

CENTRALIZED ADMIN MANAGEMENT WITH ROLE BASED ACCESS CONTROL

**A Project Report Submitted in the fulfillment of the requirement for the
award of degree**

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND TECHNOLOGY

SUBMITTED BY

D.V.SAI GANESH (213C1A0509)

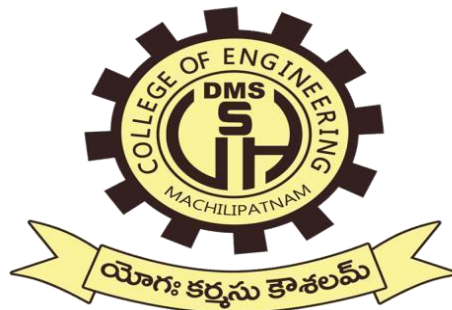
M.RADHA KRISHNA (213C1A0521)

KJEEVANA REKHA (213C1A0519)

Under the Esteemed Guidance of

Smt. N.SUSHMA M.Tech

(Associate Professor, Department of CSE)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

D.M.S.S.V.H COLLEGE OF ENGINEERING

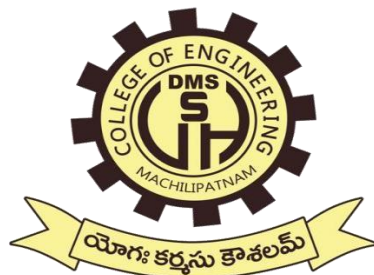
(Affiliated to Jawaharlal Nehru Technological University, Kakinada)

Approved by AICTE and GOVT of AP, An ISO 9001:2015 Certified Institution)

MACHILIPATNAM-521002, KRISHNA, AP

APRIL-2025

D.M.S.S.V.H COLLEGE OF ENGINEERING
Affiliated to Jawaharlal Nehru Technological University, Kakinada
MACHILIPATNAM-521002, KRISHNA, AP



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project work entitled

**CENTRALIZED ADMIN MANAGEMENT WITH ROLE BASED ACCESS
CONTROL**

Is the bonafide work done by

D.V.SAI GANESH (213C1A0509)

M.RADHA KRISHNA (213C1A0521)

K.JEEVANA REKHA (213C1A0519)

Submitted in the partial fulfillment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND TECHNOLOGY

2024-2025

H.O.D

Smt. N.SUSHMA

Head of the Department, CSE

Project Guide

Smt.N.SUSHMA

Head of the Department, CSE

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any work would be incomplete without mentioning the people who made it possible and whose encouragement and guidance has been a source of inspiration throughout the course of the project “**CENTRALIZED ADMIN MANAGEMENT ROLE BASED ACCESS CONTROL**”. We are thankful to sanctum **DMSSVH COLLEGE OF ENGINEERING** for giving us the opportunity to fulfill our aspirations.

We take the opportunity to express our heartfelt gratitude to **Prof.Dr.T.RAVI KUMAR**, Principal for their kind support in doing this project.

We are privileged for giving her continuous support and guidance.

We express our sincere thanks to our guide **Smt.N.Sushma** garu for being a source of inspiration.

We would like to express our deepest gratitude to our Internship trainer **Sri.D.Chinna Venkataswamy** garu, Director, App Genesis Soft Solutions Pvt.Ltd recognized by AICTE&APSCHE. We are especially grateful for your invaluable guidance and mentoring throughout our Internship, which has significantly contributed to our professional growth. Your expertise in "Full Stack Development with Django" helped us to develop strong understanding of industry specific standards and boosted our confidence. Thank you for making this internship so valuable.

Project Team



(213C1A0509)

D.V.SAI GANESH



(213C1A0521)

M.RADHA KRISHNA



(213C1A0519)

K.JEEVANA REKHA

LIST OF CONTENTS

CHAPTER	PAGE NO
1. INTRODUCTION	1–2
1.1 Scope	1
1.2 Purpose of the Project	2
1.3 Key Words	2
2. OBJECTIVE OF THE PROJECT	4–7
2.1 Existing System and Disadvantages	5
2.2 Proposed System and Advantages	6
2.3 Module Description	7
3. LITERATURE SURVEY	8–10
4. SYSTEM ANALYSIS	11–14
4.1 Preliminary Investigation	12
4.2 Request Clarification	12
4.3 Feasibility Analysis	12
4.3.1 Operational Feasibility	13
4.3.2 Economical Feasibility	13
4.3.3 Technical Feasibility	14
5. SYSTEM REQUIREMENT SPECIFICATION	15–17
5.1 Functional Requirements	16
5.2 Non-Functional Requirements	16
5.3 System Requirements	17
6. SYSTEM DESIGN	18–27
6.2 UML Diagrams	21
6.2.1 Use Case Diagram	22

6.2.2 Sequence diagram	23
6.2.3 Class Diagram	24
6.2.4 State Diagram	25
6.2.5 Activity Diagram	26
6.2.6 Compnent Diagram	27
7. SOFTWARE TECHNOLOGIES	28–32
8. IMPLEMENTATION	33–40
9. TESTING AND TEST CASES	41–45
9.1 System Testing	42
9.2 Test Case	44
10. OUTPUT SCREENS	46–50
11. CONCLUSION AND FUTURE SCOPE	51
12. REFERENCES	52

ABSTRACT

ABSTRACT

The project entitled "**Centralized Admin Management with Role-Based Access Control**" aims to develop a secure and efficient administrative management system that simplifies the process of assigning, monitoring, and managing user roles within an organization. Ensuring secure access to resources and data based on user roles is one of the key tasks in modern enterprise and institutional systems. The system is developed using modern technologies that support a robust backend with centralized database integration. It enables administrators to define roles, assign specific permissions, and manage user access effectively across multiple modules. The system enhances organizational security and improves workflow efficiency by ensuring that only authorized users have access to specific functionalities and data. The proposed system consists of role creation modules, user access control panels, and activity monitoring features. It supports scalability and can be extended to include department-level management and cross-functional user role mapping. The use of centralized authentication and authorization makes the overall management process more streamlined and secure. The platform provides a web-based interface to develop and deploy the system across different environments, ensuring platform independence, usability, and administrative transparency.

CHAPTER 1

1. INTRODUCTION

In today's online shopping systems, it is important to have a proper backend system to manage everything like products, users, orders, and deliveries. The **Admin Panel** is the main control system that helps manage and monitor all these activities in one place. Without a proper admin panel, handling large data and users can become confusing and lead to mistakes or security issues.

This project is about creating a Centralized Admin Panel with **Role-Based Access Control (RBAC)** for an e-commerce platform. This means that every user (admin, seller, delivery person) will only have access to the features they need based on their role. This helps keep the system more secure and organized.

The admin panel will allow the admin to perform tasks such as adding or removing products, checking orders, managing users, assigning deliveries, and viewing reports. It is built using modern web technologies and supports a proper database to store and handle all data safely.

The main goal of this admin panel is to make e-commerce management simple, fast, and secure. It helps the admin control everything from one place, reduce manual work, and improve the performance of the entire system

1.1 SCOPE

This project is about creating a system that helps an organization manage users and control what each person can access. In many places, admin work is done manually, which takes time and can cause mistakes. Our system makes this easier by allowing the admin to create roles like Admin, Staff, or User, and give each role certain permissions. This means people will only see and use what they are allowed to. The system will have a login page, a dashboard, and tools to manage users and roles. It will be a website, so it can be used from any computer with internet. This project is mainly for making admin work faster, safer, and more organized.

1.2 PURPOSE OF THE PROJECT

The purpose of this project is to develop a **centralized admin management system with role-based access control for e-commerce platforms**. This system helps in managing different users such as admins, sellers, and customers by giving them specific roles and permissions. By using this system, only authorized users can access certain features, making the platform more secure and well-organized. Role-based access control ensures that users can only perform tasks allowed by their roles, reducing the chances of errors or unauthorized actions. The centralized system makes it easier to monitor activities, manage users, and improve the overall efficiency of the e-commerce platform. This project helps e-commerce businesses run smoothly and safely by giving the right control to the right users.

1.3 KEYWORDS

- Admin panel
- E-commerce
- Login and Logout
- User roles
- Product add/edit/delete
- Order status update
- Customer details
- Secure access
- Dashboard
- Reports and analytics.

CHAPTER 2

2 OBJECTIVE OF THE PROJECT

In the existing e-commerce platforms, admin functionalities are often limited or not well-organized. Admins may have to use multiple tools to manage products, users, and orders, which can lead to confusion and inefficiencies. Role-based access is usually missing or not well-defined, so any user with access can make changes, which is a security risk. Approvals for sellers and logistics partners are mostly manual, usually done through phone calls or emails, which makes the process slow and untrackable. Moreover, the absence of proper reporting tools and dashboards makes it difficult for administrators to monitor platform performance and make informed decisions

Disadvantages

- 1) Lack of Role-Based Access Control: Most existing systems do not provide clear separation of roles, which can lead to unauthorized access and security risks.
- 2) Manual Seller and Logistics Approval: Seller and logistics partner approvals are often handled manually or via email, causing delays and inefficiency.

2.1 Proposed System

Usage of a centralized admin management system with role-based access control in e-commerce and organizational platforms is highly desirable. The proposed system is able to manage users efficiently by assigning specific roles such as Admin, Seller, and Customer, and restricting access based on these roles. It ensures that only authorized users can access or perform certain actions, increasing the overall security and reliability of the system. We use modern web technologies in the proposed system to automatically handle user role assignment, permission control, and activity tracking. This approach is more accurate and efficient than traditional manual methods of access control and user management.

Advantages

- 1)Improved Security with Role-Based Access: Only specific users can access particular sections, which reduces the risk of unauthorized activities.
- 2)Faster Operations with Automation: Seller and logistics approvals are automated and managed within the panel, saving time and improving workflow

2.2 Modules Description

In software development, a module is a separate part of the system that handles a specific function. The Modules Description section explains each part of the project and its purpose. Breaking the project into modules helps in better organization, easier development, and efficient maintenance. Each module in this Admin E-commerce Panel is designed to manage one core feature of the platform. These modules work together to provide a smooth and secure admin experience.

The main modules of the system include:

- Authentication Module – Manages secure admin login/logout.
- Role-Based Access Module – Assigns specific roles (admin, seller, logistics) and permissions.
- Product Management Module – Allows adding, editing, and deleting product information.
- Order Management Module – Helps admins view and update order statuses.
- User Management Module – Provides access to customer details and role assignments.
- Seller Approval Module – Handles approval or rejection of new seller registrations.
- Logistics Approval Module – Manages the approval of logistics and delivery partners.

CHAPTER 3

3.LITERATURE SURVEY

Title: - Role-Based Security in E-Commerce Admin Panels

Authors: R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman

Published in: IEEE Computer, 2000 (Foundational Study)

Role-Based Access Control (RBAC) is a widely accepted method to secure admin panels by assigning permissions to roles instead of individual users. Sandhu et al. emphasized that RBAC minimizes errors, enforces consistent policies, and enhances data protection. In the context of e-commerce admin systems, RBAC ensures that only authorized personnel (such as Admins or Managers) can access sensitive operations like editing products, modifying orders, or managing user data.

RBAC is efficiently implemented in frameworks like Django, where permissions can be dynamically assigned to different user groups. This promotes better system governance, accountability, and access transparency.

The concept of **Role-Based Access Control (RBAC)**, which is now widely implemented in modern web frameworks such as Django. RBAC enhances the security and efficiency of admin systems by assigning specific roles (e.g., Admin, Manager, Staff) with defined permissions. It restricts unauthorized access to critical operations and provides scalable control over user access rights within e-commerce admin dashboards.

Title :- Admin Dashboard Design Patterns for E-Commerce Applications

Authors: N. Joseph, M. Kumar, S. Rathi

Reference: International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 11, Issue 5

Admin dashboards serve as the central control hub for e-commerce platforms, offering interfaces to monitor performance, manage inventory, analyze customer behavior, and handle orders. The design and user experience (UX) of these dashboards play a crucial role in the efficiency of administrative tasks. This paper explores effective design patterns and frontend integrations that contribute to building high-quality, responsive admin dashboards.

The authors emphasize the importance of separating frontend and backend concerns by leveraging Django's templating system and integrating it with frontend libraries like Bootstrap, Tailwind CSS, and JavaScript-based charting tools. Dynamic tables, data filters, sorting mechanisms, and responsive grids are implemented to ensure that the UI adapts seamlessly across devices.

Usability testing conducted as part of the research indicates that dashboards with intuitive navigation, color-coded status indicators, and keyboard shortcuts significantly enhance the productivity of admin users. For example, a product inventory table with color-coded low-stock alerts helps warehouse managers take action quickly.

CHAPTER 4

4. SYSTEM ANALYSIS

4.1 PRELIMINARY INVESTIGATION

The very first step in the development of this project begins with the idea of creating an Admin eCommerce Panel that allows business owners or store managers to efficiently handle their digital store from a centralized platform. This panel is intended to support the management of products, users, orders, and real-time analytics. Once the idea was approved by the organization and project guide, the initial activity, namely Preliminary Investigation, was conducted. This activity consists of three major components:

- Request Clarification
- Feasibility Study
- Request Approval

4.2 REQUEST CLARIFICATION

After obtaining approval from both the organization and the project guide, and with the project under consideration for development, it becomes essential to examine the request in detail to determine the exact requirements of the proposed system. In this case, our project focuses on creating a centralized admin management system, designed specifically for internal users within the company who will access the system based on defined roles and responsibilities.

The system architecture is intended for environments where user access can be securely managed through role-based permissions, allowing each user to interact with the system based on their assigned privileges. Given the growing complexity in managing administrative tasks across various departments, a centralized platform with structured access control is crucial.

4.3 FEASIBILITY ANALYSIS

An essential outcome of the preliminary investigation is the determination of whether the system is feasible within the available time, resources, and budget.

The following feasibility studies were conducted:

- Operational Feasibility
- Economic Feasibility
- Technical Feasibility

4.3.1 Operational Feasibility

Operational feasibility examines how well the proposed system solves the current problems and how it will function once implemented. This Admin Panel reduces the burden on the admin by automating repetitive tasks such as stock updates, order tracking, and customer handling. This leads to significant time and resource savings and increases productivity. Based on the analysis, the system has been found operationally feasible and supports smooth business operation.

4.3.2 Economic Feasibility

Economic feasibility evaluates whether the benefits of the system outweigh the costs involved. In this case, the system is designed to run on existing hardware and infrastructure, significantly reducing setup costs. As the panel is web-based, multiple admins or staff within the organization can access it using standard browsers over a Local Area Network or the internet. Hence, the development and deployment of the Admin eCommerce Panel is economically feasible.

4.3.3 Technical Feasibility

Backend Framework: Django (Python-based web framework)

Database: SQL (Structured Query Language)

Frontend Technologies: HTML, CSS, and JavaScript for building responsive and interactive user interfaces.

CHAPTER 5

5. SYSTEM REQUIREMENTS SPECIFICATION

5.1 Functional Requirements

1.Admin Login/Logout

- Secure login for admin users using username/email and password.

2. Role-Based Access Control

- Different roles (e.g., Super Admin, Manager) with specific Permissions.

3. Product Management

- Add, edit, delete, and view products with details (name, price, image)

4. Order Management

- View all customer orders.
- Update order statuses like "Pending", "Shipped", "Delivered".

5.User Management

- Assign or modify user roles.

5.2 Non- Functional Requirements

1. Usability:

- User-friendly dashboard with a clean interface.
- Easy navigation for non-technical users.

2.Availability

- The system should be available 24/7 with minimal downtime.

3.Data Integrity

- All data entered should be accurate and securely stored in the database.

4.Scalability

- The system should support future expansion, such as adding new modules

5.3 SYSTEM REQUIREMENTS

5.3.1 HARDWARE REQUIREMENTS

- **Processor:** Intel Core i3 or higher
- **RAM:** Minimum 8 GB
- **Hard Disk:** Minimum 320 GB
-

5.3.2 SOFTWARE REQUIREMENTS

- **Programming Language:** Python
- **Web Framework:** Django
- **Database:** SQLite / MySQL / PostgreSQL
- **Frontend Technologies:** HTML, CSS, JavaScript
- **IDE/Code Editor:** Visual Studio Code
- **Browser:** Google Chrome / Mozilla Firefox (for testing)
- **Operating System:** Windows 10 or above

CHAPTER 6

6. SYSTEM DESIGN

6.1 DATA FLOW DIAGRAM

1. The **Data Flow Diagram (DFD)**, also referred to as a *bubble chart*, is a graphical representation used to visualize the flow of data in a system. In our *Centralized Admin Management System with Role-Based Access*, DFDs help represent how data enters the system, how it is processed based on user roles, and how the output is generated and delivered securely.

2. The DFD serves as a crucial modeling tool for our project. It captures various **system components**, including:

- **Processes** such as login authentication, role validation, and data access.
- **Data stores** like user credentials, permissions database, and logs.
- **External entities** such as admins, managers, and general users who interact with the system.
- **Data flows** like user credentials submission, access rights retrieval, and data modification requests.

3. The DFD illustrates **how information flows** through our centralized system. For example:

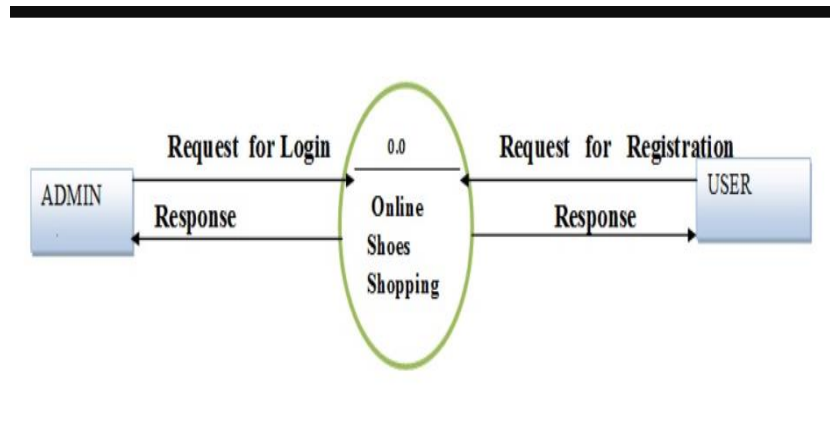
- A user logs in and the credentials flow to the *Authentication Process*.
- Based on the role (e.g., admin, manager, viewer), access is granted to different parts of the system.
- Actions like creating users, assigning roles, or viewing reports flow through the system's logic.

4. The DFD can be **used at multiple levels of abstraction**, such as:

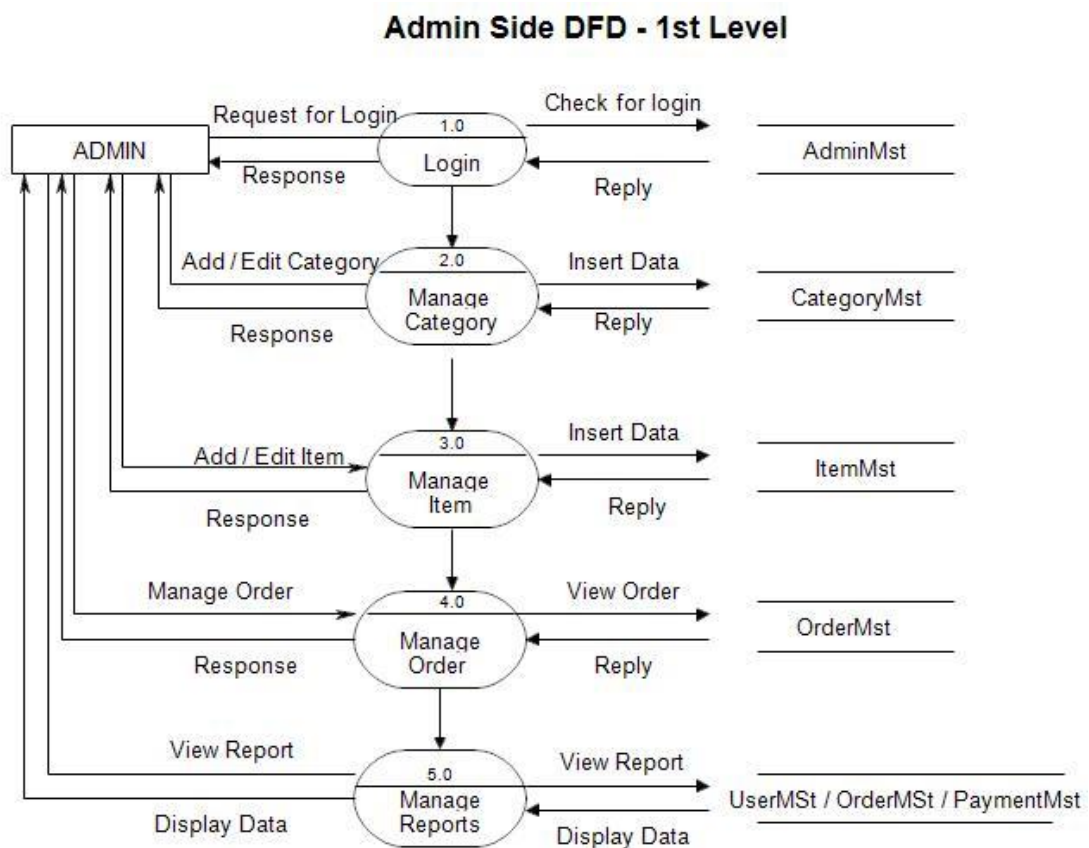
- **Level 0 (Context Diagram)**: Showing the system as a single process with interactions from external entities.

Level 1: Breaking down the system into sub-processes like Role Management, Access Control, and Data Handling.

LEVEL 0:



LEVEL 1:



6.2 UML DIAGRAM

The **UML (Unified Modeling Language) diagram** for the **Centralized Admin Management System with Role-Based Access** is a visual representation of the system's structure and behavior. It is used to model the interactions between different users (such as Admins, Managers, and Viewers) and the system functionalities such as user management, role assignment, and permission-based access control.

This diagram helps in understanding how different components of the system interact with each other, ensuring that the software design follows object-oriented principles. It also assists in visualizing how users with specific roles can access different modules of the system securely. Using UML diagrams in this project enables better planning, development, communication, and documentation, which are essential for building a secure and scalable admin management system with controlled access based on user roles. UML can be integrated into tools for code generation, documentation, and version control.

GOALS

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to use, expressive visual modeling
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.

6.2.1 USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor.

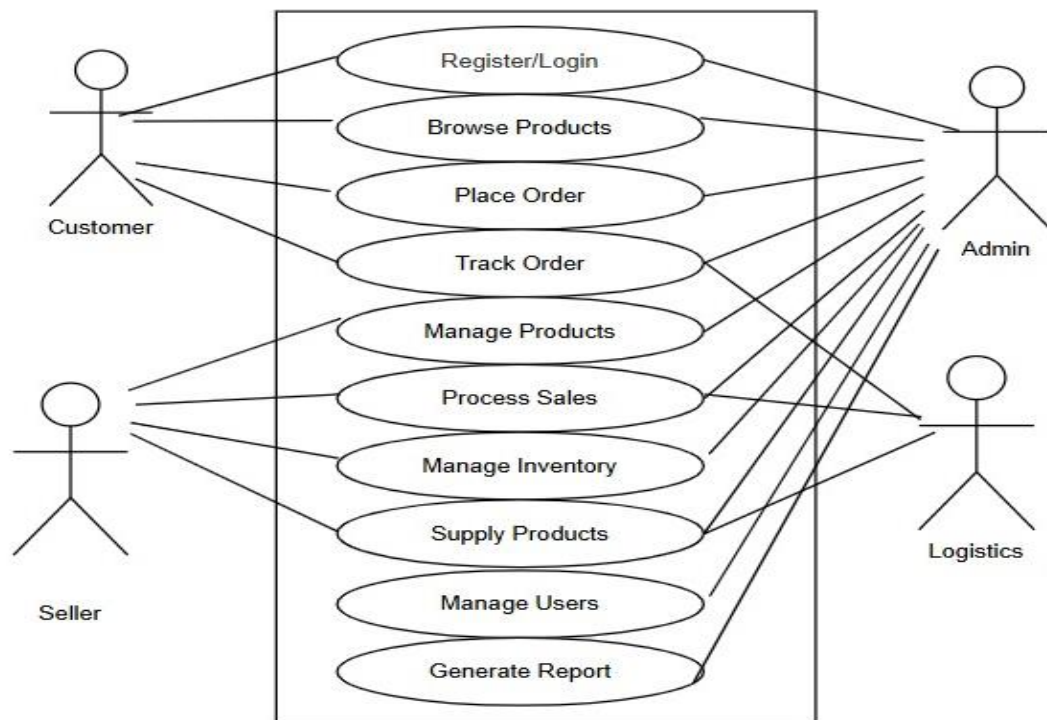


Fig 6.2.1: UseCase Diagram

DESCRIPTION:

This use case diagram represents an e-commerce platform where Customers, SellersAdmins, and Logistics interact with various system functionalities. It outlines user-specific actions such as managing products, placing orders, tracking orders, and generating reports.

6.2.2 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams.

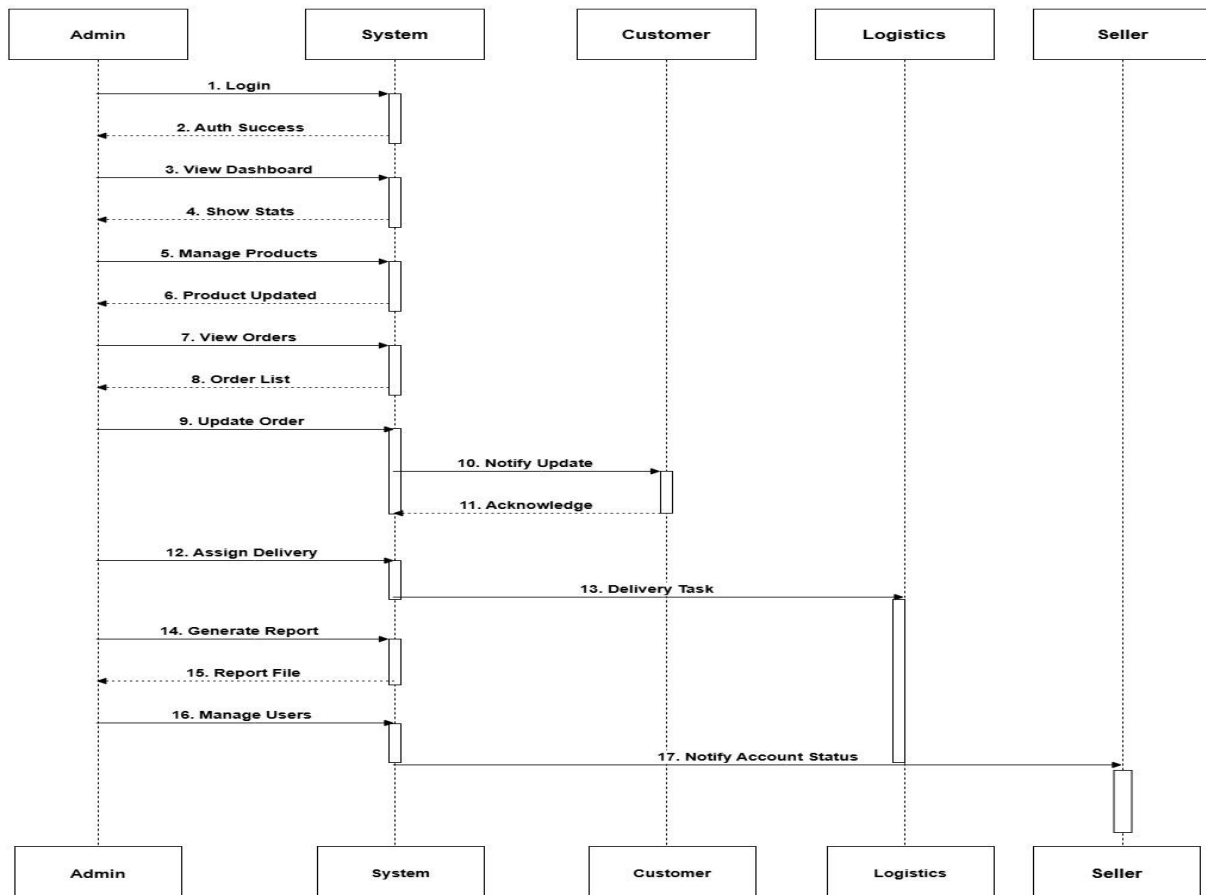


Fig 6.2.2: sequence Diagram

DESCRIPTION:

This sequence diagram illustrates the interaction flow in an e-commerce admin panel, starting from admin login to managing products, orders, deliveries, reports, and user accounts. It shows how the system communicates updates to customers, logistics, and sellers at each stage.

6.2.3 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

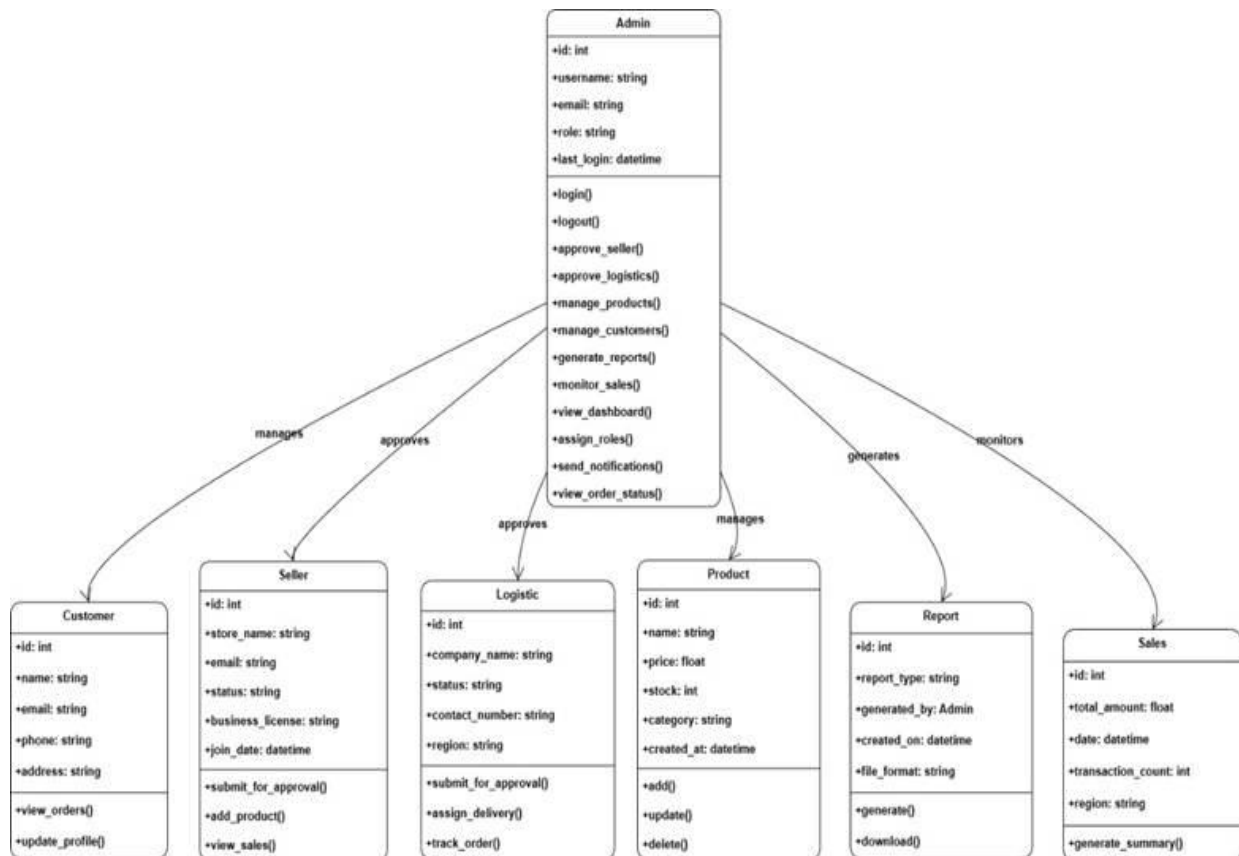


Fig 6.2.3: Class Diagram

DESCRIPTION:

This class diagram illustrates the core components of an E-Commerce Admin Panel, showing relationships between Admin, Customer, Product, Order, Report, and Delivery. Each class contains key attributes and methods that support management, ordering, tracking, and reporting functionalities.

6.2.4 STATE CHART DIAGRAM

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

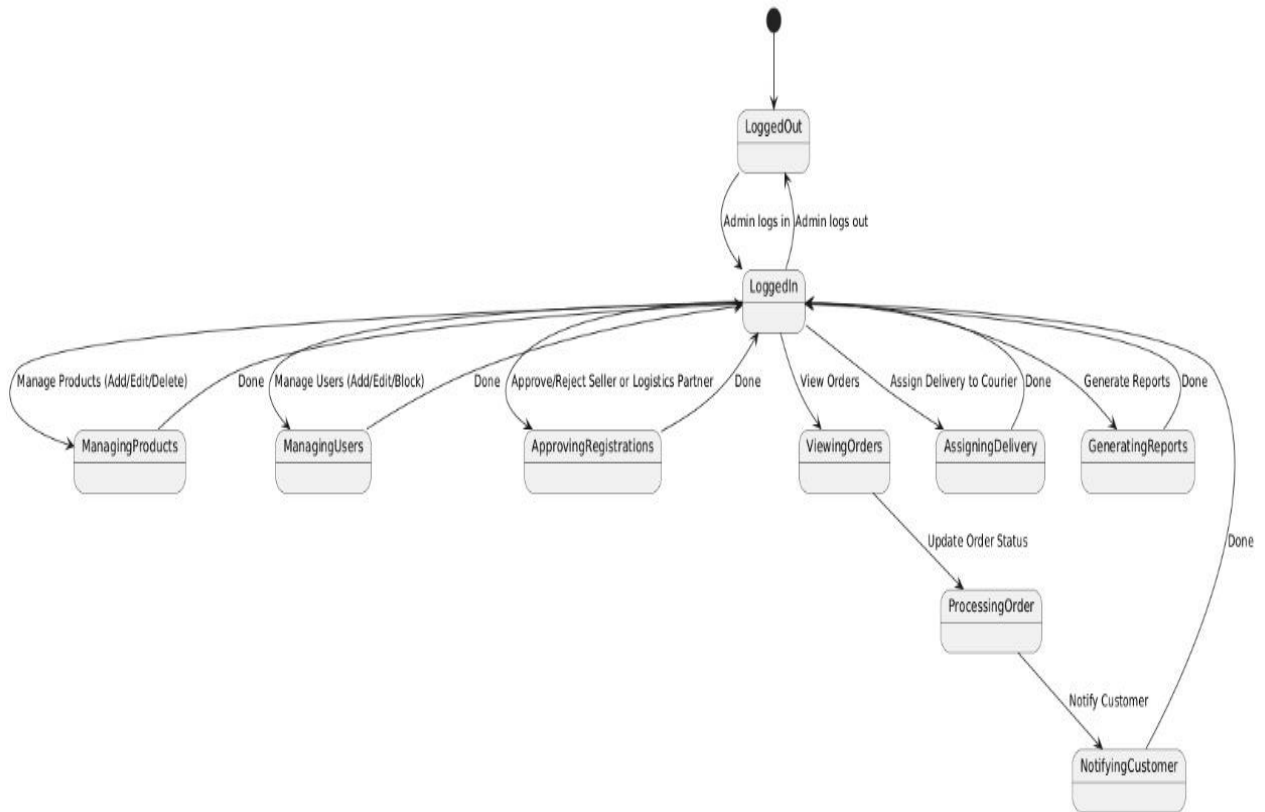


Fig 6.2.4: State Chart Diagram

DESCRIPTION:

This state diagram illustrates the admin's workflow within the e-commerce panel, transitioning from login to various operational tasks like managing products, users, and orders. It clearly maps out state changes and actions such as assigning deliveries and notifying customers post order processing.

6.2.5 ACTIVITY DIAGRAM:

Activity diagrams show the step-by-step flow of a process or task. They help visualize decisions, loops, and actions in a system.

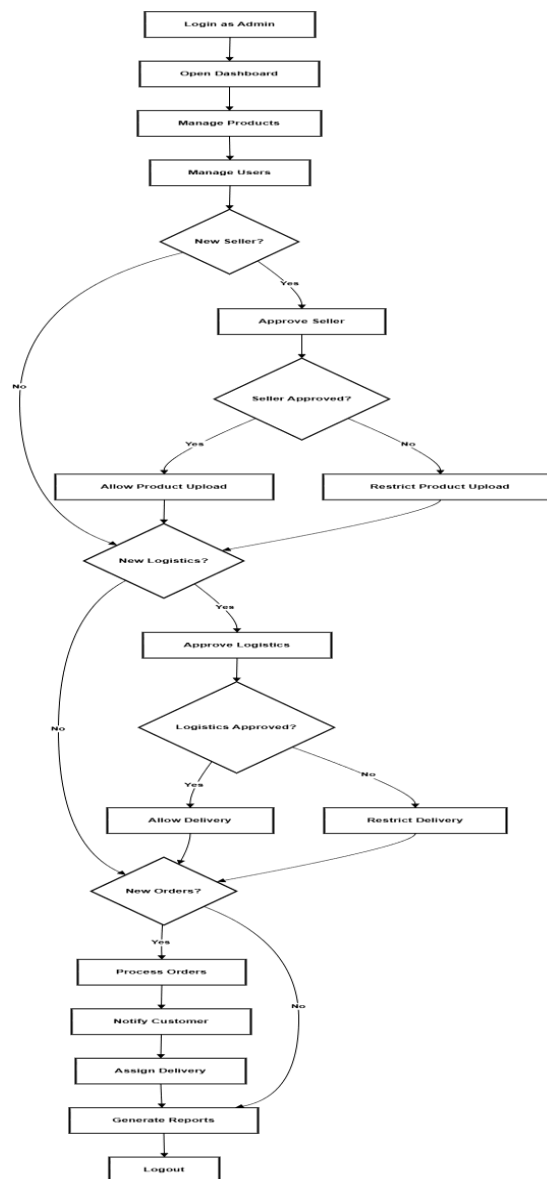


Fig 6.2.6: Activity diagram

DESCRIPTION: This activity diagram outlines the admin's end-to-end operational flow—from logging in, managing products and users, approving sellers/logistics, to processing orders, notifying customers, and generating reports. It includes decision nodes for conditional tasks and ensures a structured sequence for e-commerce administration.

6.2.6 COMPONENT DIAGRAM

A component diagram is used to breakdown a large object- oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides with in the node.

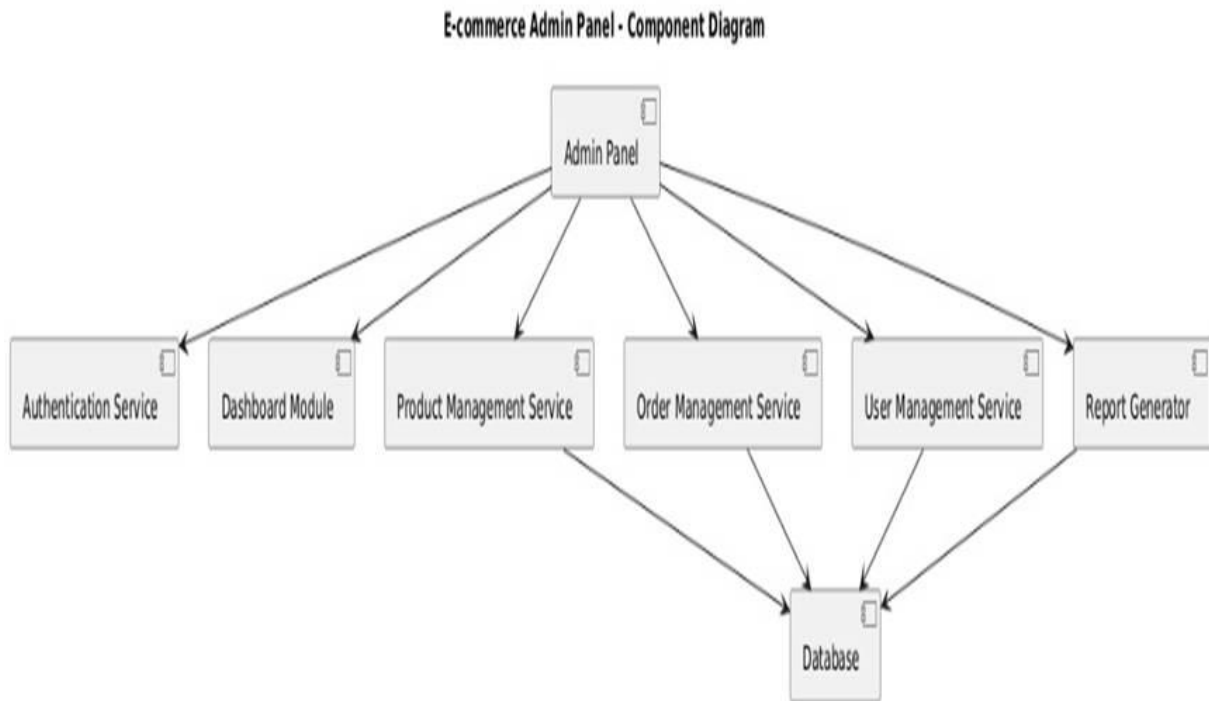


Fig 6.2.6: component diagram

DESCRIPTION:

This component diagram represents the architecture of an E-commerce Admin Panel, showing its interaction with services like authentication, dashboard, product, order, user management, and reporting. Each service communicates with a central database to manage and retrieve data effectively.

CHAPTER 7

7. SOFTWARE TECHNOLOGIES

The successful implementation of the *Online Shopping System – Admin Panel* relies heavily on a combination of modern, powerful technologies. This section highlights the core technologies used in building the application, covering the backend, frontend, templating engine, database solutions, and development tools. The selection of each technology was made with scalability, ease of development, and maintainability in mind.

BACKEND TECHNOLOGY: DJANGO (PYTHON WEB FRAMEWORK)

Django is a high-level Python web framework that promotes rapid development and a clean, pragmatic design. It follows the *Model-View-Template (MVT)* architectural pattern and comes equipped with a range of built-in features that simplify the creation of robust web applications.

Key Features of Django:

- **Security-first approach:** Prevents XSS, CSRF, SQL injection, etc.
- **ORM (Object Relational Mapper):** Handles database queries using Python code.
- **Automatic Admin Panel:** Simplifies model data management.
- **Scalability & Flexibility:** Suitable for small to enterprise-level projects.
- **Rapid Development:** Ideal for quickly building production-ready applications.

FRONTEND TECHNOLOGIES: HTML, CSS, & JAVASCRIPT:

The frontend layer focuses on building a visually appealing and user- friendly interface for both the admin and end-users.

A. HTML (Hyper Text Markup Language) : HTML is used to define the structure of webpages*such as product pages, cart layout, dashboard statistics, and login pages.

B. CSS (Cascading Style Sheets) : CSS provides styling and layout control for the website. Custom styles were written to ensure consistency across devices using media queries, and dynamic components were designed with hover effects, transitions, and responsive layouts.

C. JavaScript: JavaScript was used to enhance interactivity. Common functionalities like input validations, modal pop-ups, cart interactions, and dynamic UI updates were handled using vanilla JavaScript.

DJANGO TEMPLATE ENGINE:

Django's template system bridges the gap between backend logic and frontend design. Templates allow dynamic data from Python models to be rendered directly into HTML.

Key Features:

- **Template Inheritance:** Enables code reuse (e.g., base.html)
- **Dynamic Data Rendering:** Injects data using double curly braces `{{ }}`
- **Control Structures:** Uses `{% if %}`, `{% for %}` tags
- **Filters & Tags:** Used to modify variables before display

Application in Project:

- Dynamic display of product listings
- Admin dashboard showing orders, customer statistics
- Search results rendering based on queries
- Modular code separation for consistent layout

DATABASE SYSTEMS:

Data is central to any web application. This project utilized two different databases at different stages of the development lifecycle.

A. SQLite (For Testing/Development)

SQLite is a serverless, file-based lightweight database used for initial testing, prototyping, and small-scale development.

B. MySQL (For Production/Deployment)

MySQL is an open-source relational database management system widely used in production environments. It was integrated using Django's ORM for better scalability.

VERSION CONTROL AND DEVELOPMENT ENVIRONMENT

Efficient development and collaboration were ensured using industry-standard tools

Git & GitHub:

Git was used for version control, allowing multiple versions of the project to be maintained safely. GitHub was the hosting platform for backups and collaboration.

Visual Studio Code:

VS Code served as the primary development IDE due to its lightweight nature and strong Python/Django support through extensions.

7.6 OVERALL TECHNOLOGY STACK (SUMMARY)

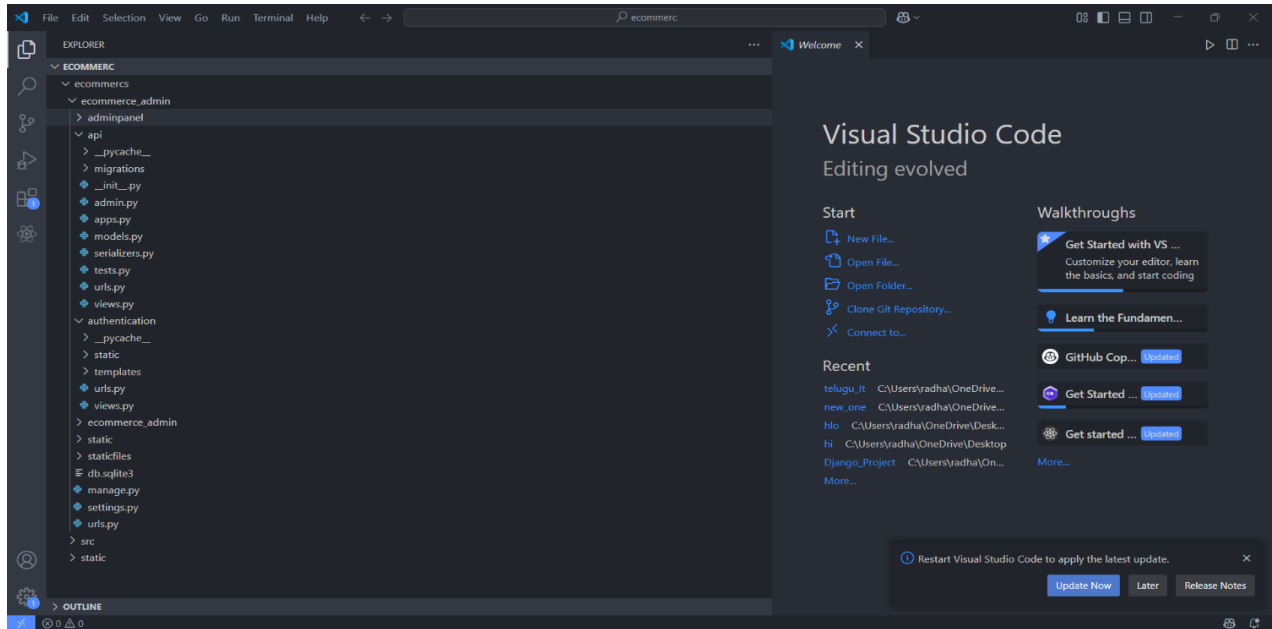
Layer	Technology Used
Backend	Django (Python)
Frontend	HTML, CSS, JavaScript
Templating	Django Template Engine
Database	SQLite (testing), MySQL (production)
Version Control	Git & GitHub
Development IDE	Visual Studio Code

CHAPTER 8

8. IMPLEMENTATION

Project Structure Screenshot:

Include a screenshot of your main Django project folder



1. models.py

(Contains your database design)

```
from django.db import models
```

```
from django.contrib.auth.models import User
```

```
# Category model for products
```

```
class Category(models.Model):
```

```
    name = models.CharField(max_length=100)
```

```
    description = models.TextField()
```

```
    def __str__(self):
```

```
        return self.name
```

```
# Product model for items sold on the e-commerce platform
```

```
class Product(models.Model):
```

```
    name = models.CharField(max_length=255)
```

```
    description = models.TextField()
```

```
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    stock = models.IntegerField()
```

```
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

```
    image = models.ImageField(upload_to='products/')
```

```

created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

def __str__(self):
    return self.name

# Seller model to link with users and store additional info about sellers
class Seller(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    company_name = models.CharField(max_length=255)
    address = models.TextField()
    phone_number = models.CharField(max_length=15)

    def __str__(self):
        return self.company_name

# Order model for customer orders
class Order(models.Model):
    STATUS_CHOICES = [
        ('PENDING', 'Pending'),
        ('SHIPPED', 'Shipped'),
        ('DELIVERED', 'Delivered'),
        ('CANCELLED', 'Cancelled'),
    ]
    customer = models.ForeignKey(User, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.IntegerField()
    order_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default='PENDING')

    def __str__(self):
        return f'Order #{self.id} by {self.customer.username}'

# Admin model for managing platform users
class Admin(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    role = models.CharField(max_length=50, choices=[('Super Admin', 'Super Admin'), ('Admin', 'Admin')])

    def __str__(self):
        return f'{self.role} - {self.user.username}'

```

2. views.py

(Handles the logic)

```

from orders.models import Order # For customer_dashboard

# ◆ Home Page View
def home(request):
    return render(request, "home.html")

# ◆ Login View
def user_login(request):
    if request.user.is_authenticated:
        return redirect_dashboard(request.user)

    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)

        if user:
            if not user.role:
                messages.error(request, "Your account does not have a role assigned. Please contact support.")
                return redirect("login")

            login(request, user)
            messages.success(request, f"Welcome {user.username}!")
            return redirect_dashboard(user)
        else:
            messages.error(request, "Invalid username or password.")
    return render(request, "accounts/login.html")

# ◆ Logout View
@login_required
def user_logout(request):
    logout(request)
    messages.success(request, "You have been logged out successfully.")
    return redirect("login")

# ◆ Admin Dashboard
@login_required
@role_required("Admin")
def admin_dashboard(request):
    users = CustomUser.objects.all()
    vendor_type = VendorType.objects.all()

```



```

return render(request, "accounts/admin_dashboard.html", {"users": users, "vendor_types": vendor_types})

# ♦ Vendor Dashboard
@login_required
@role_required("Vendor")
def vendor_dashboard(request):
    return render(request, "vendors/vendor_dashboard.html")

# ♦ Customer Dashboard - shows user's orders with products
@login_required
@role_required("Customer")
def customer_dashboard(request):
    orders = Order.objects.filter(customer=request.user).prefetch_related("order_items__product")
    view_cart_url = reverse("cart:view_cart")
    return render(request, "accounts/customer_dashboard.html", {
        "orders": orders,
        "view_cart_url": view_cart_url,
    })

# ♦ Logistics Dashboard
@login_required
@role_required("Logistics")
def logistics_dashboard(request):
    return render(request, "accounts/logistics_dashboard.html")

```

4.urls.py

(Shows how your routes are defined)

```

from django.contrib import admin
from django.urls import path, include
from django.shortcuts import redirect # ↪ for redirect
from django.contrib.auth.views import LogoutView # Import Django's built-in LogoutView
from django.conf.urls import handler404
from django.shortcuts import render

urlpatterns = [

    path("login/", views.user_login, name="login"),
    path("logout/", views.user_logout, name="logout"),

```

```

path("admin-dashboard/", views.admin_dashboard, name="admin_dashboard"),
path("vendor-dashboard/", views.vendor_dashboard, name="vendor_dashboard"),
path("customer-dashboard/", views.customer_dashboard, name="customer_dashboard"),
path("logistics-dashboard/", views.logistics_dashboard, name="logistics_dashboard"),
]

```

3. Templates (HTML Files)

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Page Not Found</title>
    <style>
        @import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap');

    </style>
</head>
<body>
    <div class="error-container">
        <h1>404</h1>
        <p>Oops! The page you are looking for does not exist.</p>
        <p><a href="/">Go back to Home</a></p>
    </div>
</body>
</html>

```

5. settings.py

(Shows installed apps, database settings, static files, etc.)

```

import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'django-insecure-!63fd(r*8@-r2!@9^qm_mff{cxn=m=zryjpp-8bu9d0w88hs9g'
DEBUG = True
LOGIN_URL = '/login/'
# Application definition

```

```

INSTALLED_APPS = [

    # Custom Apps

    'accounts',
    'customers',
    'products',
    'orders',

    'payments',
    'logistics',
    'vendors',
    'adminpanel',
    'reviews', # New reviews app
    'cart',
    "widget_tweaks",
    'django.contrib.humanize',
    # 'inventory',
]

# TEMPLATES configuration
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / "templates"],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

# Static & Media Files Configuration
STATIC_URL = "/static/"
STATICFILES_DIRS = [BASE_DIR / "static"]

MEDIA_URL = "/media/"
MEDIA_ROOT = BASE_DIR / "media"

```

```
# Authentication URLs
LOGIN_URL = "/login/"
LOGIN_REDIRECT_URL = "/customer-dashboard/"
LOGOUT_REDIRECT_URL = "/login/"
```

```
# Database settings (using SQLite for development)
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

CHAPTER 9

9 TESTING AND TEST CASES

9.1 SYSTEM TESTING

Introduction to Testing

Testing is a crucial part of the Software Development Life Cycle (SDLC), ensuring that each module performs as intended and meets the defined requirements. For this Admin Panel E-commerce system built with Django, we applied both **manual** and **automated** testing strategies.

Key focus areas during testing included:

- Ensuring backend logic performed accurately.
- Validating data consistency across views and models.
- Verifying the user interface and experience across user roles (admin, seller, and customer).

Types of Testing Performed

Unit Testing:

Unit testing was carried out to test individual components such as Django views, models, and forms. This ensured that each function and method worked correctly on its own before integrating it into the full system.

Integration Testing:

Integration testing focused on verifying that the frontend templates and backend logic worked together seamlessly. It helped us identify and fix any issues in the interaction between different modules.

System Testing:

System testing involved checking the entire application as a whole. We tested end-to-end processes like user login, product management, and the order flow to ensure the complete system functioned properly.

User Acceptance Testing (UAT):

UAT was performed to validate that the system met all business requirements. Real users tested the platform to confirm that it was user-friendly, functional, and ready for deployment.

Database Testing:

Database testing ensured that data operations like insertion, deletion, updating, and retrieval worked correctly. We performed tests on both the SQLite (for development and testing) and MySQL (for production) databases.

White Box Testing :

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing :

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. You cannot “see” into it.

9.2 TEST CASES:

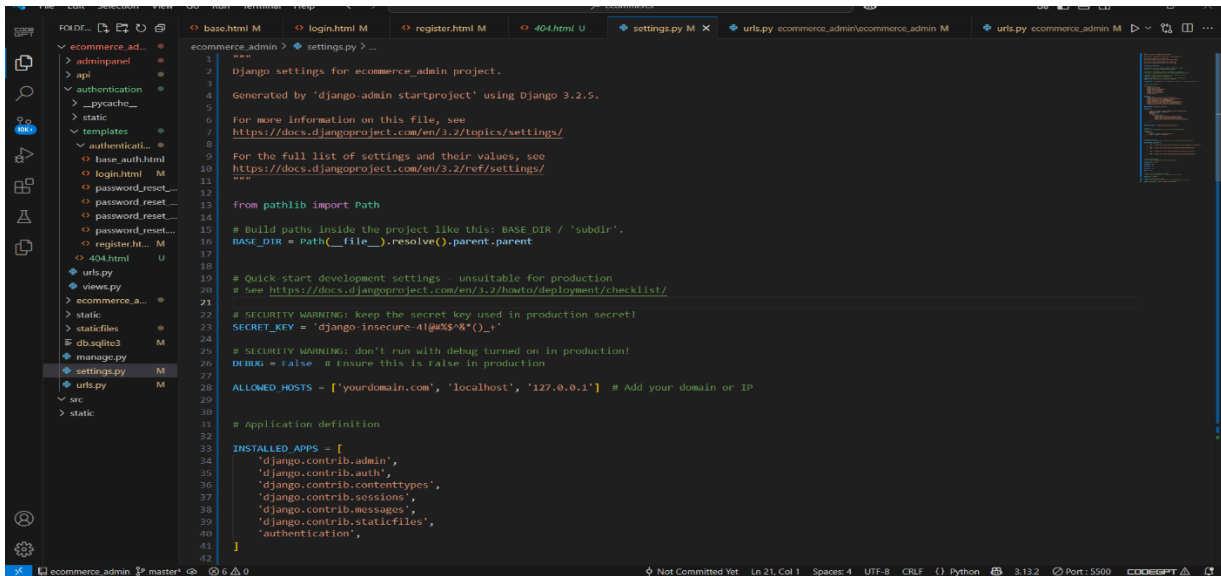
Test Case Id	Test Case Name	Test Case Desc	Test Steps	Expected Result	Test Case Status
TC001	Admin Login	Verify login with valid credentials	Open Admin Panel → Enter credentials → Submit	Dashboard should be displayed	Pass
TC002	Admin Logou	Ensure secure logout	Click logout button	Redirected to login page	Pass
TC003	Seller Approval	Approve a new seller	Go to Sellers → Select pending seller → Click Approve	Seller marked as “Approved”	Pass
TC004	Logistics Partner Approval	Approve logistics provider	Go to Logistics → Approve partner	Status set to “Approved”	Pass
TC005	Product Upload	Upload product as admin	Go to Products → Add product → Fill details → Submit	Product listed in the catalog	Pass

TC006	Product Restriction	Restrict upload by unapproved seller	Try uploading from unapproved seller account	Upload blocked with warning	Pass
TC007	Invalid Login Attempt	Check login fails for wrong credentials	Enter wrong email/password → Click Login	Error message displayed	Pass
TC008	Change Password	Admin can update own password	Go to admin panel → Change Password	Password updated and login required again	Pass

CHAPTER 10

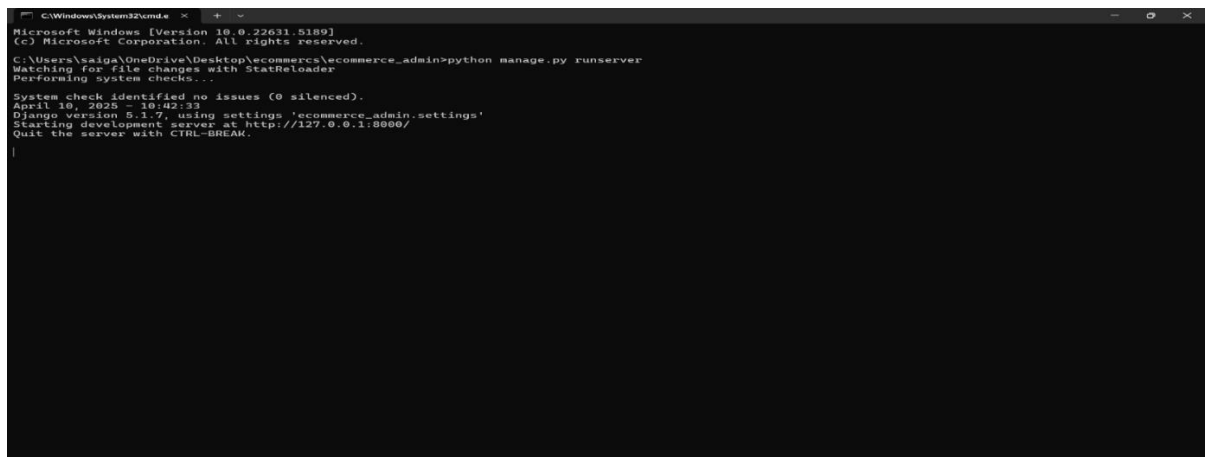
10. OUTPUT SCREENS

The following figure shows the visual studio code:



Description: This is the main code editor used to build and manage the project.

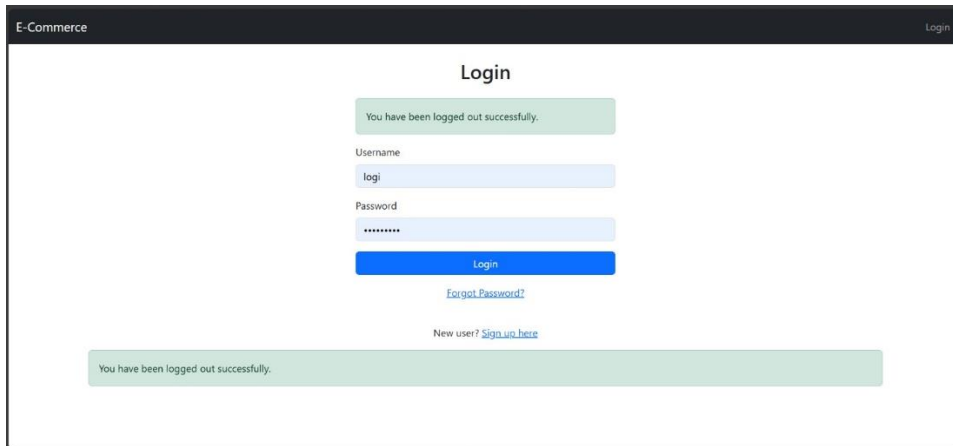
Command Prompt (CMD):



Description:

This window is used to run the Django project using command line instructions.

Admin Login Page:

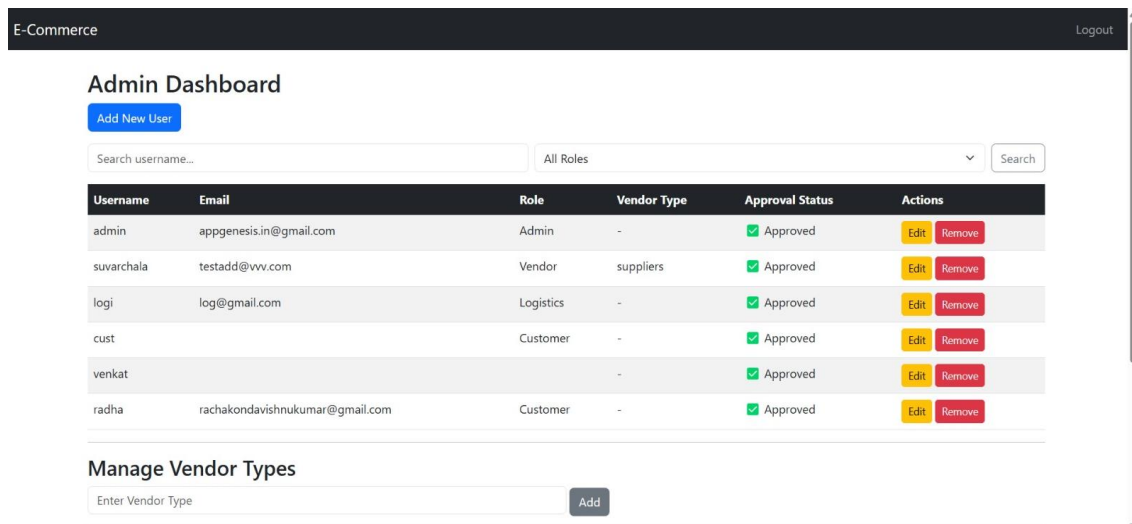


The screenshot shows the Admin Login page of an E-Commerce system. The page has a dark header with 'E-Commerce' on the left and 'Login' on the right. The main content area is titled 'Login'. It features a green success message: 'You have been logged out successfully.' Below this, there are input fields for 'Username' (containing 'logi') and 'Password' (masked with '*****'). A blue 'Login' button is positioned below the password field. A link for 'Forgot Password?' is located below the login button. At the bottom, there is a link for 'New user? Sign up here'. A second green success message is displayed at the very bottom of the form area.

Description:

This page is part of the role-based access system. It allows admin users to login and manage the e-commerce project from the admin panel.

Admin Dashboard:



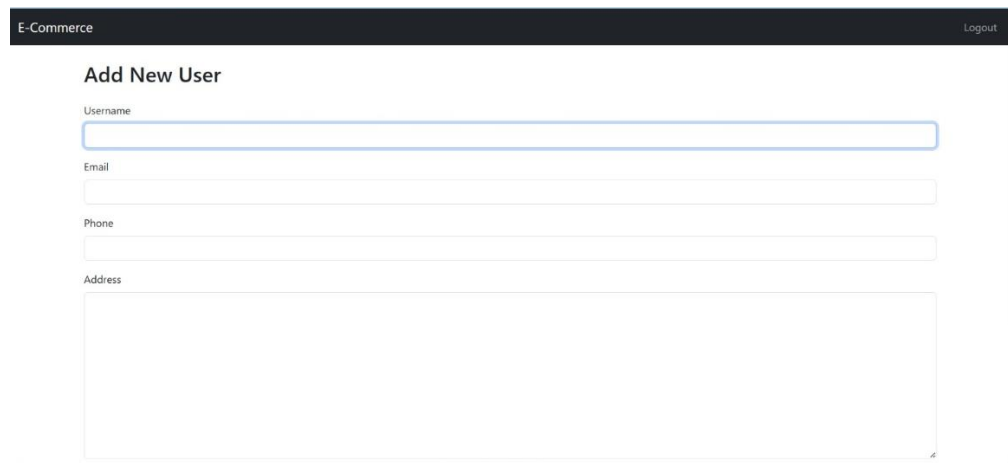
The screenshot displays the Admin Dashboard of an E-Commerce system. The header includes 'E-Commerce' and a 'Logout' link. The main section is titled 'Admin Dashboard' and contains an 'Add New User' button. Below this is a search bar with the placeholder 'Search username...' and a dropdown menu for 'All Roles'. A table lists users with columns for Username, Email, Role, Vendor Type, Approval Status, and Actions. The table contains six rows of user data. Below the table is a section titled 'Manage Vendor Types' with an input field for 'Enter Vendor Type' and an 'Add' button.

Username	Email	Role	Vendor Type	Approval Status	Actions
admin	appgenesis.in@gmail.com	Admin	-	✓ Approved	Edit Remove
suvarchala	testadd@vww.com	Vendor	suppliers	✓ Approved	Edit Remove
logi	log@gmail.com	Logistics	-	✓ Approved	Edit Remove
cust		Customer	-	✓ Approved	Edit Remove
venkat			-	✓ Approved	Edit Remove
radha	rachakondavishnukumar@gmail.com	Customer	-	✓ Approved	Edit Remove

Description:

This is the Admin Dashboard of an eCommerce platform where the admin can manage users such as **sellers, logistics, and customers**. It allows the admin to **approve roles**,

Add New User:

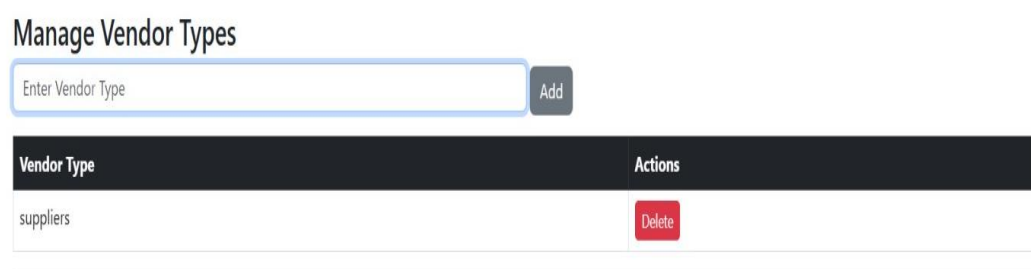


The screenshot shows a web interface for adding a new user. At the top, there is a dark header bar with 'E-Commerce' on the left and 'Logout' on the right. Below the header, the title 'Add New User' is centered. The form consists of four input fields: 'Username' (a single-line text box), 'Email' (a single-line text box), 'Phone' (a single-line text box), and 'Address' (a larger multi-line text area). The 'Username' field is currently selected, indicated by a blue border.

Description:

This page allows the admin to manually add new users by entering their username, email, phone number, and address. It simplifies user registration and management within the e-commerce system.

Manage Vendor Type:



The screenshot shows a web interface for managing vendor types. At the top, the title 'Manage Vendor Types' is displayed. Below the title is a form with a text input field labeled 'Enter Vendor Type' and a grey 'Add' button. Below the form is a table with two columns: 'Vendor Type' and 'Actions'. The table has one row with the value 'suppliers' in the 'Vendor Type' column and a red 'Delete' button in the 'Actions' column.

Vendor Type	Actions
suppliers	Delete

Description:

The "Vendor Type" section allows the admin to add and manage different categories of vendors such as suppliers or distributors. It helps organize vendors based on their specific roles in the e-commerce system.

Pending Product Approvals:

Pending Approval			
Product Name	Category	Price	Actions
Testing	clothing	\$44.00	<input checked="" type="checkbox"/> Approve <input checked="" type="checkbox"/> Reject
car	Electronics	\$500.00	<input checked="" type="checkbox"/> Approve <input checked="" type="checkbox"/> Reject

Approved Products		
Product Name	Category	Price
Mobile	Electronics	\$5.00
laptop	Electronics	\$255.00
checking	clothing	\$21.00

Description:

This section helps the admin manage products submitted by vendors. Admins can **approve** or **reject** new product listings, ensuring only valid items are shown to customers.

CHAPTER 10

CONCLUSION AND FUTURE SCOPE

Conclusion

The Admin Panel for the E-commerce website was successfully developed using the Django framework, with a strong focus on modularity, security, and user-friendly design. The system allows administrators and sellers to manage products, view orders, and monitor user activity efficiently.

Through rigorous testing—including unit, integration, and system testing—we ensured that the platform is reliable and functions as intended. The admin panel significantly simplifies backend operations, improves data management, and supports real-time decision-making for online commerce businesses.

This project demonstrates how Django's robust features like the ORM, authentication system, and admin interface can be used to build scalable web applications.

Future Scope

- **Analytics Dashboard:** Add visual reports and graphs for sales, product trends, and user activity.
- **API Integration:** Develop RESTful APIs to integrate the admin panel with mobile apps or third-party services.
- **Email and Notification System:** Send automated email alerts to admins and users for order updates, low stock, etc.
- **Product Recommendation System:** Use AI/ML models to suggest products based on user behavior and sales trends.
- **Performance Optimization:** Improve load times and scalability using caching techniques and database indexing.

CHAPTER 12

REFERENCES

IEEE- Reference List:

1. R. Krishna, M. Jayalakshmi, and A. Kumar – Designed an admin panel using Django for better e-commerce management (2023).
2. A. Sharma and V. Patel – Explained how to manage roles and products in admin panels (2022).
3. K. Verma et al. – Proposed a secure admin panel design for modern e-commerce (2021).
4. N. Al-Mutairi and H. Al-Zahrani – Built an analytics dashboard using the MERN stack (2020).
5. B. Ramesh and K. Sinha – Created scalable admin panels using cloud microservices (2021).
6. M. Khan and L. Li – Shared best design patterns for admin dashboards (2020).
7. A. Joshi and P. Kaur – Used WebSockets for real-time order tracking in admin panels (2022).
8. D. Smith – Focused on adding analytics for better business decisions (2023).
9. L. Wang and H. Chen – Reviewed mobile app admin panel designs (2021).
10. S. Gupta and M. Jain – Used AI to automate admin alerts in online stores (2020).

Reference Format (APA Style):

11. Django Software Foundation. (n.d.). *Django documentation*. Retrieved April 13, 2025, from <https://docs.djangoproject.com/>