Industrial Internship Report

Tech Elecon Pvt Ltd.

Submitted by

AKSHIT ALPESHKUMAR MACWAN

12102080701007

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

Information Technology

Madhuben and Bhanubhai Patel Institute of Technology

The Charutar Vidya Mandal (CVM) University,

Vallabh Vidyanagar - 388120

May, 2025

Madhuben and Bhanubhai Patel Institute of Technology

Information Technology

CERTIFICATE

This is to certify that **Akshit Alpeshkumar Macwan (12102080701007)** has submitted the Industrial Internship report based on internship undergone at **Tech Elecon Pvt Ltd.** for a period of **16** weeks from **01/01/2025** to **30/04/2025** in partial fulfillment for the degree of Bachelor of Engineering in **Information Technology, Madhuben and Bhanubhai Patel Institute of Technology** at The Charutar Vidya Mandal (CVM) University, Vallabh Vidyanagar during the academic year 2024 – 25.

Prof. Jagruti Prajapati

Internal Guide

Prof. Jagruti Prajapati

Head of the Department

[Industry Letter Head]

Date: DD/MM/YYYY

TO WHOM IT MAY CONCERN

This is to certify that **Akshit Alpeshkumar Macwan**, a student of has successfully completed her internship in the field of React JS from 01/01/2025 to 30/04/2025 (Total number of Weeks: 16) under the guidance of Mr. Satyam Raval.

Her internship activities include <Internship Activities>.

During the period of her internship program with us, she had been exposed to different processes and was found diligent, hardworking, and inquisitive.

We wish her every success in his life and career.

For Tech Elecon Pvt Ltd.

Authorized Signature with Industry Stamp

DECLARATION

I, **Akshit Alpeshkumar Macwan (12102080701007)** hereby declare that the Industrial Internship report submitted in partial fulfillment for the degree of Bachelor of Engineering in Information Technology, Madhuben and Bhanubhai Patel Institute of Technology, The Charutar Vidya Mandal (CVM) University, Vallabh Vidyanagar, is a bonafide record of work carried out by me at Tech Elecon Pvt Ltd. under the supervision of Mr. Satyam Raval and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of the Student

Akshit Alpeshkumar Macwan

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Tech Elecon Pvt Ltd. for providing me with the opportunity to undergo this industrial internship. I extend my heartfelt thanks to my industry mentor, Mr. Satyam Raval, for their invaluable guidance and support throughout the 16-week internship period. Their insights and feedback have been instrumental in shaping my learning experience.

I also wish to convey my deep appreciation to my internal guide, Prof. Jagruti Prajapati, and the faculty members of Madhuben and Bhanubhai Patel Institute of Technology for their constant encouragement and advice. Their mentorship has greatly contributed to the successful completion of this report.

Finally, I would like to thank my family and friends for their unwavering support and motivation during this journey.

Akshit Alpeshkumar Macwan

ABSTRACT

The Sales Return Project is a web-based application designed to streamline the return process for online purchases. Built using React, Tailwind CSS, Vite, Node.js, and MongoDB, the system enables users to generate invoices in PDF format, validate product returns using an Order ID, and track return statuses efficiently. The backend ensures secure data storage, seamless request handling, and optimized database performance, while the frontend provides an intuitive and responsive interface for both customers and administrators. Additionally, the system incorporates error handling mechanisms, validation checks, and role-based access control to ensure secure and reliable operations. This project enhances the customer experience by simplifying return requests, reducing manual processing time, and offering real-time status updates. Future enhancements may include automated refund processing, AI-powered return analytics, chatbot support for customer queries, and third-party logistics integrations to further improve efficiency and scalability.

Through this internship, I enhanced my technical skills in React.js, API integration, and UI/UX design, while also improving my problem-solving and teamwork abilities. The experience gained during this period has been invaluable in bridging the gap between academic learning and real- world software development.

Keywords: React JS, Sales Return Management System, MERN Stack, MongoDB Atlas, APIs, Authentication.

List of Figures

List of Tables

Abbreviations

SDLC	System Development Life Cycle
MERN	MongoDB, Express.js, React.js, Node.js
MVP	Minimum Viable Product
UI	User Interface
UX	User Experience
API	Application Programming Interface
JWT	JSON Web Token
RBAC	Role-Based Access Control
PDF	Portable Document Format
CRUD	Create, Read, Update, Delete
DFD	Data Flow Diagram
ER	Entity-Relationship
MVC	Model-View-Controller
WBS	Work Breakdown Structure
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SQL	Structured Query Language

Table of Contents

CHAPTER 1
OVERVIEW OF THE COMPANY

1.1 Company Introduction

Tech Elecon Pvt. Ltd. is the It division of the Elecon group of companies and has more than 25 years of experience in the fields of hardware, software, and networking solutions.

It is situated in the heart of Vithal Udyognagar Industrial Estate and in the proximity of the educational town of Vallabh Vidyanagar.

Tech Elecon is all set to reach new heights in the field of IT solutions. Tech Elecon is ready with all sorts of solutions and delivers any application that is web based and further our solutions are designed to adapt your business rather than your business adapting the software. Their solutions are 100% fruitful and empower you to take control of client's business online and in real time.

Tech Elecon have more than 100 employees with specialized skills in software development, custom software development, and e-commerce software development using custom software programming including .NET, C#.NET, PHP, and open Source and Oracle.

Tech Elecon delivers quality products and services with a focus on integrating the same with existing technologies, providing the required automation to our customers to help them achieve their business objectives.

Mr. Nilesh Naik, the company's Vice President, is at the helm of the Tech Elecon organization. Mr. Satyam Raval, as Deputy General Manager, and after that, Manager and Associate Manager positions are listed. At the bottom, there are trainees at entry level, who follow up to engineer, senior engineer, also executive and senior executive manager.

1.2 Services and Expertise

Tech Elecon provides various services for business:

- Hardware maintenance and repairing
- Service desk management
- Desktop management
- Network management
- Messaging administrator
- Back-up management

Other services:

- Software Development Services
- Software Licensing
- Microsoft Product Implementation

1.3 Organizational Structure and Work Culture

Tech Elecon operates with a hierarchical structure, led by Vice President Mr. Nilesh Naik. Below him are the Deputy General Manager (Mr. Satyam Raval), Managers, and Associate Managers who oversee various departments and functions. Engineers, Senior Engineers, Executives, and Senior Executives handle specific tasks, while entry-level trainees learn and grow within the company.

The company fosters a collaborative, innovative, and supportive work culture. Employees are encouraged to take initiative and contribute to improving processes. There's a strong emphasis on continuous learning, with regular training programs to ensure employees stay updated on industry trends. The work environment promotes a healthy work-life balance, offering both professional growth and personal well-being.

1.4 Internship Environment and Tools Provided

During the internship, I worked on the **Sales Return Project**, a web-based application developed using **React.js**, **Tailwind CSS**, **Vite**, **Node.js**, and **MongoDB**. The project allowed me to enhance my skills in **frontend development** and **backend integration**.

Key tools provided included:

- **React.js**: For building dynamic, interactive UIs and components.
- **Tailwind CSS**: To style the application with a utility-first CSS framework for responsiveness.
- **Vite**: Used for fast and optimized development, reducing build times.
- **Node.js**: The server-side environment for handling backend operations.
- **MongoDB**: For secure and efficient data storage, using MongoDB Atlas for cloud deployment.
- **APIs**: Used for handling product return validation and order status tracking.
- **Authentication**: Implemented for role-based access control and secure user authentication.

The internship also provided exposure to **UI/UX design principles**, **error handling**, and **API integration**. Working with these tools allowed me to contribute to a fully functional, scalable application, and helped me develop both technical and teamwork skills, bridging the gap between academic learning and real-world software development.

CHAPTER 2
INTRODUCTION TO THE PROJECT

2.1 Project Overview

The Sales Return Management System is a web-based solution designed to streamline and automate the process of handling product returns in e-commerce businesses. Efficient return management is crucial for maintaining customer satisfaction, improving operational efficiency, and reducing financial losses. This system allows customers to initiate return requests, validate product eligibility using an Order ID, generate invoices in PDF format, and track return statuses in real time.

Built using React, Tailwind CSS, Vite, Node.js, and MongoDB, the system ensures a seamless and secure return process for both customers and administrators. The backend efficiently manages return requests, secure data storage, and real-time processing, while the frontend provides an intuitive and user-friendly interface.

2.2 Objective of the Project

Automate the Return Process – Provide a seamless and efficient system for customers to request product returns with minimal manual intervention.

Order Validation & Invoice Generation – Ensure secure return processing by validating Order IDs and generating invoices in PDF format for proper documentation.

Real-Time Status Tracking – Enable customers and administrators to track return requests in real time, improving transparency and customer satisfaction.

Secure Data Management & Scalability – Utilize MongoDB and Node.js for secure data storage, with future scalability for automated refunds and third-party integrations.

2.3 Problem Statement

The **Sales Return Management System** addresses the challenges faced by e-commerce businesses in managing product returns efficiently. Manual return processes are often slow, error-prone, and inefficient, leading to customer dissatisfaction, operational delays, and financial losses. There is a need for an automated, secure, and user-friendly system that simplifies return requests, ensures accurate product eligibility checks, generates invoices for documentation, and provides real-time tracking of return statuses. This system aims to reduce the time spent on manual processing, improve transparency, and enhance customer experience during product returns.

2.4 Scope of the Project

The scope of the **Sales Return Management System** includes:

- **Return Request Automation**: Automating the process for customers to submit return requests, reducing manual effort and errors.
- **Order Validation**: Ensuring that returns are only processed for eligible products based on Order ID validation.
- **Invoice Generation**: Automatically generating PDF invoices for each return request for proper documentation and record-keeping.
- **Real-Time Tracking**: Providing customers and administrators with real-time updates on the status of return requests.
- **Admin Functionality**: Enabling administrators to manage return requests, update product information, and approve or reject returns based on predefined criteria.
- **Security & Scalability**: Ensuring secure data storage, role-based access control, and future scalability to accommodate features like automated refunds and third-party integrations.

2.5 Tools and Technologies Used

The **Sales Return Management System** is built using the following technologies:

- **React.js**: For building the dynamic, interactive user interface.
- **Tailwind CSS**: A utility-first CSS framework used for fast, responsive styling.
- **Vite**: A build tool that provides fast development and optimized builds.
- **Node.js**: For handling backend server-side logic and processing requests.
- **MongoDB**: A NoSQL database used for secure and scalable data storage, utilizing **MongoDB Atlas** for cloud deployment.
- **APIs**: For integrating different functionalities such as order validation, return processing, and status tracking.
- **PDF Generation**: Tools or libraries (like **jsPDF**) for generating return invoices in PDF format.
- **Authentication**: Role-based access control using **JWT (JSON Web Tokens)** for secure user authentication.

CHAPTER 3
SDLC APPROACH

3.1 Introduction to SDLC

The **System Development Life Cycle (SDLC)** is a systematic methodology used for planning, creating, testing, and deploying information systems. It ensures a structured and quality-driven development process that meets user requirements within time and resource constraints.

The **Sales Return Management System** followed the **SDLC** process, starting with requirement gathering, design using **React.js** and **Node.js**, and development with the **MERN stack**. After testing, the system was deployed and continuously improved based on user feedback to ensure scalability, security, and reliability.

3.2 SDLC Model Followed

For the **Sales Return Management System** project, a **Modified Waterfall model** was adopted. While the development phases followed a mostly sequential order, feedback loops were included to allow for minor refinements after reviews or testing. This hybrid approach provided the necessary structure while accommodating the flexibility needed in real-world development. The process followed the Modified Waterfall model as recommended by Pressman [1], ensuring that each phase, from requirements gathering to deployment, was carefully managed with room for adjustments based on ongoing feedback.

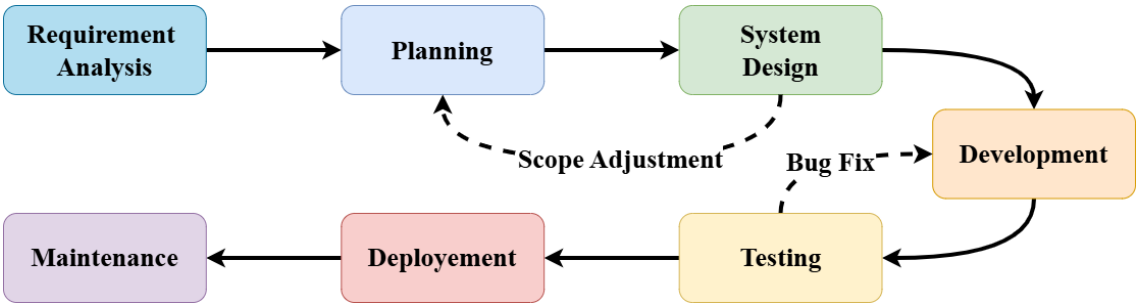

Fig 3.1 Modified Waterfall Model

3.3 SDLC Phases Followed in the HRMS Project

3.3.1 Requirement Analysis

- Conducted requirement gathering to understand the needs of two user roles:
    1. **Customer**
    2. **Administrator**

- Identified and documented functional and non-functional requirements, focusing on the return process, order validation, and real-time tracking.
- Analyzed the current challenges in managing product returns for e-commerce businesses, including the need for a streamlined process and efficient invoice generation.

3.3.2 Planning

- Defined project scope, deliverables, and priorities based on the return management use cases.
- Created a task list aligned with the internship timeline, ensuring key milestones were met.
- Designed a Gantt chart for milestone tracking to manage time efficiently.
- Identified risks such as integration issues and delay in testing, and created mitigation strategies.

3.3.3 System Design

- Designed the database schema, ensuring proper relationships between orders, returns, and user roles (Customer and Administrator).
- Prepared UI wireframes using tools like Figma to visualize the customer and admin interfaces.
- Defined system architecture, focusing on a modular design that supports scalability and ease of maintenance.

3.3.4 Development

- Developed the system using the **MERN stack** (MongoDB, Express.js, React.js, Node.js) for a robust, full-stack application.
- Implemented:
    - **Order ID validation** for return eligibility
    - **PDF invoice generation** for return requests
    - **Real-time return status tracking**
    - **Admin dashboard** for managing returns and viewing return requests
- Managed versions and updates using **Git/GitHub** for collaborative development.

3.3.5 Testing

- Conducted unit testing on individual features like order validation and invoice generation.
- Performed integration testing to ensure smooth coordination between frontend and backend.
- Debugged and reviewed code under mentor guidance to address bugs and optimize performance.

3.3.6 Deployment

- Deployed the system in a local test environment simulating real-world conditions.
- Conducted UI testing to ensure all pages functioned correctly across roles (customer and administrator).
- Ensured the system was **mobile responsive** and provided a smooth user experience across devices.

3.3.7 Maintenance

- Post-deployment feedback was gathered from both customers and administrators to refine the user interface and improve workflow transitions.
- Added missing validation checks and refined error messages for better clarity.
- Regular backups and data integrity checks were set up to ensure system reliability and security.

3.4 Justification for Using SDLC

The SDLC framework ensured that the project was completed within the internship timeline while maintaining quality. Key advantages observed:

- Phase-wise clarity and ownership
- Reduced risk of overlooked features
- Easy tracking of bugs and scope changes

CHAPTER 4
REQUIREMENT ANALYSIS

4.1 Introduction

The Requirement Analysis phase is crucial for the success of any software project. It defines what the system must achieve based on user needs, business requirements, and system constraints. For the **Sales Return Management System**, this phase involved gathering both functional and non-functional requirements from stakeholders, analyzing the existing processes for handling returns in e-commerce businesses, and identifying areas for improvement.

4.2 Requirement Gathering Method Used

Since I was the sole developer working closely with my project mentor, I employed the **Stakeholder Interview** method to gather the necessary requirements effectively.

4.2.1 Stakeholder Interview

- A series of informal discussions were held with the mentor, who acted as the client representative for the project.
- Questions were focused on understanding the current pain points in handling product returns, user expectations, and essential features needed in the return management system.
- Key insights gathered included:
    - The need for a **return request** submission process with order ID validation.
    - **PDF invoice generation** for return documentation.
    - A **role-based system** for customer and admin interactions.
    - The ability to **track return statuses** in real-time for both customers and admins.
- These discussions helped prioritize features and define the scope of the system.

4.3 Functional Requirements

The **Sales Return Management System** supports two primary user roles:
A. Customer

- Initiate return requests by providing Order ID.
- View the eligibility of returned products based on predefined conditions.
- Track the status of the return request.
- Download a PDF invoice for the return request.

B. Administrator

- Manage return requests, approve or reject returns.
- View and update return request statuses.
- Maintain product catalog and return conditions.
- Generate reports for returns and refunds.

Table 4.1 Functionality Table

Functionality	Customer	Administrator
View Profile	☑	☑
Submit Return Request	☑	✗
Validate Return	✗	☑
Track Return Status	☑	☑
Generate PDF Invoice	☑	✗
Manage Return Requests	✗	☑
Product Management	✗	☑

4.4 Non-Functional Requirements

- **Security:** Implement role-based access control to restrict unauthorized actions, and ensure data encryption for sensitive information.
- **Usability:** The UI must be intuitive and responsive, providing clear feedback messages for successful or failed actions.
- **Performance:** The system should support fast data retrieval, especially for order validation and real-time tracking updates.
- **Scalability:** The system should be scalable to handle increasing numbers of return requests and product listings as the business grows.
- **Maintainability:** The system should follow a modular design for easy updates and bug fixes.
- **Data Integrity:** Data must be accurately stored in a secure and consistent manner for reporting and auditing purposes.

4.5 System Constraints

- The system was developed using:
    - **Backend**: Node.js with Express
    - **Database**: MongoDB

  - **Frontend**: React.js, Tailwind CSS
- The project needed to be completed within the internship duration (16 weeks).
- As a single intern, all development, testing, and deployment were managed by me.

4.6 Assumptions

- Customers will always provide accurate Order IDs for return validation.
- Administrators are responsible for managing the product catalog and approving or rejecting return requests.
- The system will be accessed through secure login credentials for both customers and administrators.

CHAPTER 5
PLANNING AND TIMELINE

5.1 Introduction

The **Planning Phase** is critical for defining the scope, deliverables, timeline, and resource allocation to ensure the successful execution of the project. For the **Sales Return Management System**, which was developed during my 16-week internship, effective time management and milestone planning were essential to ensure that all core modules were completed on schedule.

5.2 Project Objectives

The goals for the planning phase were:

- Finalize the features and requirements identified in the **Requirement Analysis** phase.
- Break the project into realistic tasks and milestones.
- Allocate time and effort per module based on its complexity and priority.
- Identify potential risks and prepare mitigation strategies.

5.3 Work Breakdown Structure (WBS)

The project was divided into phases aligned with the SDLC stages. Below is the high-level breakdown:

Table 5.1 Phases of Project

Phase	Key Deliverables
Requirement Analysis	Finalized functional and non-functional requirements
Planning	Timeline, milestones, risk assessment
Design	Database schema, UI wireframes, system architecture
Development	Core modules: Return Request, Order Validation, Invoice Generation, Real-Time Status Tracking
Testing	Unit testing, bug resolution, API and UI testing
Deployment	Local deployment with demo-ready features
Maintenance	Bug fixes, UI refinements, and enhancements
Evaluation	Final review and feedback from mentor and stakeholders

5.4 Gantt Chart (Timeline)

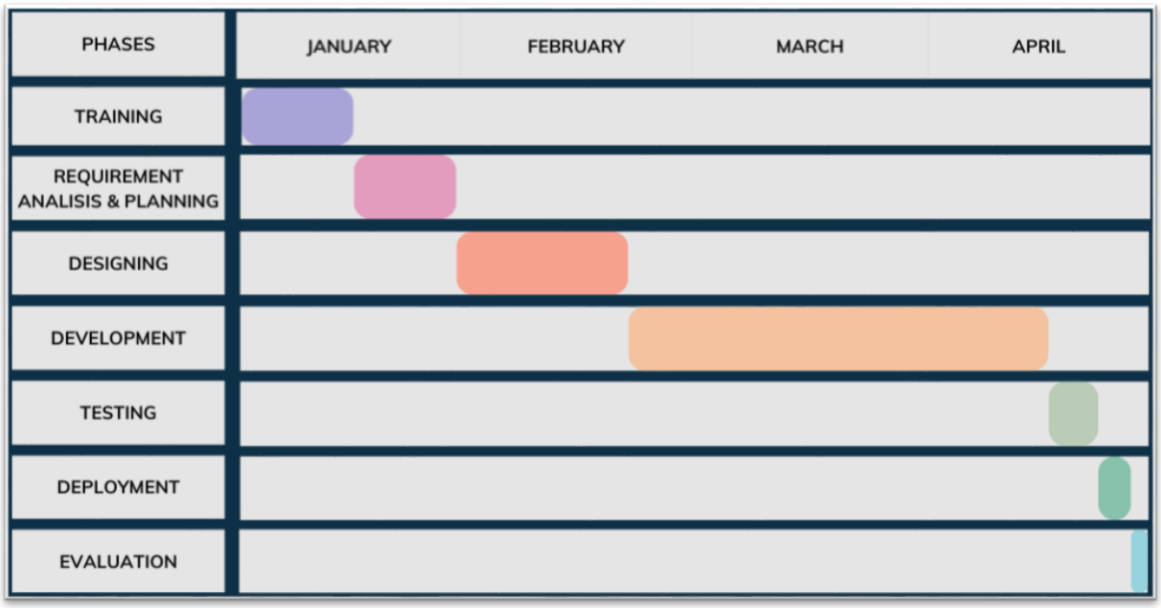

Fig 5.1 Project Gantt Chart

5.5 Tools Used for Planning

• **Gantt Chart**: Created using **Google Sheets** for timeline visualization.

• **Wireframes**: Designed using **Figma** for UI planning and layout.

• **Project Notes**: Documented and tracked using **Google Docs** for task organization and planning.

5.6 Risk Assessment

Table 5.2 Risk Assessment Table

Risk	Impact	Mitigation Strategy
Time constraint for feature delivery	High	Prioritize MVP features (e.g., return request, invoice generation) and establish weekly review checkpoints.
Working solo	Medium	Use structured task lists, modular coding for easier updates and quick changes.
Scope creep	Medium	Finalize requirements early and freeze them for the MVP version. Any new features will be considered post-MVP.
Integration bugs	High	Frequent testing, especially after integrating the frontend and backend; utilize version control with **Git/GitHub**.

5.7 Outcome of Planning Phase

- A clearly defined project roadmap and delivery schedule.
- Milestones aligned with the internship duration.
- Core modules for MVP: Return request submission, order validation, real-time tracking, and PDF invoice generation.
- A realistic assessment of risks and actionable mitigation strategies.

CHAPTER 6
SYSTEM DESIGN

6.1 Introduction

The system design of the **Sales Return Management System** focuses on creating an intuitive, secure, and scalable web-based platform that enhances the product return process in e-commerce. The design integrates smooth user interactions, secure transaction processing, and efficient management tools for both customers and administrators. By leveraging modern web technologies such as React.js, Node.js, and MongoDB, the portal aims to provide a seamless, responsive, and user-friendly interface. The system is designed to be scalable, allowing future features such as automated refunds, AI analytics, and third-party logistics integrations to be easily incorporated. The architecture focuses on ensuring high performance and flexibility across various devices, allowing the system to meet evolving business needs and user expectations.

Key system design components include:

- **DB (Database):** MongoDB is used to store and manage data related to orders, returns, products, and customer profiles.
- **DFD (Data Flow Diagram):** Visual representation of data movement through the system.
- **ER Diagram (Entity Relationship Diagram):** A model showing the relationships between entities in the system.
- **MVC (Model-View-Controller):** This design pattern is adopted for clean separation of concerns, enabling scalable and maintainable code.

6.2 System Design

A. Context Diagram

The **Context Diagram** (or Level-0 Data Flow Diagram) outlines the overall flow of information within the **Sales Return Management System**. It helps in defining the boundaries of the system by illustrating how external entities interact with the system. These entities include:

- **Customers:** Initiate return requests and track return statuses.
- **Admins:** Oversee return approvals, manage product data, and generate reports.
- **Payment Gateway:** Used for handling transactions related to return processing or refunds.
- **Notification Service:** Sends alerts to customers and admins about return statuses.

This context diagram helps in visualizing the major interactions and serves as a high-level abstraction of the system's scope.

Fig 6.1 Context diagram

B. - ER–Diagram

The **Entity-Relationship (ER) Diagram** illustrates how the different entities within the system relate to each other. Key entities in the **Sales Return Management System** include:

- **Customer:** Contains information about the customer, such as name, email, and address.
- **Return Request:** Represents a customer's return request, including the return reason, status, and associated order ID.
- **Order:** Links return requests to specific orders made by customers.
- **Product:** Represents the product being returned, including details like product name, SKU, and category.
- **Admin:** An entity that represents administrators, who approve/reject return requests and manage products.

The **ER Diagram** defines the relationships between these entities, such as the relationship between an order and the return request or between a product and the return request.

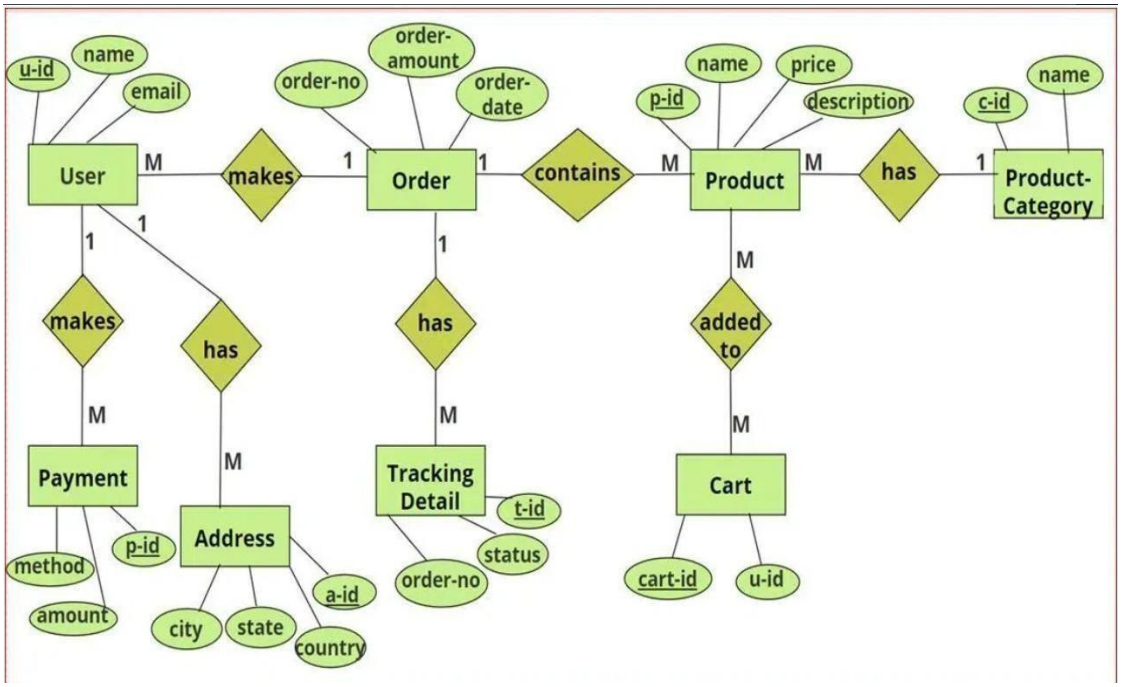

Fig 6.2 ER-Diagram

C. Level 1 DFD:

This diagram illustrates the high-level flow of data within the Sales Return Management System. Key processes include:

- **Initiating Return Request:** The customer submits a return request with the order ID, triggering validation.
- **Validate Return:** The system verifies the order ID and eligibility for return.
- **Return Approval:** The admin approves or rejects the return based on the product and return criteria.
- **Generate Invoice:** Once a return is approved, a PDF invoice is generated**.**
- **Track Status:** The system updates and provides real-time tracking of return statuses to the customer.

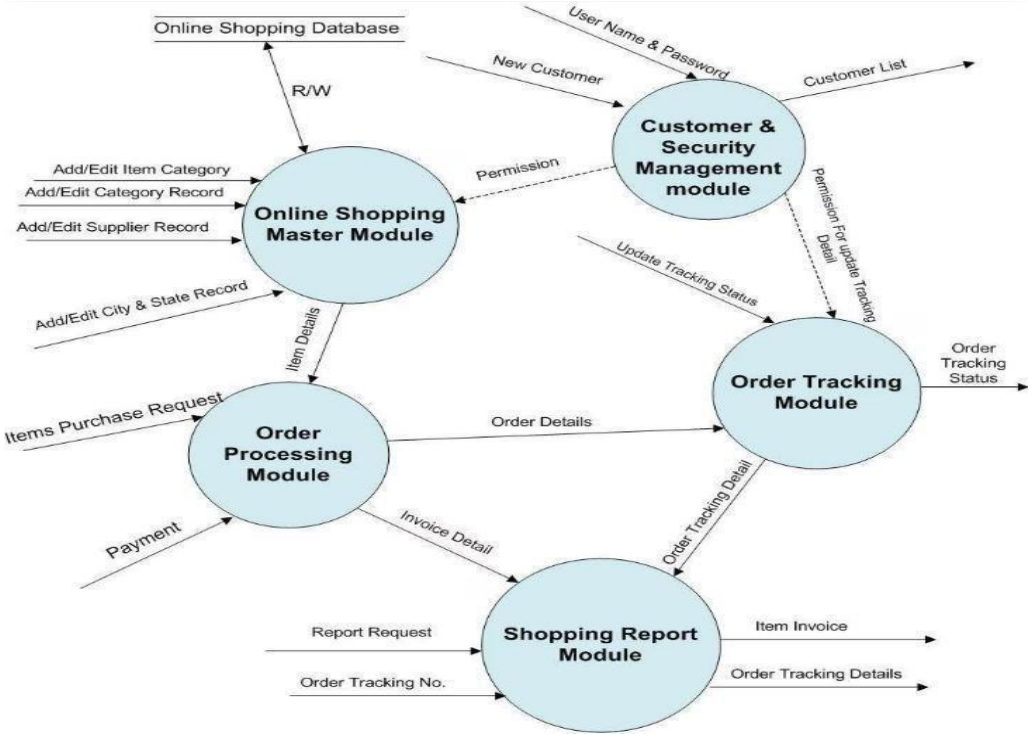

Fig 6.3 Level 1 DFD

C. Level 2 DFD for Admin

This diagram focuses on the processes involved from the admin's perspective, such as:

- **Return Management:** Admins manage the return requests by viewing, approving, or rejecting them.
- **Product Management:** Admins have the ability to add, update, or remove products in the system.
- **Report Generation:** Admins can generate reports on return status, processing times, and refund statistics for better decision-making.

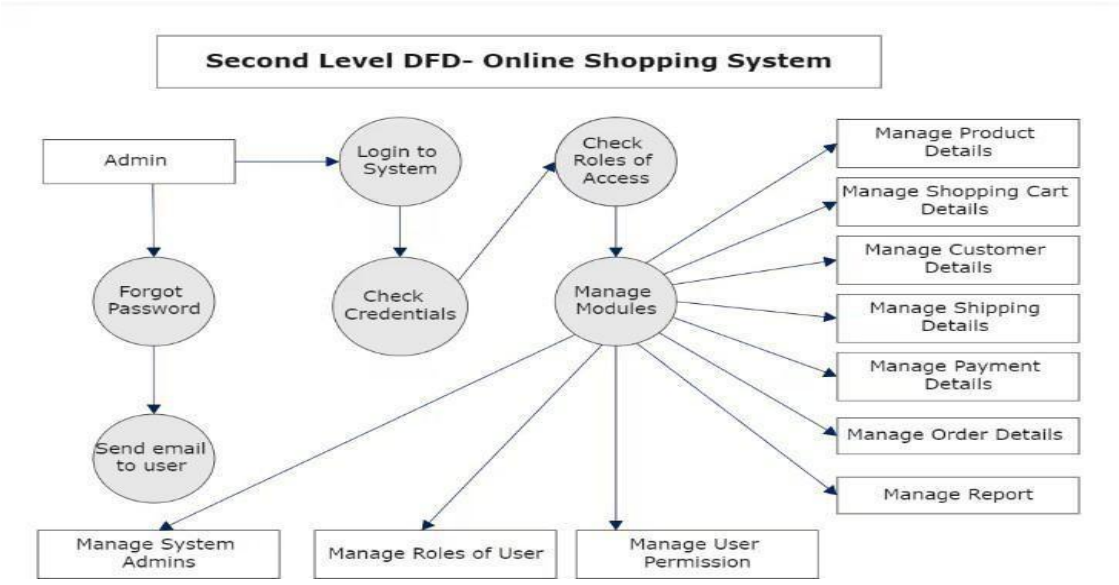

Fig 6.4 Level 2 DFD

D. USE CASE Diagram:

The **Use Case Diagram** illustrates the interactions between different types of users and the system. It highlights the system's focus on user-friendly interactions and efficient processes for both customers and admins.

(A) Admin:

The Admin role in the system is responsible for overseeing the entire return process. Key use cases include:

- **Manage Return Requests:** Admins review return requests, approve/reject them, and update statuses.
- **View Return Reports:** Admins can generate reports related to return history and statistics.
- **Manage Products:** Admins add or remove products from the system and update product details.
- **Generate PDF Invoices:** After a return is approved, admins can generate PDF invoices for customers.

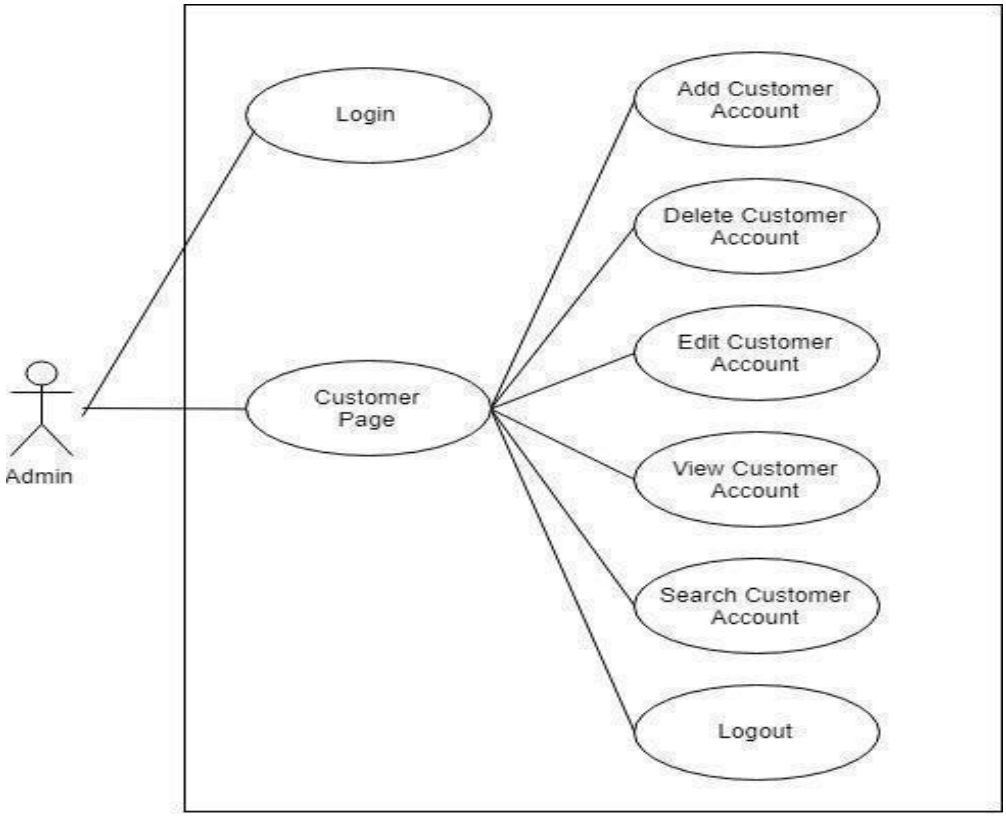

Fig 6.5 Use Case for Admin

(B) User:

The User role is responsible for submitting return requests and tracking their return statuses. Key use cases include:

- **Initiate Return Request:** User submit a return request by entering the order ID and selecting a return reason.

- **Track Return Status:** User can track the status of their return request in real time.

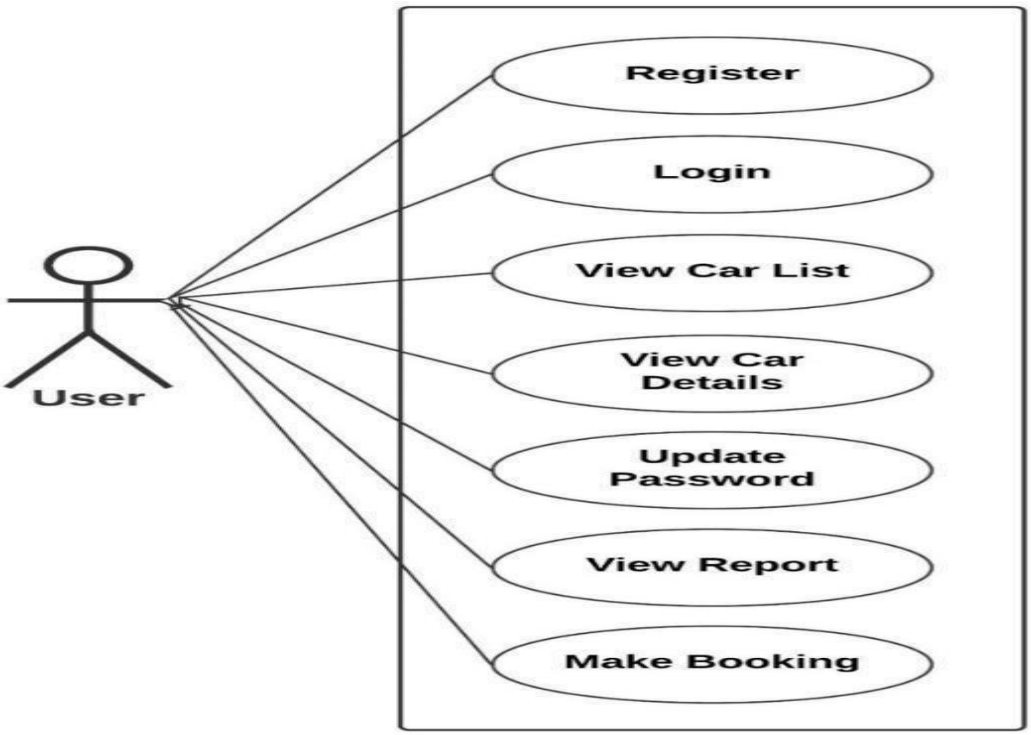

Fig 6.6 Use Case for User

CHAPTER 7
IMPLEMENTATION

7.1 Frontend Development

Key design principles and practices followed during the development include:

1. **Component-Based Development:** The entire frontend architecture was based on **component-driven development**, following the core principles of **React.js**. Every part of the user interface — from buttons, modals, and product cards to complex forms and dashboards — was broken down into reusable components. This modular design facilitated easier maintenance, quicker debugging, and faster scalability of the application. The project was scaffolded using **Vite**, a fast modern frontend build tool, which dramatically improved development speed and Hot Module Replacement (HMR) performance.

   Furthermore, **Tailwind CSS** was used extensively to style these components. Tailwind's utility-first classes enabled rapid UI development while ensuring that designs remained consistent across the entire application. The combination of React, Vite, and Tailwind CSS resulted in a powerful, responsive, and seamless user experience.

2. **State Management:** State management across the application was handled efficiently using the **React Context API**, eliminating the complexities of prop drilling and minimizing the chances of data inconsistencies. Global states such as user authentication status, cart contents, and return request statuses were managed through centralized context providers.

   Additionally, **custom hooks** were developed to encapsulate and reuse complex logic, ensuring clean and maintainable code. This approach not only optimized the performance of the application by preventing unnecessary re-renders but also provided a scalable solution to grow the application's complexity in the future.

3. **User Experience Optimization:** Creating a smooth and intuitive user experience

was a top priority. Tailwind CSS made it easy to maintain a **mobile-first**, clean, and modern layout across the platform. Special attention was given to:

- **Button placements** for ease of use
- **Accessible forms** with proper labels and error messages
- **Loading spinners** and **toast notifications** for user feedback
- **Confirmation modals** for critical actions like deleting or submitting returns

By applying **UX heuristics** and **best practices**, we ensured that customers could easily initiate return requests, track their orders, view invoices, and manage their profiles.

4. **Responsive Design:** The frontend was meticulously designed with responsiveness in mind. Adopting a **mobile-first design philosophy**, the application was optimized for various devices, including smartphones, tablets, laptops, and desktops.
   - **Flexible grid layouts**,
   - **Tailwind's responsive utility classes**, and
   - **Media queries**

   were heavily utilized to achieve fluid responsiveness. Whether users accessed the system via a large monitor or a small phone screen, the experience remained consistent, intuitive, and visually pleasing.

5. **Animations and Transitions:** To further enhance user engagement, **Framer Motion**, a powerful animation library for React, was integrated into the project. Key features included:
   - Smooth **page transitions** between routes
   - Elegant **modal opening/closing animations**
   - Subtle **hover effects** on buttons and product cards
   - **Loading indicators** and micro-interactions to guide users through processes

These animations not only made the platform visually appealing but also improved the perceived performance of the application.

6. **API Communication:** Efficient client-server communication was achieved through **Axios**, a popular HTTP client. Key API interactions included:
    - **Fetching product listings**
    - **Submitting return requests**
    - **Tracking order statuses**
    - **User registration and authentication**
    - **Managing cart and invoice downloads**

    Axios interceptors were used for handling **request authentication tokens**, **automatic retries** on failure, and **global error handling**, ensuring a seamless communication layer between frontend and backend.

- **Main Page:** Main Page Preview of Our Website

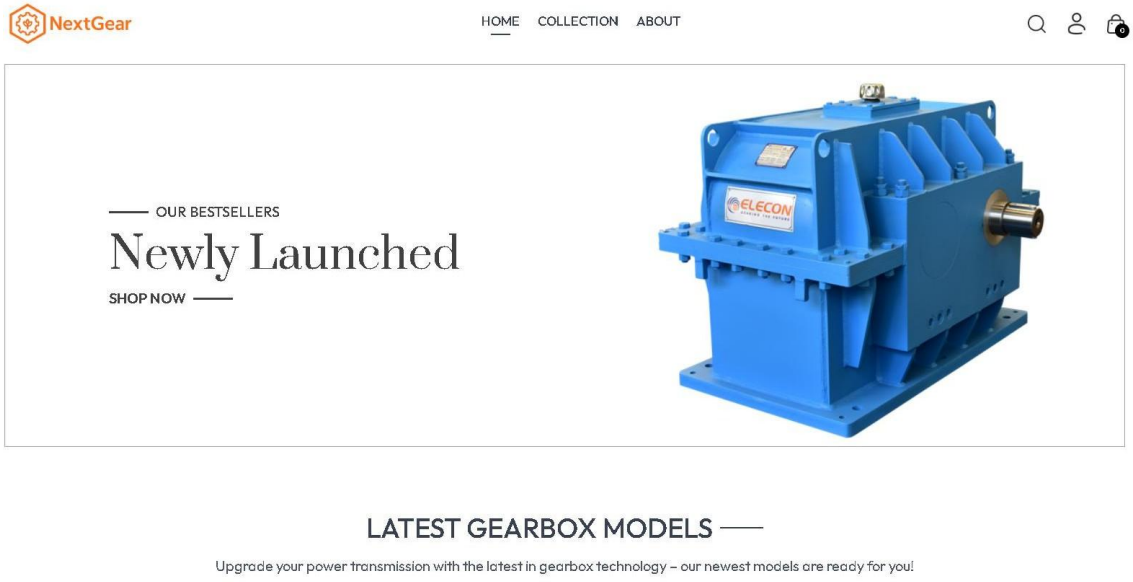

Fig: 7.1 Main Page

- **Product Collections:** Product Listings on Our E-Commerce Website

Fig: 7.2 Product Collection

➢ **Product-Categories:** Wide Range of Product Categories

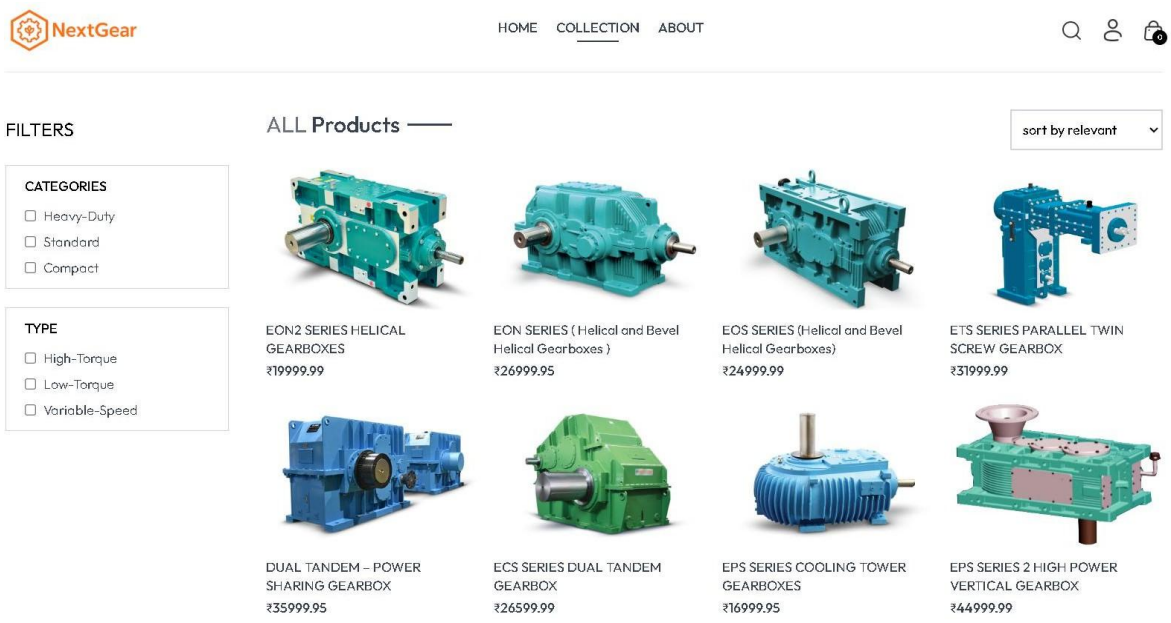

Fig: 7.3 Products categories

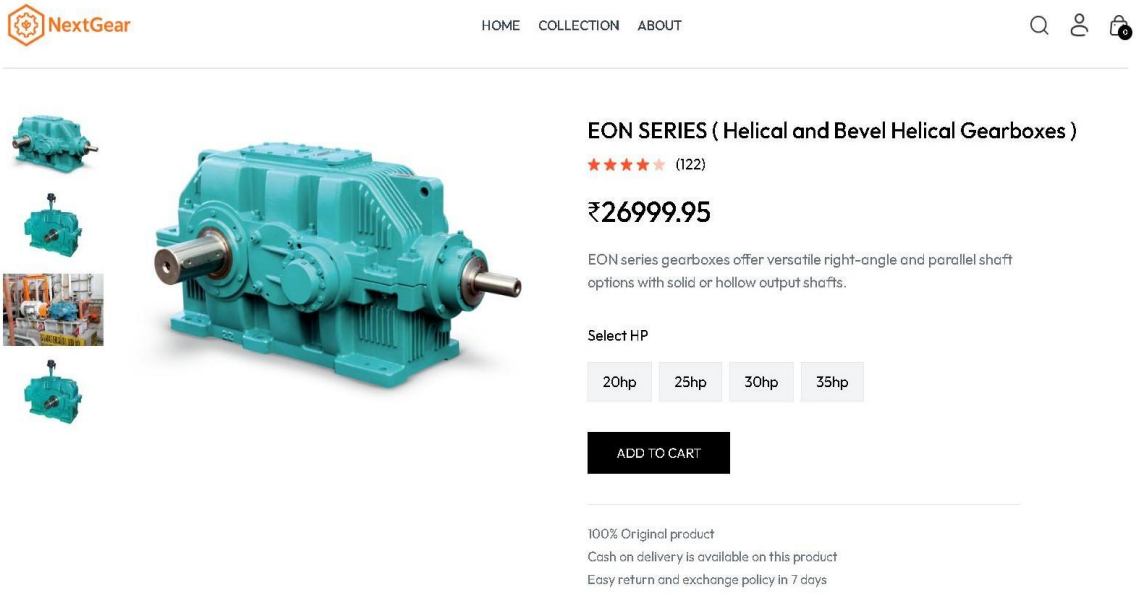

Fig: 7.4 Product Detail page

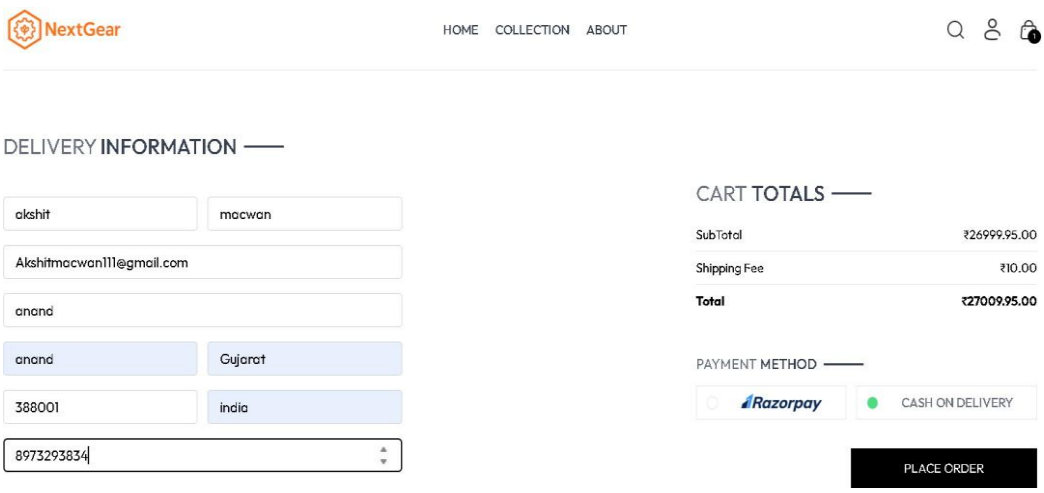

Fig: 7.5 Cart and Delivery Information Page

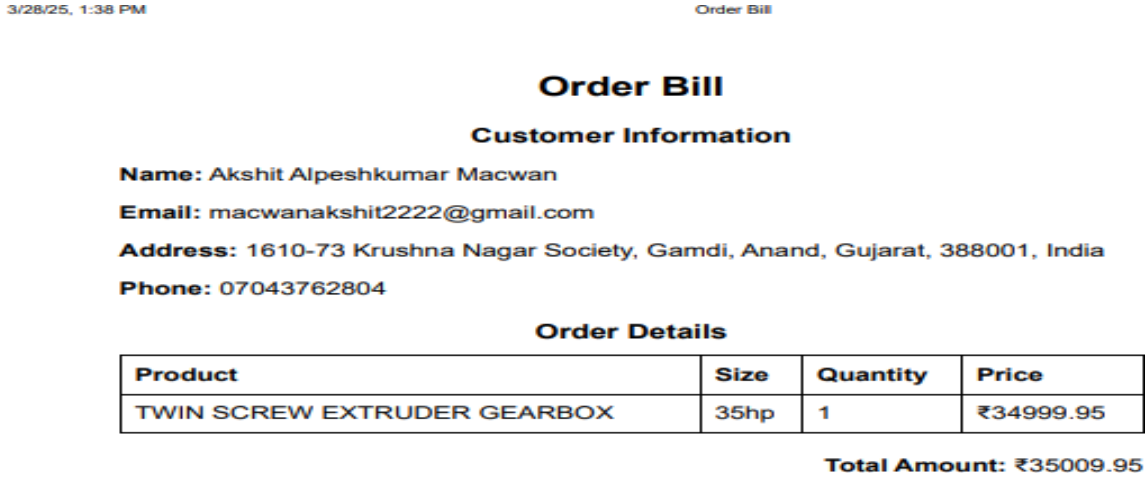

Fig: 7.6 Product bill

7.2 Backend Development

The The backend development for the **Sales Return Management System** emphasized security, efficiency, and scalability. Built using **Node.js** and **Express.js**, with **MongoDB Atlas** as the database, the backend was structured around a **RESTful API** approach, ensuring standardized and maintainable data handling.

Key Architectural Elements

a. Order Processing and Return Management

The system used **MongoDB Atlas** — a highly available, scalable NoSQL cloud database — to store and manage critical business data such as orders, returns, user profiles, and product inventories.

- When an order was placed, **server-side validations** were performed to ensure data integrity before persisting it into MongoDB.
- Users could initiate **return requests** by referencing their **Order IDs**, after which the backend verified the request based on order status, product condition, and return eligibility rules.
- Once validated, the return status was updated in real-time, and necessary notifications were triggered to inform users and administrators.

b. Invoice Generation and Management

An essential feature of the system was the automatic **generation of PDF invoices**.

- Upon successful order placement, a **PDF invoice** was created using libraries like **PDFKit** or **html-pdf**.
- The invoice included critical information such as buyer details, purchased items, pricing, taxes, and shipping information.
- Generated invoices were stored securely and made accessible for download via a unique URL on the frontend.

c. RESTful API Development

Using **Express.js**, a series of well-structured API endpoints were developed to manage all core operations:

- **User Authentication**: Registration, login, and token-based session management.
- **Product Management**: CRUD operations for managing product listings.
- **Order Management**: Creating new orders, retrieving order history, initiating

returns.

- **Return Management**: Handling and tracking return requests.

All APIs returned data in **standardized JSON** formats, enabling seamless integration with the frontend and future third-party systems if required.

d. Middleware Implementation

Custom middleware functions enhanced the system's maintainability and security:

- **Authentication Middleware**: Protected sensitive routes using JWT tokens.
- **Validation Middleware**: Ensured incoming request data met specified formats and business rules.
- **Error Logging Middleware**: Captured and logged API request data, helping in monitoring system usage and debugging anomalies.

These middleware layers created a clean separation of concerns, improved code reusability, and strengthened the application's security posture.

e. Error Handling

The system adopted **centralized error handling** mechanisms:

- Errors were caught through **try-catch** blocks and funnelled into a centralized error handler.
- Custom error classes provided clear and consistent error messages for frontend consumption.
- Proper HTTP status codes (e.g., 400, 401, 404, 500) were returned with all API responses to aid in debugging and enhance client-server communication reliability.

f. Data Validation

Robust data validation was enforced both at the API entry point and the database level:

- Using libraries like **Joi** or **express-validator**, incoming payloads were validated against pre-defined schemas before processing.
- Sensitive user inputs (e.g., email addresses, phone numbers, addresses) were sanitized to prevent injection attacks or malformed data entry.

7.3 Database Design

A robust and scalable database design was critical for ensuring the **Sales Return Management System** could handle real-world scenarios efficiently, securely, and reliably. The backend infrastructure was powered by **MongoDB**, a flexible, NoSQL database well-suited for document-based storage needs. In this system, multiple collections were designed carefully to optimize data retrieval, maintain security standards, and ensure overall data integrity.

a. Users Collection

The **Users** collection plays a pivotal role in the system's architecture, serving as the foundation for managing customer and administrator data. Special focus was given to security, scalability, and ease of integration with authentication mechanisms.

Key Features:

- **User Information Storage:**

  Each document within the Users collection holds critical user details such as:

  - **Username:** A unique identifier chosen by the user.
  - **Email Address:** Used for communication, notifications, and authentication.
  - **Password:** Stored in an **encrypted** format to enhance security and prevent unauthorized access.

- **Password Encryption:**

  Passwords are **never stored in plain text**. Instead, they are encrypted using strong hashing algorithms like **bcrypt** before being saved to the database. This approach protects user credentials even in the event of a data breach.

- **Role-Based Access Control (RBAC):**

  The system employs a **role** field in the user documents to enforce **Role-Based Access Control**. Roles such as:

  - **Admin** (privileged access)
  - **User** (standard access)

  are assigned upon account creation or during administrative management. RBAC allows the system to securely differentiate functionalities, ensuring that sensitive operations are restricted to authorized personnel only.

- **JWT Token Authentication:**

  Upon successful login, users are issued **JWT (JSON Web Tokens)** that securely

authenticate them across the system without repeatedly sending credentials. Each JWT contains:

- A unique user ID
- Assigned role
- Expiration timestamp

Tokens are verified against the server for every protected route, maintaining a seamless and secure user session.

Fig 7.7 User login credentials with encrypted password-backend

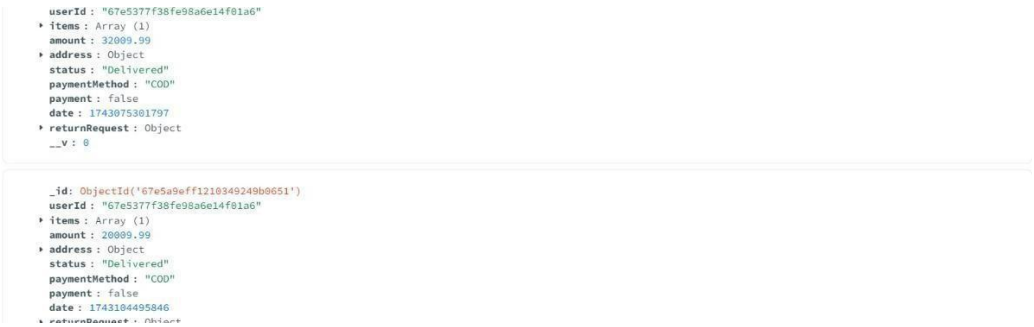

Fig 7.8 User login credentials with encrypted password-backend

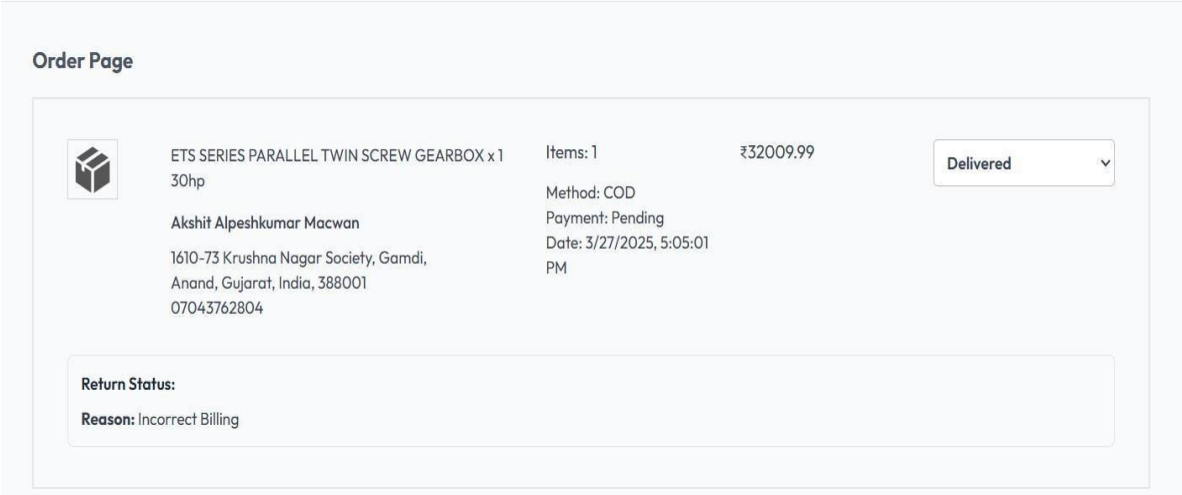

Fig 7.9 User management page front-end

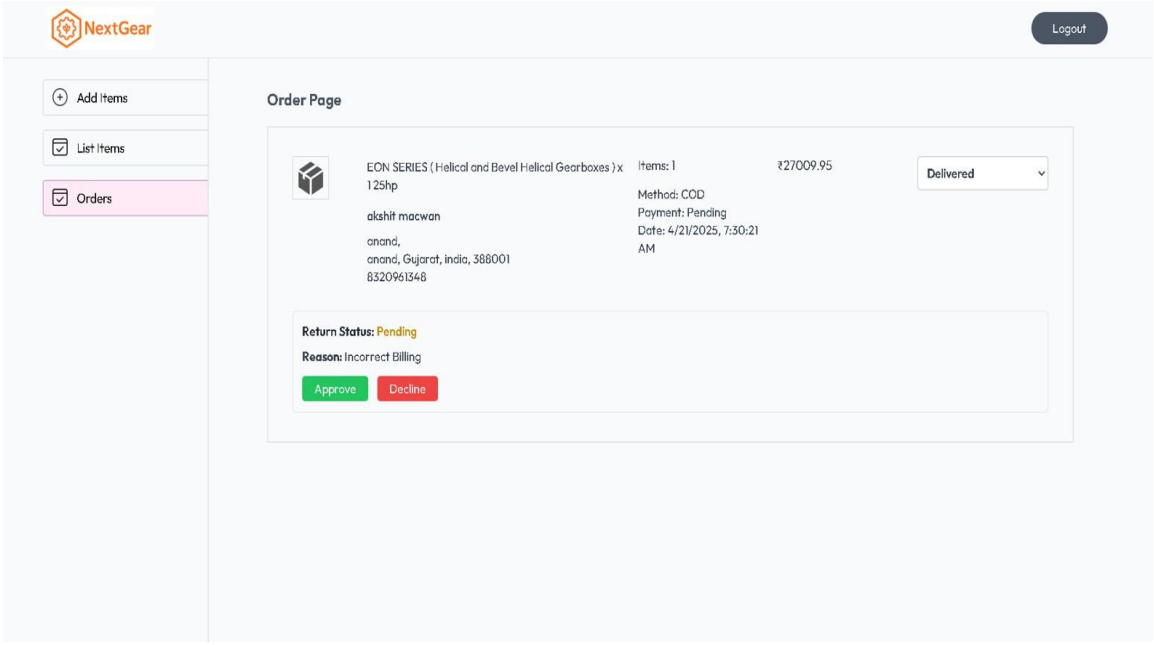

Fig 7.10 Admin panel

CHAPTER 8
TESTING

8.1 Objective of Testing

The primary objective of testing the **Sales Return Management System** was to verify that all developed modules functioned accurately according to the documented requirements. Testing aimed to confirm that the system behaved consistently across different user roles, maintained data integrity, and provided a secure, seamless user experience. Another major goal was to identify bugs, inconsistencies, performance issues, or incomplete functionalities at early stages, ensuring the system would be robust and ready for deployment.

A **black-box testing** approach was employed, focusing on verifying inputs and outputs without delving into the internal code structure [1]. This ensured that the system was tested purely from a user's perspective, closely aligning with real-world usage scenarios.

8.2 Types of Testing Performed

Throughout the development lifecycle, multiple testing techniques were applied systematically:

1. Unit Testing

- Each module (Login, Sales Return Request, Product Management, Admin Dashboard) was tested individually in isolation.
- Functions were validated using both valid and invalid inputs to ensure proper handling of edge cases and error scenarios.

2. Integration Testing

- Verified seamless interaction between modules (e.g., login → sales return form → admin approval workflow).
- Ensured session management was persistent, and user state transitions were smooth across screens.

3. UI/UX Testing

- Checked modal visibility, field focus, button activation, and responsive design across different screen sizes.
- Confirmed consistent styling, color schemes, font usage, and overall design uniformity using the standard template.

4. Database Testing

- Direct SQL queries were executed in **MongoDB** (using Compass and shell) to validate CRUD operations.

- Confirmed that document relationships (e.g., User → Sales Return) were maintained without violating referential integrity.
- Ensured soft delete operations properly flagged entries without breaking dependent modules.

5. Role-Based Access Testing

- Thorough testing was conducted across different user roles (Admin, Registered User).
- Verified role-based visibility and access control to specific pages and actions (e.g., only Admins can approve/reject returns).

8.3 Testing Strategy

A **manual testing** approach was adopted, focusing on real-time feedback and immediate bug resolution. Features were developed incrementally, tested locally at http://localhost:3000, and iteratively improved.
Each test cycle involved submitting forms with a variety of input types (correct, missing, and invalid) to evaluate validation logic, error handling, and feedback responsiveness.

Table 8.1 Testing Covered

Category	Covered Elements
Form Validation	Required field checks, email and numeric format validation, minimum/maximum length validations
Session Management	JWT token persistence after login, auto-logout on token expiry, secure page access after refresh
Modal Behaviour	Modal open/close animations, data binding inside modals, error and success feedback on actions
Export Accuracy	PDF/Excel export matches on-screen filtered data, correct column alignment and formatting
Toast Feedback	Display appropriate success, error, and warning messages for user actions
Sales Return Logic	Correct creation of sales return requests, prevention of duplicate requests, validation of product condition selection
Admin Approval Logic	Admin can approve/reject returns, real-time status updates, error handling if action fails

8.4 Sample Test Case Format

Table 8.2 Sample Test Case Format

Test Case	Input	Expected Result	Actual Result	Status
Login with valid Admin credentials	Email, Password	Redirect to Admin Dashboard	Success	✔ □
Submit new sales return request	Order ID, Reason, Product Condition	Sales return request saved and listed	Success	✔ □
Approve a pending return	Click "Approve"	Status updated to Approved, toast shown	Success	✔ □
Export returns filtered by date range	Date From, Date To	PDF/Excel shows matching entries only	Success	✔ □
Try submitting without reason field	Empty reason	Validation error displayed, form blocked	Error handled	✔ □

8.5 Result Summary

The testing phase confirmed that the **Sales Return Management System** was stable, secure, and ready for production deployment. All major functional paths—including authentication, sales return submission, admin approval/rejection workflows, and report generation—passed the validation process successfully.

Minor bugs discovered during early module development (such as missing input validations, inconsistent toast messages, or minor UI glitches) were promptly fixed through iterative development and retesting cycles. The system’s performance, usability, and data integrity standards met or exceeded the initial requirements, achieving the project's objectives effectively.

CHAPTER 9
DEPLOYMENT AND MAINTENANCE

9.1 Local Deployment Overview

The **Sales Return Management System** was deployed and tested in a local development environment. This deployment was focused on internal testing and demonstration purposes. Live production deployment was not included within the current project scope.

Environment used:

- **OS:** Windows 11
- **Frontend:** React.js (built with Vite) and styled using Tailwind CSS
- **Backend:** Node.js with Express.js
- **Database:** MongoDB (local instance)
- **Authentication:** JWT (JSON Web Token) for secure authentication and role-based access
- **PDF Generation:** jsPDF library for automatic invoice generation

9.2 Local Setup and Execution

To set up and run the project locally, follow these steps:

1. **Clone the project repository** (or copy source files).
2. **Install and set up the backend:**

   cd backend

   npm install

   npm run dev
3. **Install and set up the frontend:**

   cd frontend

   npm install

   npm run dev
4. **Configure MongoDB** connection settings inside the backend .env file.
5. **Ensure MongoDB is running** locally (or connect to a cloud database like MongoDB Atlas).
6. **Access the application:**
   - Frontend: http://localhost:5173/
   - Backend API: http://localhost:5000/

9.3 Deployment Readiness

While production deployment was not part of the project scope, the system is designed to be deployment-ready. It can be easily hosted on platforms such as:

- **Vercel** or **Netlify** (for frontend hosting)
- **Render**, **Railway**, **AWS EC2**, or **DigitalOcean** (for backend and database hosting)
- **Docker** (for containerized full-stack deployment)

The modular codebase, API-driven architecture, and environment-based configurations make future deployment seamless.

9.4 Maintenance During Development

Development was maintained systematically with best practices:

- **Version control:** Regular commits and collaboration using Git and GitHub
- **Separation of concerns:** Clean separation between frontend, backend, and database operations
- **Bug fixing and code review:** Issues were identified and resolved after each testing cycle
- **Database backups:** Periodic dumps of the MongoDB database were taken during major updates

9.5 Future Maintenance Considerations

For future scaling and production readiness, the following practices are suggested:

- **Automate database backups** (using MongoDB Atlas or cron jobs)
- **Enable structured logging** using libraries like winston or morgan
- **Implement monitoring tools** (e.g., PM2 for Node.js backend, UptimeRobot for server health)
- **Enhance security measures** like HTTPS enforcement, Helmet.js middleware, and strong password policies
- **Enable email notifications** for important actions (like return approval, refund status, etc.)

CHAPTER 10
EVALUATION

10.1 Objective Fulfilment

The primary objective of the Sales Return Management System project was to design and develop a full-stack web application that could efficiently manage customer orders, return requests, and generate invoices automatically. Based on the planned deliverables, all major modules were successfully implemented, including:

- Order Placement and Management
- Return Request Handling with Approval Workflows
- Automated Invoice Generation in PDF Format
- User Authentication and Role-Based Access Control (Admin, Staff, Customer)
- Product Management and Inventory Updates

Each module aligned closely with the initial functional requirements and was validated through internal testing and mentor reviews, demonstrating complete fulfilment of the project's goals.

10.2 Testing Outcomes

To ensure reliability and robustness, the following testing strategies were applied:

- **Unit Testing**: Core functionalities like order creation, return request submission, and invoice PDF generation were tested individually to validate input-output behavior.
- **Integration Testing**: Complete workflows were tested to ensure smooth operation, such as:
    - Customer places an order → Order is recorded → Invoice is generated automatically
    - Customer submits a return request → Staff/Admin reviews and updates the request status appropriately
- **Bug Fixes**: Minor issues related to form validation, authentication flow, and API response handling were encountered and resolved during development.
- **Manual Testing by Mentor**: Regular review sessions were conducted where the mentor tested the system and provided valuable feedback that was incorporated into the project.

The system passed all functional tests successfully and was considered stable for internal deployment.

10.3 Feedback on Usability and Design

The user interface was built using **React** and **Tailwind CSS**, focusing on responsiveness and an intuitive user experience. Key feedback from mentor evaluations included:

- Clean and minimalistic layout with responsive design for different devices
- Clear navigation and well-organized dashboards for different user roles
- Effective use of toast notifications and error/success handling
- Minor suggestions related to button placements and additional form validations were addressed during the final UI refinements.

10.4 Strengths of the System

- **Modern Tech Stack**: Utilized React, Tailwind CSS, Vite, Node.js, Express.js, and MongoDB, ensuring a fast and scalable architecture.
- **Secure Authentication**: Implemented JWT-based authentication and role-based access control to ensure data security.
- **Automated Processes**: Integrated PDF generation using libraries like **jsPDF** for automatic invoice creation at order placement.
- **Scalable Database Design**: MongoDB's flexible document structure supports easy expansion for future enhancements.
- **Real-World Process Mapping**: Accurately modeled common sales and returns workflows seen in e-commerce operations.

10.5 Limitations and Challenges

- Due to the time constraints of the internship, advanced features like real-time inventory synchronization, email/SMS notifications, and analytics dashboards were not included.
- Deployment was limited to a local environment; no cloud hosting (e.g., AWS, Azure) was performed during the internship phase.
- Integrating with third-party payment gateways and CRM systems was beyond the scope of the initial project.
- Balancing the development of frontend, backend, authentication mechanisms, and database operations as a solo developer was challenging but provided significant learning opportunities.

10.6 Final Remarks

The Sales Return Management System project was successfully completed within the internship timeline and achieved all the intended objectives. It offered valuable hands-on experience in full-stack web development using modern frameworks and tools.

The project was reviewed positively by the mentor and was assessed as functional, secure, and well-structured. This internship proved to be an important milestone, helping strengthen technical knowledge, practical development skills, and a deeper understanding of professional project delivery.

CHAPTER 11
CONCLUSION AND FUTURE SCOPE

11.1 Conclusion

The successful development of the **Sales Return Management System** during my internship marked a significant milestone in translating theoretical knowledge into practical software solutions. Over the course of the project, I followed a structured **System Development Life Cycle (SDLC)** approach to design, implement, and test a system that efficiently manages the end-to-end process of handling product returns from customers.

Key functionalities such as **Sales Return Request Management**, **Approval Workflows**, **Inventory Adjustment**, **Refund/Replacement Processing**, and **Role-Based Access Control** for Admins, Sales Managers, and Warehouse Staff were developed. The project enhanced my skills in **backend development** using **Python-Django**, **frontend development** with **HTML/CSS/JavaScript**, and **database integration** using **MySQL**. I also gained valuable experience in **agile planning**, **modular coding**, and **version control** through Git.

This project provided real-world exposure to building business-critical applications, focusing on data integrity, user experience, and efficient workflow management. By leveraging academic knowledge [1], [2], and current industry technologies [3]–[7], the Sales Return Management System was successfully delivered as a reliable, scalable solution for handling sales return operations.

11.2 Future Scope

Although the Sales Return Management System currently covers all essential return-related processes, there are several opportunities for future enhancements, including:

- **Integration with ERP Systems**: Synchronize return data with enterprise systems for seamless accounting and inventory management.
- **Advanced Analytics**: Implement reporting dashboards for tracking return reasons, return rates, and customer satisfaction metrics.
- **Mobile App Development**: Build a companion mobile app for sales teams and warehouse staff to manage returns on-the-go.
- **Automated Notifications**: Integrate real-time email/SMS alerts for return approvals, refund processing, and inventory updates.
- **Cloud Hosting and Scalability**: Deploy the system on cloud platforms to ensure remote accessibility, high availability, and easy scalability for larger organizations.

REFERENCES

1. Chaffey, D. (2022). Digital Business and E-Commerce Management. Pearson Education.
2. Laudon, K. C., & Traver, C. G. (2021). E-commerce 2021: Business, Technology, Society, Pearson.
3. Kotler, P., Keller, K. L., & Chernev, A. (2022). Marketing Management. Pearson Education.
4. Turban, E., King, D., Lee, J., Liang, T. P., & Turban, D. (2020). Electronic Commerce: A Managerial and Social Networks Perspective. Springer.
5. Gupta, S., & Ramachandran, D. (2021). Consumer Behavior in the Digital Age. Journal of Marketing, 85(3), 24-39.
6. Chen, Y., & Zhang, L. (2020). Return Policies and Customer Satisfaction in E-commerce. Journal of Business Research, 113, 215-225.