Product Images')
    product_price = models.IntegerField()

    def __str__(self):
        return self.product_name

class Cart(models.Model):

    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    product_image = models.ImageField(upload_to="cart_images/")
    product_name = models.CharField(max_length=255)
    product_price = models.DecimalField(max_digits=10, decimal_places=2)
    quantity = models.PositiveIntegerField(default=1)
    total_price = models.DecimalField(max_digits=10, decimal_places=2, editable=False)

    def save(self, *args, **kwargs):
        self.total_price = self.quantity * self.product_price
        super().save(*args, **kwargs)

    def __str__(self):
        return self.product_name

    def get_cart_total():
        total = Cart.objects.aggregate(Sum('total_price'))['total_price__sum']
        return total
```

```
class Wishlist(models.Model):

    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    product_image = models.ImageField(upload_to="cart_images/")
    product_name = models.CharField(max_length=255)
    product_price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.product_name


class Billingdetail(models.Model):

    user_id=models.ForeignKey(User_Detail,on_delete=models.CASCADE,blank=True,null=True)
    product = models.ForeignKey(Product,on_delete=models.CASCADE,blank=True,null=True)
    f_name=models.CharField(max_length=100,blank=True,null=True)
    l_name=models.CharField(max_length=100,blank=True,null=True)
    email=models.EmailField()
    phone=models.IntegerField(blank=True,null=True)
    address=models.CharField(max_length=250,blank=True,null=True)
    city=models.CharField(max_length=50)
    state=models.CharField(max_length=50,blank=True,null=True)
    postcode=models.IntegerField(blank=True,null=True)
    payment_mode=models.CharField(max_length=50,blank=True,null=True)
    order_id=models.CharField(max_length=50,blank=True,null=True)
    date=models.DateTimeField(auto_now_add=True)
    status=models.CharField(max_length=50,blank=True,null=True)
    country=models.CharField(max_length=50,blank=True,null=True)

    def __str__(self):
        return self.f_name

class Order(models.Model):
    user_id=models.ForeignKey(User_Detail,on_delete=models.CASCADE,blank=True,null=True)
    cart_id=models.ForeignKey(Cart,on_delete=models.CASCADE,blank=True,null=True)
    Billingdetails_id=models.ForeignKey(Billingdetail,on_delete=models.CASCADE,blank=True,null=True)
    date_time=models.DateTimeField(auto_now_add=True,blank=True,null=True)
```

4.2.2. Views.py :

```
from django.shortcuts import render, redirect, get_object_or_404
from django.db.models import Count
from .models import*


def blog_details(request):
    dept_id = Department.objects.all()
    print("Department Data:", dept_id)
    total_cart_price = Cart.get_cart_total()

    context = {
        "dept_id" : dept_id,
        "total_cart_price": total_cart_price
        }
    return render(request, 'blog_details.html', context)

# _____________________________________________________Blog
page_____________________________________________

def blog(request):
    dept_id = Department.objects.all()
    print("Department Data:", dept_id)
    total_cart_price = Cart.get_cart_total()

    context = {
        "dept_id" : dept_id,
        "total_cart_price": total_cart_price
        }
    return render(request, 'blog.html', context)

# _____________________________________________________Check-Out
page_____________________________________________

def checkout(request):

    if request.method=='POST':
        f_name = request.POST.get('f_name')
        l_name = request.POST.get('l_name')
        email = request.POST.get('email')
        phone = request.POST.get('phone')
        address = request.POST.get('address')
```

```
        city = request.POST.get('city')

        billing_detail = Billingdetail(
            f_name=f_name,
            l_name=l_name,
            email=email,
            phone=phone,
            address=address,
            city=city
        )
        billing_detail.save()

        return redirect('checkout')

    dept_id = Department.objects.all()
    print("Department Data:", dept_id)

    cart_items = Cart.objects.all()
    total_cart_price = Cart.get_cart_total()


    context = {
        "dept_id" : dept_id,
        "cart_items": cart_items,
        "total_cart_price": total_cart_price
        }
    return render(request, 'checkout.html', context)

# _____________________________________________________Contact
page_____________________________________________

def contact(request):
    dept_id = Department.objects.all()
    print("Department Data:", dept_id)
    total_cart_price = Cart.get_cart_total()

    context = {
        "dept_id" : dept_id,
        "total_cart_price": total_cart_price
        }
    return render(request, 'contact.html', context)

# _____________________________________________________Index
page_____________________________________________

def index(request):
    dept_id = Department.objects.all()
    cart_items = Cart.objects.all()
    total_cart_price = Cart.get_cart_total()

    shop_id = request.GET.get('shop_id')
    search_query = request.GET.get('search')

    product_id = Product.objects.all()

    if shop_id:
        product_id = product_id.filter(Department=shop_id)

    if search_query:
        product_id = product_id.filter(product_name__icontains=search_query)

    context = {
        "dept_id": dept_id,
        "shop_id": shop_id,
        "cart_items": cart_items,
        "product_id": product_id,
        "total_cart_price": total_cart_price,
        "search": search_query
    }
    return render(request, 'index.html', context)


# _____________________________________________________Login
page_____________________________________________

def login(request):
    if request.method == "POST":
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            uid = User_Detail.objects.get(email=email)

            if uid.password == password:
```

```
                contaxt = {
                    'msg': 'Successfully logged'
                }
                return redirect(index)
            else:
                contaxt = {
                    'msg': 'Wrong password'
                }
        except User_Detail.DoesNotExist:

            contaxt = {
                'msg': 'Invalid email, please go to the registration form'
            }

        return render(request, "login.html", contaxt)

    else:
        return render(request, "login.html")

    return render(request, "login.html")

# _____________________________________________________Sign-Up
page_____________________________________________

def signup(request):
    if request.method=='POST':
        f_name = request.POST.get('f_name')
        l_name = request.POST.get('l_name')
        email = request.POST.get('email')
        password = request.POST.get('password')
        c_password = request.POST.get('c_password')
        phone = request.POST.get('phone')
        address = request.POST.get('address')

        print(f_name, l_name, email, password, c_password, phone, address)
        print(f"Password: '{password}', Confirm Password: '{c_password}'")

        try:
            uid = User_Detail.objects.get(email=email)
            contaxt = {
                "msg": "Email already exists"
                }
            return render(request, "signup.html", contaxt)

        except User_Detail.DoesNotExist:

            if c_password != password:
                contaxt = {
                    "msg": "Enter correct password"
                    }
                return render(request, "signup.html", contaxt)

            else:
                contaxt={
                "msg":"Welcome to our family !"
                }
                User_Detail.objects.create(f_name=f_name, l_name=l_name, email=email, password=password,
c_password=c_password, phone=phone, address=address)
                return render(request,"signup.html",contaxt)
    else:
        return render(request, "signup.html")

# _____________________________________________________Main
page_____________________________________________
def main(request):
    dept_id = Department.objects.all()
    print("Department Data:", dept_id)
    context = {
        "dept_id" : dept_id,
        }
    return render(request, 'main.html', context)

# _____________________________________________________Shop-Details
page_____________________________________________

def open_details(request, id):
    product = get_object_or_404(Product, id=id)
    total_cart_price = Cart.get_cart_total()

    context = {
        "product": product,
        "total_cart_price": total_cart_price
    }
    return render(request, 'shop_details.html', context)
```

```python
# _____Shop-Grid page_____
def shop_grid(request):

    dept_id = Department.objects.all()
    product_id = Product.objects.all()
    cid=Color_filter.objects.all()
    sid = Size_filter.objects.all()
    total_cart_price = Cart.get_cart_total()


    shop_id = request.GET.get('shop_id')
    p_color=request.POST.get('p_color')
    p_count = sid.annotate(count=Count('product'))

    print(p_color)

    print("Shop ID:", shop_id)
    print("Department Data:", dept_id)
    print("Product Data:", product_id)

    email = request.session.get('email')

    if email:
        try:
            uid = User_Detail.objects.get(email=email)
            wishlist_ids = Wishlist.objects.filter(user=uid).values_list('product__id', flat=True)
        except User_Detail.DoesNotExist:
            wishlist_ids = []
    else:
        wishlist_ids = []

    if shop_id:
        product_id = Product.objects.filter(Department=shop_id)
    else:
        product_id = Product.objects.all()

    context = {
        "dept_id" : dept_id,
        "product_id" : product_id,
        "shop_id" : shop_id,
        "sid":sid,
        "wishlist_ids": wishlist_ids,
        "p_count": p_count,
        "cid":cid,
        "p_color":p_color,
        "total_cart_price": total_cart_price,

        }
    return render(request, 'shop_grid.html',context)

def get_color(request):
    dept_id = Department.objects.all()
    product_id = Product.objects.all()
    cid=Color_filter.objects.all()
    sid = Size_filter.objects.all()


    cf=request.POST.get('p_color')

    l=[]
    if cf:
        product_id = Product.objects.filter(Color_filter__p_color=cf)
        l.extend(product_id)
        print(product_id)
    else:
        product_id=Product.objects.all()

    p_count = sid.annotate(count=Count('product'))

    context={
        "dept_id" : dept_id,
        "product_id": product_id,
        "cid": cid,
        "cf": cf,
        "sid":sid,
        "p_count": p_count,
    }
    return render(request,'shop_grid.html',context)

def get_size(request):
    dept_id = Department.objects.all()
    product_id = Product.objects.all()
    cid=Color_filter.objects.all()
    sid = Size_filter.objects.all()
```

```
    sf = request.POST.get('p_size')

    l=[]
    if sf:
        product_id = Product.objects.filter(Size_filter__p_size=sf)
        l.extend(product_id)
        print(product_id)
    else:
        product_id=Product.objects.all()

    p_count = sid.annotate(count=Count('product'))

    context ={
        "dept_id" : dept_id,
        "product_id": product_id,
        "cid":cid,
        "sid":sid,
        "sf": sf,
        "p_count": p_count,
    }
    return render(request, 'shop_grid.html', context)

def price_range(request):
    dept_id = Department.objects.all()
    product_id = Product.objects.all()
    cid=Color_filter.objects.all()
    sid=Size_filter.objects.all()

    if request.POST:
        min_price = request.POST['min_price']
        max_price = request.POST['max_price']

        product_id=Product.objects.filter(product_price__lte=max_price[1:],product_price__gte=min_price[1:])

        print(min_price)
        print(max_price)
        print(product_id)

        context ={
            # "dept_id" : dept_id,
            "product_id": product_id,
            # "cid":cid,
            # "sid":sid,
            "min_price": min_price,
            "max_price": max_price,
        }
        return render(request, 'shop_grid.html', context)
    else:
        context ={
            # "dept_id" : dept_id,
            "product_id": product_id,
            # "cid":cid,
            # "sid":sid,
            "min_price": None,
            "max_price": None,
        }
        return render(request, 'shop_grid.html', context)

def wishlist(request, id):

    # uid=Register.objects.get(email = request.POST.get('email'))
    product_id = Product.objects.get(id=id)
    peid = Wishlist.objects.filter(product = product_id).exists()

    if peid:
        Wishlist.objects.filter(product = product_id).delete()
        return redirect('shop_grid')
    else:
        Wishlist.objects.create(product=product_id,
                                product_image=product_id.product_image,
                                product_name=product_id.product_name,
                                product_price=product_id.product_price)
        return redirect('shop_grid')

# _____Shoping Cart
page_____

def shoping_cart(request):
    dept_id = Department.objects.all()
    cart_items = Cart.objects.all()
    total_cart_price = Cart.get_cart_total()

    print("Department Data:", dept_id)
    context = {
```

```
        "dept_id" : dept_id,
        "cart_items": cart_items,
        'total_cart_price': total_cart_price,
        }
    return render(request, 'shoping_cart.html', context)

def add_to_cart(request,id):

    dept_id = Department.objects.all()
    product_id = Product.objects.get(id=id)

    peid = Cart.objects.filter(product=product_id).first()
    print(peid)

    if peid :
        peid.quantity +=1
        peid.total_price = peid.quantity * peid.product_price
        peid.save()

    else:
        Cart.objects.create(product=product_id,
                        product_name=product_id.product_name,
                        product_image=product_id.product_image,
                        product_price=product_id.product_price,
                        quantity=1,
                        total_price=product_id.product_price)

    return redirect('shoping_cart')

def pluscart(request,id):
    cart_item = Cart.objects.get(id=id)
    cart_item.quantity = cart_item.quantity + 1
    cart_item.total_price = cart_item.quantity * cart_item.product_price
    cart_item.save()
    return redirect("shoping_cart")

def minuscart(request,id):
    cart_item = Cart.objects.get(id=id)
    if cart_item.quantity > 1:
        cart_item.quantity -= 1
        cart_item.total_price = cart_item.quantity * cart_item.product_price
        cart_item.save()
    else:
        cart_item.delete()
    return redirect("shoping_cart")

def update_cart(request, id):
    cart_item = Cart.objects.get(id=id)
    cart_items = Cart.objects.all()
    print(cart_item)

    if request.method == "POST":
        quantity = int(request.POST.get("quantity"))

        if quantity > 0:
            cart_item.quantity = quantity
            cart_item.total_price = cart_item.quantity * cart_item.product_price
            cart_item.save()
        else :
            cart_item.delete()

        return redirect('shoping_cart')

    context = {
        "peid":peid,
        "cart_items" : cart_items,
    }
    return render(request, 'shoping_cart', context)

def remove_cart(request, id):
    cart_item = get_object_or_404(Cart, id=id)
    cart_item.delete()
    return redirect('shoping_cart')
```

4.3 System Testing :

4.3.1. Black Box test cases :

Test Case ID	Feature/Module	Test Scenario	Test Steps	Expected Result	Actual Result	Status
BB001	User Registration	User registers with valid data	Fill and submit registration form with valid details	User is registered and redirected to login page	User is registered and redirected to login page	Pass
BB002	User Registration	Attempt registration with missing required fields	Leave one or more fields empty and submit	Error message shown for missing fields	Error message shown for missing fields	Pass
BB003	User Login	Login with valid credentials	Enter correct email and password	User is logged in and redirected to dashboard	User is logged in and redirected to dashboard	Pass
BB004	User Login	Login with invalid credentials	Enter wrong email/password	Appropriate error message displayed	Appropriate error message displayed	Pass
BB005	Product Listing	View product list and details	Visit home/shop page, click on a product	Product details page is displayed	Product details page is displayed	Pass
BB006	Search/Filter Products	Apply category and price filters	Use filter sidebar to choose category/price range	Filtered products are displayed	Filtered products are displayed	Pass
BB007	Add to Cart	Add a product to cart	Click "Add to Cart" on a product detail page	Product is added and reflected in cart page	Product is added and reflected in cart page	Pass
BB008	Cart Operations	Change quantity of items in cart	Increase/decrease item quantity in cart	Quantity updates and total price recalculates	Quantity updates and total price recalculates	Pass
BB009	Checkout Process	Complete checkout with valid billing details	Fill checkout form with valid data and place order	Order is successfully placed and confirmation shown	Order is successfully placed and confirmation shown	Pass
BB010	Checkout Process	Attempt checkout with empty cart	Click checkout without adding items	Appropriate message shown or checkout blocked	Appropriate message shown or checkout blocked	Pass
BB011	Form Validation	Input invalid data (e.g., wrong email format, phone number)	Enter invalid data in forms and submit	Validation errors shown for each field	Validation errors shown for each field	Pass
BB012	Navigation	Navigate across key pages	Use navbar/links to go from Home → Shop → Cart → Checkout	Pages load correctly with proper layout	Pages load correctly with proper layout	Pass
BB013	Responsive UI	View website on different screen sizes	Open site in browser and resize window or use dev tools	Layout adapts (mobile/tablet/desktop)	Layout adapts (mobile/tablet/desktop)	Pass
BB014	Security	Try accessing protected pages without login	Access cart or checkout directly via URL	Redirected to login or shown access error	Redirected to login or shown access error	Pass
BB015	Order Confirmation	Verify confirmation message post-order	Complete order flow with valid details	Success message with order ID is displayed	Success message with order ID is displayed	Pass

4.3.2. White Box test cases :

Test Case ID	Function/Module	Test Type	Test Objective	Test Steps	Expected Output	Actual Result	Status
WB001	register_user()	Condition Testing	Ensure registration logic handles all input validations	Call with valid, missing, and duplicate inputs	Success or appropriate error messages	Sucess	Pass
WB002	login_user()	Decision Testing	Check if login logic correctly distinguishes valid and invalid credentials	Pass combinations of correct and incorrect login credentials	Auth token or login error	Author Token	Pass
WB003	add_to_cart()	Path Testing	Validate all logical paths: new item, update existing item, stock check	Call function with new item, duplicate item, and out-of-stock product	Item added/updated or error returned	Item Added/Updated	Pass
WB004	checkout()	Loop Testing	Ensure all cart items are processed and validated in the loop	Add multiple items to cart, trigger checkout	All items validated and processed	All items validated	Pass
WB005	calculate_total()	Loop + Condition	Ensure discounts, quantity multipliers, and conditions apply correctly	Call with varied product prices, quantities, discount rules	Correct total returned	Correct total returned	Pass
WB006	filter_products()	Path Testing	Validate that all filters (price, category, size, color) are applied correctly	Apply each filter alone and in combination	Only matching products returned	Only matching products returned	Pass
WB007	create_order()	Condition Testing	Verify order creation path: validate billing, cart, generate order, clear cart	Simulate order placement with valid/invalid billing and cart	Order created or appropriate error	Order created	Pass
WB008	update_cart_quantity()	Loop Testing	Ensure quantity updates don't exceed stock and update total correctly	Try increasing quantity of cart item beyond stock	Quantity adjusted or stock error shown	Quantity adjusted	Pass
WB009	validate_billing()	Path & Condition	Ensure all required fields are validated	Submit billing with missing fields like city, state	Error messages returned	No error	Pass
WB010	apply_coupon() (if any)	Condition Testing	Validate coupon logic for expired, invalid, or applicable coupons	Use valid, expired, and inapplicable coupon codes	Correct discount or error applied	Correct discount	Pass

5. EXPERIENCE IN INTERNSHIP

5.1. Work Tasks I Have Been Executing

During my internship, I was entrusted with the responsibility of developing the backend logic for an e-commerce platform named OGANI. My tasks primarily revolved around implementing and managing the core functionality of the system, including setting up Django models, creating and linking views, handling form data, and ensuring secure **CRUD** operations for various modules like User Management, Product Listings, Cart Operations, and Order Processing. I also worked on AJAX-based product filtering, enabling real-time search and category updates without full-page reloads, which significantly improved user experience. Furthermore, I contributed to the creation of dynamic routing, developed multiple templates, and integrated essential features like session handling and validation mechanisms to ensure robust backend processing. I documented system flows and created technical diagrams such as **ER diagrams**, Class Diagrams, and **PERT/CPM** charts to reflect project planning and system architecture.

5.2. Challenges I Have Been Facing While Performing Given Tasks

Despite gaining significant knowledge, I encountered several challenges throughout the internship. One of the recurring issues was managing the integration of backend logic with frontend templates, especially when implementing **AJAX** calls and ensuring seamless data rendering. Another challenge was resolving URL mapping errors, such as **NoReverseMatch**, which occurred due to incorrect argument passing or missing identifiers. Handling static files and CSS rendering in Django during template transitions was also problematic, particularly while configuring static paths and resolving broken links. Additionally, ensuring database integrity while defining relationships among models and debugging logical errors in form processing required meticulous attention. Time management became a concern when coordinating multiple tasks such as diagram creation, testing, and documentation, all while meeting deadlines. However, I overcame these difficulties through consistent learning, debugging, and guidance from my mentors.

5.3. Benefits I Gained from the Internship

This internship proved to be an enriching and transformative experience for my technical and professional development. I significantly improved my understanding of Django framework, especially its MTV architecture, database ORM, and modular development practices. I gained hands-on experience in handling real-time project scenarios, from understanding client requirements to deploying key functionalities. My problem-solving and critical-thinking abilities were enhanced by tackling backend issues and implementing optimized solutions. I also improved my technical documentation skills, having developed several types of system diagrams and flowcharts that supported the project's structural clarity. Moreover, the internship nurtured my teamwork, communication, and project management abilities, preparing me for real-world software development environments. Overall, this experience laid a strong foundation for my future career in backend development and full-stack projects.

6. CONCLUSION

6.1 Brief Description of the Worked-out Tasks :

During the internship, my primary responsibility was to develop and manage the backend logic of the OGANI e-commerce platform using the Django framework. I was involved in the creation of database models, setting up relationships between entities, and writing views to handle various functionalities such as product management, user sessions, cart operations, and billing details.

I also implemented AJAX-based search and filter functionalities, enabling dynamic interaction without full page reloads. In addition, I created technical diagrams including ER diagrams, class diagrams, and flowcharts to represent the system architecture and logic flow. Comprehensive white-box and black-box testing was also conducted to ensure the reliability and accuracy of the backend modules.

My contributions were integral in building a functional and scalable backend system tailored to the needs of an e-commerce platform.

6.2 Further Enhancement :

While the current backend system of the OGANI e-commerce platform is robust and functional, there are several potential areas for future enhancement that could significantly improve the overall system performance, scalability, and user experience.

1. **Integration of RESTful APIs**
   Introducing RESTful APIs would allow the system to decouple the frontend and backend, thereby making it easier to develop mobile applications and enable seamless third-party integrations. This would also improve maintainability and scalability.

2. **Advanced Search and Filter Functionality**
   The current search is AJAX-based, but it could be enhanced with full-text search engines like **Elasticsearch** or **Whoosh** for faster, smarter search results. Filtering by rating, discount, and brand could also improve user engagement.

3. **User Role Management and Admin Dashboard**
   Adding a custom dashboard for administrators and vendors with analytics and management tools would streamline backend operations. A more granular user role system would also enhance data security and access control.

4. **Recommendation System**
   Implementing machine learning-based recommendation algorithms can personalize the shopping experience for users based on their browsing and purchase history.

5. **Automated Testing and CI/CD Pipelines**
   To ensure code quality and deployment efficiency, integration of automated testing and Continuous Integration/Continuous Deployment (CI/CD) tools like GitHub Actions or Jenkins can be highly beneficial.

6. **Performance Optimization and Caching**
   Optimization of database queries, implementation of caching mechanisms like Redis, and use of Content Delivery Networks (CDNs) can significantly enhance site speed and responsiveness.

7. **Security Enhancements**
   Incorporating advanced security features such as two-factor authentication (2FA), rate-limiting, and automated security audits would strengthen the system against potential threats.

7. REFERENCES

7.1 Useful Web links :

1. **Django Documentation**
   Django Software Foundation. (2024). *The Web framework for perfectionists with deadlines.*
   URL: https://docs.djangoproject.com
2. **Python Official Documentation**
   Python Software Foundation. (2024). *Python Language Reference, Version 3.*
   URL: https://docs.python.org
3. **W3Schools – HTML, CSS, JavaScript**
   Refsnes Data. (2024). *W3Schools Online Web Tutorials.*
   URL: https://www.w3schools.com
4. **GeeksforGeeks – Data Structures and Backend Concepts**
   GeeksforGeeks.org. (2024). *Programming and Web Development Tutorials.*
   URL: https://www.geeksforgeeks.org
5. **AJAX and jQuery Documentation**
   jQuery Foundation. (2024). *jQuery API Documentation.*
   URL: https://api.jquery.com
6. **Software Testing Help – Test Case Writing & Methods**
   SoftwareTestingHelp.com. (2024). *Manual and Automation Testing Guides.*
   URL: https://www.softwaretestinghelp.com
7. **Draw.io (diagrams.net)**
   For creating ER diagrams, class diagrams, and logic flowcharts.
   URL: https://app.diagrams.net
8. **TutorialsPoint – PERT & CPM**
   TutorialsPoint. (2024). *Project Management Concepts.*
   URL: https://www.tutorialspoint.com/pert-cpm/index.htm
9. **GitHub**
   GitHub, Inc. (2024). *Version control and collaborative development.*
   URL: https://github.com
10. **Bootstrap Documentation**
    The Bootstrap Team. (2024). *Build fast, responsive sites.*
    URL: https://getbootstrap.com/docs