"MercatoHub: Empowering Small-Scale Businesses Through a Unified E-Commerce Platform"

A project report submitted for the partial fulfillment of academic requirements for the award of Degree

in

BACHELOR OF ENGINEERING

in the Department of

INFORMATION SCIENCE & ENGINEERING

Major Project (21IS8P01)

Submitted by

Poorvansh Gupta	4NI21IS073
Shubham Sahu	4NI21IS101
Yanamandram Sai Koushik	4NI21IS118
Kshitiz Mayank	4NI21IS128

Under the Supervision of
Ms. Mayura D Tapkire
Assistant Professor
Department of Information Science and Engineering, NIE, Mysuru

Department of Information Science & Engineering

The National Institute of Engineering

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

Manandavadi Road, Mysuru 570 008, Karnataka, India

2024-25

The National Institute of Engineering

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

Manandavadi Road, Mysuru–570008, Karnataka, India

Recognized by AICTE, New Delhi
Accredited by National Board of Accreditation, New Delhi
Grant-in-Aid by Government of Karnataka

Department of Information Science & Engineering

CERTIFICATE

Poorvansh Gupta	4NI21IS073
Shubham Sahu	4NI21IS101
Yanamandram Sai Koushik	4NI21IS118
Kshitiz Mayank	4NI21IS128

Certified that the project work entitled "**MercatoHub: Empowering Small-Scale Businesses Through a Unified E-Commerce Platform**" carried out by above bonafide students of 8th Semester is submitted in partial fulfillment as part of **Major Project** for the award of Bachelor of Engineering Degree in Information Science and Engineering of **The National Institute of Engineering, Mysuru,** an autonomous institute under Visvesvaraya Technological University, Belagavi during the 2024-2025. It is certified that all suggestions/corrections suggested during the Internal Assessment have been incorporated in the report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project Work prescribed for the award of the said Degree.

Guide Signature	HoD Signature	Principal Signature
Ms. Mayura D Tapkire	**Dr. Girish**	**Dr. Rohini Nagapadma**
Assistant Professor,	**Prof. &HoD.**	**Principal,**
Dept of ISE, NIE, Mysore	**Dept of ISE, NIE, Mysore**	**NIE, Mysore**

Viva-Voce:

Name of the Examiners	Signature with Date
1.	
2.	

DECLARATION

We, Poorvansh Gupta bearing USN: 4NI20IS073, Shubham Sahu bearing USN: 4NI20IS101, Yanamandram Sai Koushik bearing USN: 4NI21IS118 and Kshitiz Mayank bearing USN: 4NI21IS128, students of 8th semester of UG program, Department of Information Science and Engineering, The National Institute of Engineering, Mysuru hereby declare that the project work entitled "**MercatoHub: Empowering Small-Scale Businesses Through a Unified E-Commerce Platform**" has been carried out by us under the Guidance of **Ms. Mayura D Tapkire**, Professor, Department of I.S.&E. This project work is submitted to **The National Institute of Engineering**, Mysuru, (An Autonomous institute under VTU, Belagavi) in partial fulfillment of the **Major Project** requirements for the award of a degree in Information Science & Engineering during the academic year 2024-2025. This written submission represents a record of original work, and we have adequately cited and referenced the original sources. Further, the matter embodied in this report has not been submitted to any other University or Institution for the award of any degree.

Place: Mysuru
Date: 21/04/2025

Shubham Sahu	4NI21IS101
Poorvansh Gupta	4NI21IS073
Yanamandram Sai Koushik	4NI21IS118
Kshitiz Mayank	4NI21IS128

ACKNOWLEDGEMENT

We are extremely thankful to **Dr. Rohini Nagapadma**, Principal, NIE, Mysuru, for providing us with the academic ambiance and laboratory facilities to work, and everlasting motivation to carry out this work and shape our careers.

We express our sincere gratitude to **Dr. Girish**, HoD, Dept. of Information Science & Engineering, NIE, Mysuru, for his stimulating guidance, continuous encouragement, and motivation throughout the course of the present work.

We extend our gratitude to our Guide and Project Coordinator **Mrs. Mayura D Tapkire**, Dept. of Information Science and Engineering, NIE, Mysuru for her support and guidance over the entire course of work.

We would also extend our gratitude to our Co-Guide **Ms. Lammiya Huda**, Dept. of Information Science and Engineering, NIE, Mysuru for her support and guidance

We take this opportunity to thank all our friends, classmates who always stood by us in difficult situations also helped us in some technical aspects, and last, but not least, we wish to express our inspiration and sense of gratitude to our parents who were a constant source of encouragement and stood by us as a pillar of strength for completing this work and course successfully.

Shubham Sahu	4NI21IS101
Poorvansh Gupta	4NI21IS073
Yanamandram Sai Koushik	4NI21IS118
Kshitiz Mayank	4NI21IS128

ABSTRACT

MercatoHub is a comprehensive full-stack e-commerce platform meticulously crafted to support and uplift small-scale businesses and local entrepreneurs by offering them a feature-rich, cost-effective, and accessible digital marketplace. In contrast to traditional e-commerce giants that often impose high commission fees and prioritize large-scale sellers, MercatoHub emphasizes affordability, inclusivity, and personalized service by enabling emerging and local vendors to seamlessly showcase, manage, and sell their unique product offerings online.

MercatoHub is designed with dual-user interaction models: administrators and customers. Admin users can access dashboards to monitor real-time analytics, manage product inventories, process orders, and oversee business insights, whereas customers can enjoy seamless product exploration, categorized browsing, real-time cart operations, and a smooth checkout simulation.

This document details the architectural decisions, implementation methodologies, component integrations, and workflow pipelines that make up the MercatoHub platform. By providing a realistic and academic demonstration of a full-stack commerce application, MercatoHub not only showcases development best practices but also offers a strategic solution for empowering underserved business segments in the digital age.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SCREENSHOTS

CHAPTER 1
INTRODUCTION

MercatoHub is a full-stack e-commerce platform created to serve as a digital marketplace tailored specifically for small-scale businesses and local entrepreneurs. In contrast to dominant platforms like Amazon or Flipkart, which often overshadow smaller sellers, MercatoHub offers a space where these businesses can showcase and sell their products at competitive prices.

This document outlines the motivation, objectives, and technological backbone of the project, including its evolution from existing systems to the proposed custom-built solution.

1.1 Objective

The primary objective of MercatoHub is to create a platform that bridges the gap between small-scale product manufacturers and digital consumers. Key goals include:

Empowering small businesses with a dedicated marketplace.

Offering customers affordable and unique alternatives to mass-produced goods.

Providing an intuitive and responsive frontend experience for seamless shopping.

Implementing admin functionalities for order and product analytics.

Simulating a realistic e-commerce environment with secure login, cart functionality, and payment workflow.

This objective extends beyond academic demonstration; it lays a foundation for a socially impactful application that promotes grassroots commerce.

1.2 Existing System

Currently, mainstream e-commerce ecosystems are highly competitive and saturated. Popular platforms like Amazon, Flipkart, and Myntra offer:

- Massive inventories backed by large corporations.
- High entry fees or listing commissions for small vendors.
- Algorithmic favoritism towards popular or high-revenue items.
- Limited visibility for newly listed or small-business products.
- These systems often discourage or underrepresent micro-entrepreneurs who don’t have the resources or supply chain capacity to compete on equal footing.

Limitations of existing systems:

- Lack of identity for small brands.
- High commission and logistical costs.
- Generic product saturation with minimal differentiation.
- Inaccessible analytics or sales data for sellers without premium accounts.

1.3 Proposed System

MercatoHub proposes a solution specifically designed to:

- Provide a zero-cost listing platform for small vendors.
- Simulate a marketplace experience where product categories (electronics, fashion, footwear, etc.) are visually segmented.
- Incorporate cloud-based image hosting using Cloudinary to handle seller-uploaded product media efficiently.
- Leverage Firebase Authentication to securely manage users and roles.
- Introduce a mock payment gateway using Redis to simulate transactions, without real-world financial risk.
- Enable admin oversight through dashboards that track sales performance and user activity.

Key features of the proposed system:

- User-friendly frontend powered by Vite + React + TypeScript.
- RESTful API architecture for seamless frontend-backend interaction.
- Scalable backend using Node.js and Express.js, with Redis and MongoDB for data management.
- Admin role functionality for business control and decision-making.
- Minimal cost setup with free-tier tools for academic scalability and real-world potential.

In essence, the proposed system not only solves the accessibility issue for small sellers but also showcases how modern web development tools can create real-world business applications at near-zero cost.

CHAPTER 2

LITERATURE SURVEY

A literature survey provides the foundation upon which the proposed system, MercatoHub, has been built. It involves a critical examination of existing research papers, online platforms, whitepapers, and technical documentation concerning e-commerce systems, platforms supporting small-scale businesses, authentication mechanisms, cloud-based image hosting, and session-based payment simulations. The insights gathered through this review helped guide the design and development of MercatoHub to ensure it is aligned with current best practices and the unique needs of its target users.

Modern e-commerce systems are generally structured around modular, scalable architectures, often incorporating REST APIs, client-server models, and data-driven user interfaces. Platforms like Amazon and Flipkart use complex recommendation engines, cloud-hosted microservices, and large-scale analytics to enhance user interaction and product placement. Although MercatoHub is an academic project, it borrows from these structural paradigms—particularly the Model-View-Controller (MVC) architectural pattern—for modularity and maintainability. Notably, literature suggests that monolithic architectures struggle to scale efficiently, which has driven many developers toward microservices [Gupta et al., 2021]. Furthermore, UI/UX design is cited as essential for reducing bounce rates and establishing customer trust (Nielsen Norman Group, 2020), which directly influenced the frontend decisions in our project.

To manage user authentication and access control, Firebase was adopted as a backend-as-a-service (BaaS) solution. Firebase provides fast, secure, and scalable services for user registration, token-based login, and role-based access, all of which are essential for maintaining a secure and streamlined platform. Google’s documentation and numerous community sources reinforce Firebase’s effectiveness in supporting quick integration with modern web applications. MercatoHub utilizes Firebase to handle email-password authentication flows and restrict access based on user roles (admin or customer), ensuring both security and personalization.

For media management, Cloudinary was integrated into the system to handle image hosting, transformation, and delivery. Cloudinary is widely respected in the development community for

its ability to optimize images via automatic format conversion (e.g., WebP), responsive delivery, and CDN-backed performance. According to a 2022 technical whitepaper by Cloudinary, image optimization can reduce load times by up to 70%, thereby improving user retention and SEO metrics. Its API-first approach fit well within the development framework of MercatoHub, ensuring consistent image rendering across different devices and browsers.

In terms of temporary data storage and simulation of payment sessions, Redis was utilized due to its high-speed in-memory data operations. Redis is frequently used in session handling for e-commerce carts and simulating ephemeral transactional data. Although MercatoHub does not implement a real payment gateway, Redis enabled the platform to generate mock payment IDs and checkout sessions that mimic real-world behaviour. The utility of Redis in similar domains has been documented by industry blogs, including those by Zalando and Shopify, further validating our design choice.

Several existing platforms were analysed for reference and contrast. Etsy supports independent creators but remains niche and lacks customization flexibility. Meesho facilitates micro-entrepreneurs but is built on a reseller-based model with limited branding opportunities. Indiamart, on the other hand, functions primarily as a B2B portal and is not designed for a user-centric retail experience. These platforms served as case studies to identify gaps that MercatoHub attempts to address, especially in terms of seller control, pricing transparency, and customizable presentation.

In summary, the literature survey highlights that small businesses benefit greatly from accessible, low-cost, and customizable digital platforms. Tools like Firebase, Cloudinary, and Redis enable rapid development of scalable and feature-rich systems without significant overhead. Current mainstream platforms fall short in offering full ownership and customization to sellers. These observations were central to the conception of MercatoHub, which aims not only to demonstrate technical proficiency but also to empower small businesses through inclusive and efficient design.

CHAPTER 3
SYSTEM REQUIREMENTS

This section outlines the necessary hardware and software configurations required to develop, test, and deploy the MercatoHub platform. The requirements are based on the project's chosen tech stack, performance expectations, and integration of external services such as Firebase, Redis, and Cloudinary.

3.1 Software requirements:

The software stack for MercatoHub is based on modern full-stack development tools, combining powerful frontend frameworks with efficient backend services.

Software Component	**Details / Version**
Operating System	Windows 10/11, Linux (Ubuntu), or macOS
Frontend Framework	React (via Vite) with TypeScript
Backend Environment	Node.js (v18.x or higher)
Package Managers	npm / yarn
Database	MongoDB (cloud or local instance)
Authentication	Firebase Authentication
Media Storage	Cloudinary
Session Store	Redis (for cart/payment simulation)
Code Editor	Visual Studio Code
API Testing Tools	Postman / Thunder Client
Version Control	Git + GitHub
Browser Support	Chrome, Firefox, Edge

Table 3.1 Software Requirements

This software ecosystem was chosen to ensure fast development, seamless integration, and scalable simulation of e-commerce features.

3.2 Hardware Requirements:

Local Devices at Hospitals (For Local Training) To ensure smooth development and execution of both frontend and backend components, the following hardware specifications were considered ideal:

Component	Minimum Requirement	Recommended Requirement
Processor	Intel i3 10th Gen / AMD Ryzen 3	Intel i5 11th Gen or higher
RAM	8 GB	16 GB
Storage	256 GB SSD or 500 GB HDD	512 GB SSD
Graphics	Integrated GPU	Dedicated GPU (Optional)
Display	13-inch, Full HD	15.6-inch or larger, Full HD
Internet	Stable broadband connection	High-speed broadband (5+ Mbps)

Table 3.2 Hardware Requirements

These specifications support local development servers, build processes, and real-time testing using tools such as Vite, Firebase Emulator, and Redis.

CHAPTER 4

SYSTEM DESIGN AND ARCHITECTURE

This section outlines the system design and architectural considerations for "MercatoHub" — an e-commerce platform dedicated to hosting and promoting products from small-scale businesses. The design encapsulates both high-level workflow and detailed component interactions to ensure scalability, maintainability, and smooth user experiences.

4.1 Workflow

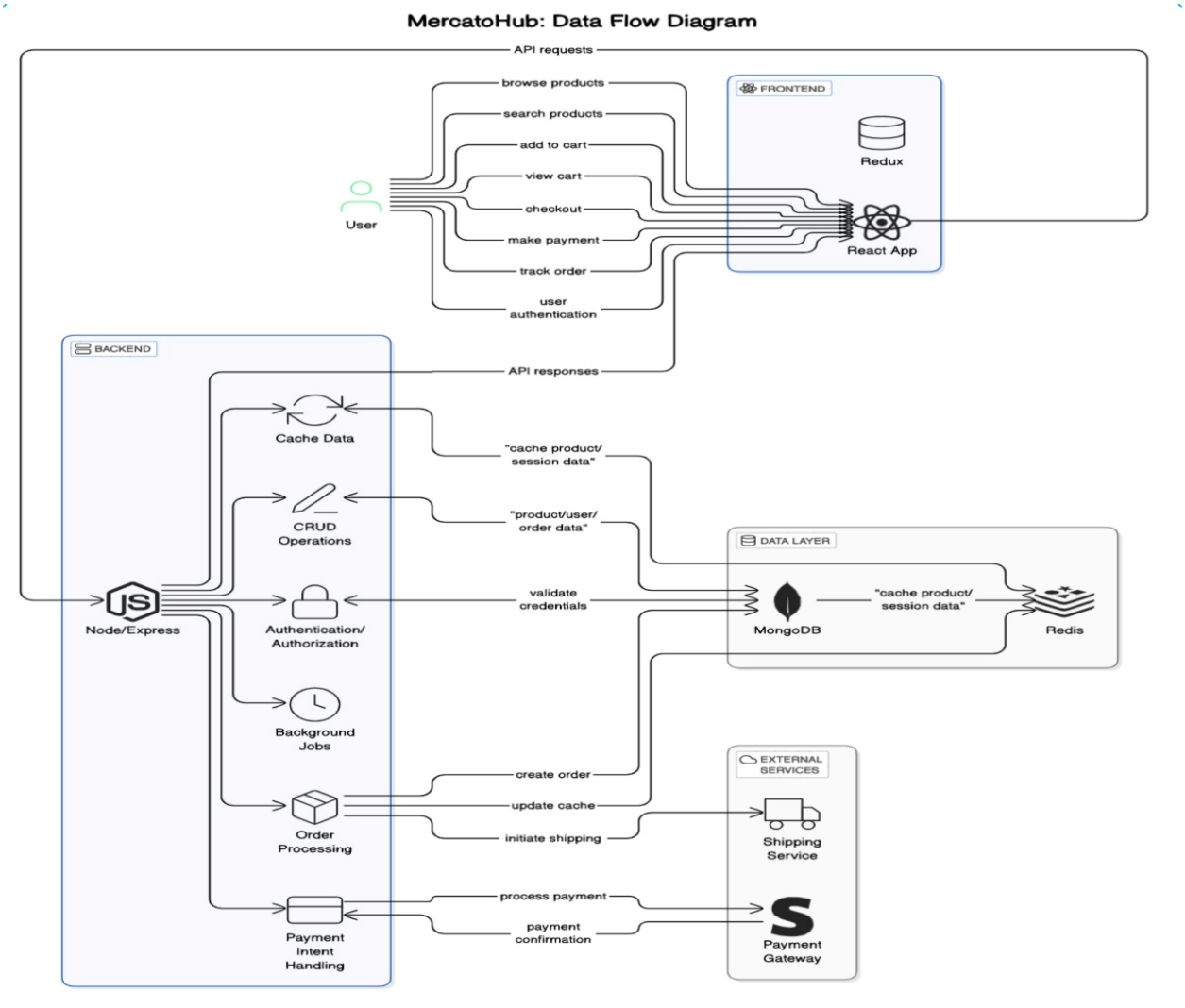

Fig 4.1 Workflow Diagram

4.1.1 User Workflow

From the moment a user lands on the website, they are introduced to a range of products across various categories such as Electronics, Fashion, Apparel, and Shoes. The landing page displays featured products, promotional banners, and category filters, all dynamically loaded from the backend database.

Users can either explore anonymously or register through a simple signup process powered by Firebase Authentication. Once registered and authenticated, users gain access to personalized features such as order history, secure checkout, and cart persistence.
Users can:

- Browse products by category.
- Search using keywords.
- View product details including descriptions, prices, and images hosted on Cloudinary.
- Add products to the cart.

Upon successful mock payment, users are redirected to an order confirmation page that details the items purchased, prices, and order summary.

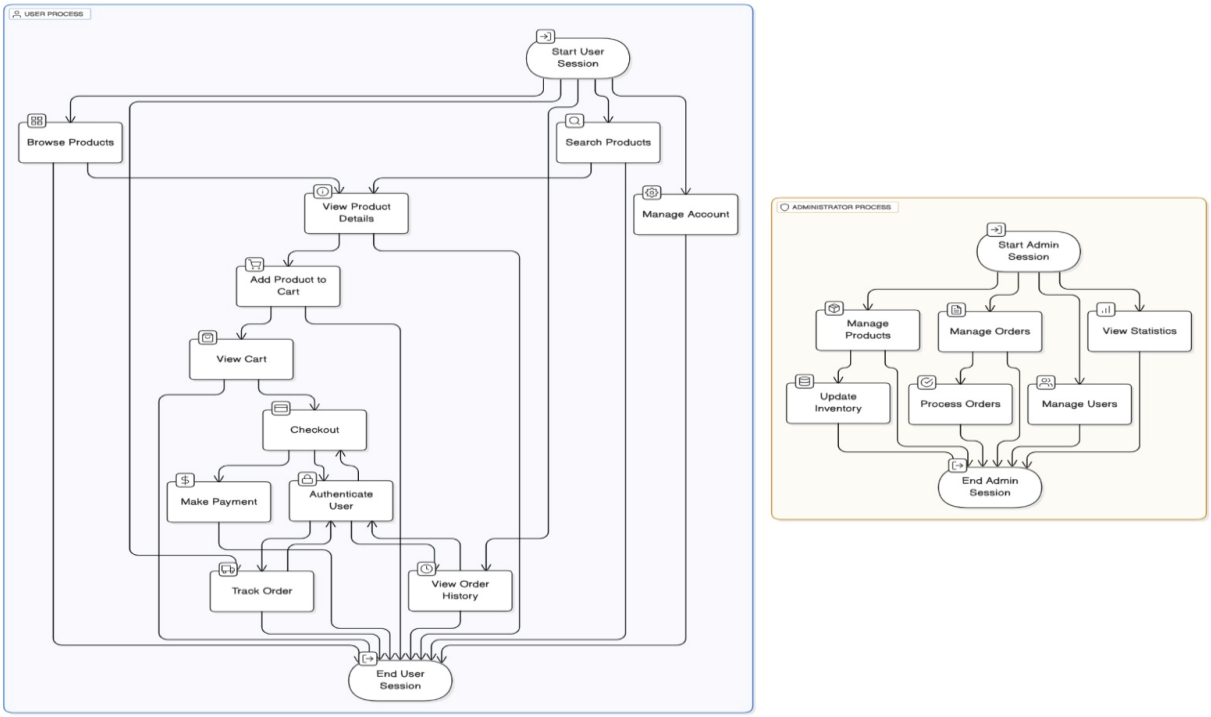

Fig 4.1.1 User Workflow

4.1.2 Admin Workflow

Administrators access a protected route managed by Firebase Authentication. After successful login, they are redirected to a dedicated admin dashboard.

Admin functionalities include:

- Adding, updating, or deleting product listings.
- Managing product categories.
- Monitoring overall platform analytics such as number of users, product performance, and sales metrics.

Admins can also upload product images directly to Cloudinary from the dashboard interface. These images are instantly linked to their respective products and displayed on the frontend.

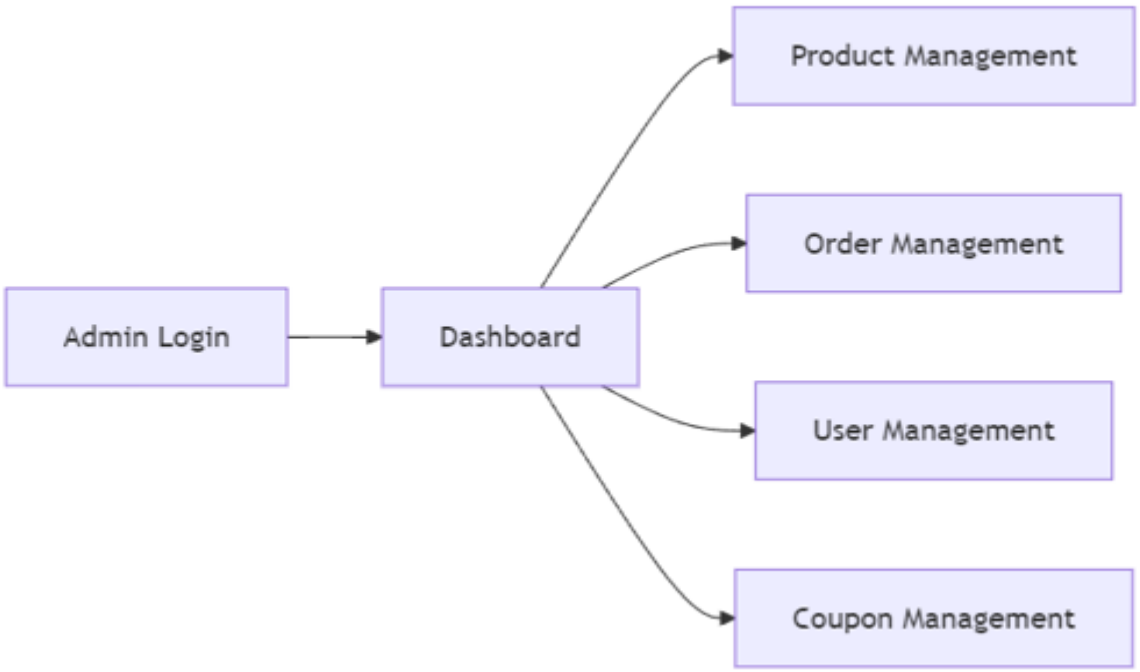

Fig 4.1.2 Admin functionalities

4.1.3 Integration Workflow

MercatoHub integrates with several third-party services for enhanced functionality:

- Firebase manages user authentication and role-based access.
- Cloudinary handles media uploads and fast delivery of product images.

- Redis is used to simulate payment and temporarily store session data.

This seamless integration between frontend, backend, and external services ensures users and admins experience minimal latency and efficient data flow throughout the application lifecycle.

4.2 Low Level Design

4.2.1 Frontend Architecture

The frontend is built using Vite with React and TypeScript, organized in a component-based architecture. Major components include:

- HomePage: Displays featured products and categories.
- ProductList: Shows product listings by category.
- ProductDetail: Displays a single product's information.
- Cart: Manages and displays items added for checkout.
- Checkout: Handles mock payment interaction and confirmation.
- Auth: Manages user login, registration, and authentication using Firebase.
- AdminDashboard: A role-based protected component with admin functionalities.

React Hooks and Context API manage state and effects, ensuring smooth user interaction and data synchronization with the backend.

4.2.2 Backend Architecture

The backend is developed using Node.js and Express, with both JavaScript and TypeScript for enhanced typing and error prevention. Key backend modules include:

- AuthController: Integrates Firebase to authenticate users and verify JWT tokens.
- ProductController: Handles all product-related CRUD operations.
- OrderController: Simulates and logs orders using Redis.

The backend exposes RESTful APIs consumed by the frontend. Middleware ensures secure access, rate-limiting, and error handling. Redis is used for managing temporary data such as cart sessions

and simulating a payment system.

4.2.4 Database Schema and Data Flow

The platform operates using the following logical entities:

- **Users**: { userId, name, email, role }
- **Products**: { productId, title, description, price, imageUrl, category }
- **Orders**: { orderId, userId, products[], totalAmount, createdAt }

Data flows from the frontend (user actions) to backend via API calls, triggering logic based on controller responsibilities. For example, when a user checks out, a Redis record is created and the product stock is adjusted (if applicable).

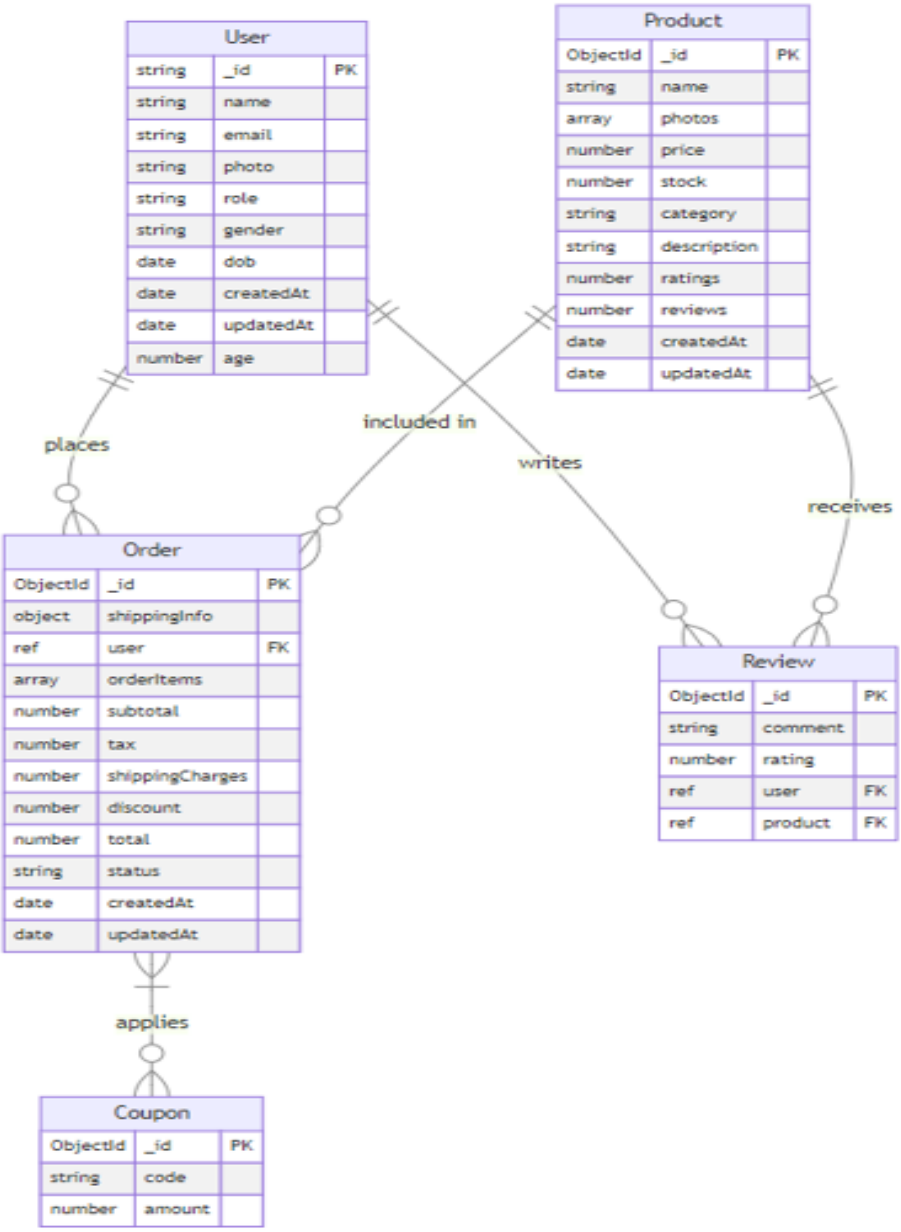

Fig 4.2.4 Database Design

CHAPTER 5

SYSTEM IMPLEMENTATION

This section covers the implementation phase of MercatoHub, describing how data and functionality were structured and simulated for the purposes of demonstrating a near-real e-commerce environment. Although the project does not involve machine learning models or traditional datasets, the following subsections adapt those concepts to reflect data flow and structure in a full-stack e-commerce application.

5.1 Frontend Implementation

5.1.1 User Interface & Core Features

The frontend implementation focuses heavily on creating an intuitive and engaging user interface. The home page showcases the latest products through responsive product cards, complemented by a dynamic banner slider and video cover that enhances visual appeal. Product details are presented through an interactive image carousel with zoom capabilities, allowing customers to examine products closely. The shopping experience is streamlined with features like quick add-to-cart functionality, real-time stock monitoring, and an efficient category-based filtering system. User authentication is seamlessly integrated, providing secure access to features like profile management, order tracking, and review systems. The interface maintains consistency across devices through responsive design principles, ensuring accessibility on both desktop and mobile platforms.

5.1.2 State Management & API Integration

At the core of the application lies a robust state management system implemented using Redux, handling everything from cart operations to user authentication states. The cart management system maintains persistent storage of items, processes real-time price calculations, and manages order states efficiently. API integration is accomplished through RTK Query, providing optimized data fetching with built-in caching mechanisms. This ensures smooth data flow between the frontend and backend services, handling product management, user authentication, order processing, and review systems. The implementation includes comprehensive error handling and

loading states, ensuring a reliable user experience even during network fluctuations. Real-time updates and state persistence are maintained throughout the application, providing a seamless shopping experience.

5.1.3 Admin Dashboard & Analytics

The administrative interface provides comprehensive control and monitoring capabilities through an intuitive dashboard. Real-time statistics display key metrics including revenue tracking, user analytics, and inventory levels. The dashboard features interactive charts and visualizations that present data on revenue trends, user growth, and order statistics. Management tools are integrated for efficient product handling, order processing, and user management. The analytics section includes detailed gender ratio analysis and transaction monitoring, providing valuable insights for business decisions. The implementation ensures secure access to administrative functions while maintaining high performance and real-time data updates. This comprehensive admin system enables effective business monitoring and quick response to market changes.

The frontend implementation creates a robust e-commerce platform that balances user experience with administrative control, built using modern React practices with TypeScript for enhanced code reliability. The integration of Redux for state management and RTK Query for API calls provides a solid foundation for scaling the application while maintaining optimal performance.

5.2 Backend Implementation

5.2.1 Controllers & Business Logic

The backend implementation centers around a robust controller architecture that manages core business operations. The product controller handles essential e-commerce functionalities including product creation, retrieval, updates, and deletion, along with sophisticated features like review management and category organization. Order management is implemented through dedicated controllers that process new orders, handle order status updates, and manage inventory adjustments automatically. User-related operations are managed through specialized controllers that handle user registration, authentication, and profile management. The implementation includes comprehensive error handling through try-catch middleware, ensuring reliable operation even under unexpected conditions. Redis caching is implemented strategically across controllers to optimize performance and reduce database load for frequently accessed data.

5.2.2 Data Management & API Architecture

The application's data layer is built on MongoDB, utilizing Mongoose schemas for structured data modelling. The models are carefully designed to handle complex relationships between products, orders, and users while maintaining data integrity. The API routes are organized following RESTful principles, providing clear endpoints for all major operations. Authentication middleware ensures secure access to protected routes, with special provisions for admin-only functionalities. The implementation includes sophisticated query handling for product filtering, sorting, and pagination, enabling efficient data retrieval. Real-time inventory management is implemented through automated stock updates during order processing, while the caching system ensures optimal performance for frequently accessed data patterns.

5.2.3 Analytics & Administrative Features

The backend provides comprehensive analytics capabilities through specialized statistics controllers. These controllers generate detailed insights including revenue analysis, user demographics, and inventory statistics. The dashboard functionality includes real-time data processing for various chart types (pie, bar, and line charts), providing valuable business intelligence. Administrative features include robust user management, order processing capabilities, and inventory control systems. The implementation includes Redis caching for dashboard statistics to ensure quick access to analytical data while maintaining system performance. Security measures are implemented through middleware that validates administrative access and protects sensitive operations.

The backend implementation creates a secure, scalable, and efficient foundation for the e-commerce platform, built using Node.js and Express with TypeScript. The integration of MongoDB for data storage and Redis for caching provides a robust infrastructure capable of handling high-volume operations while maintaining optimal performance.

5.3 Model Architecture

This section outlines the logical design and structure of MercatoHub’s backend application, which is responsible for handling API requests, user roles, and business logic.

5.3.1 Data Models

- User Model: Stores login data, roles, and session-related metadata
- Product Model: Captures product name, price, stock, description, and image links
- Order Model: Includes list of items, user references, and total transaction value

5.3.2 Controllers and MiddlewareREST API endpoints defined for CRUD operations

Middleware for:

- Role-based access control
- Input validation
- Token authentication

5.3.3 Redis Integration

Redis was used to simulate a working cart and payment session mechanism:

- Stores active cart sessions with expiry (TTL)
- Tracks order session IDs to simulate payments

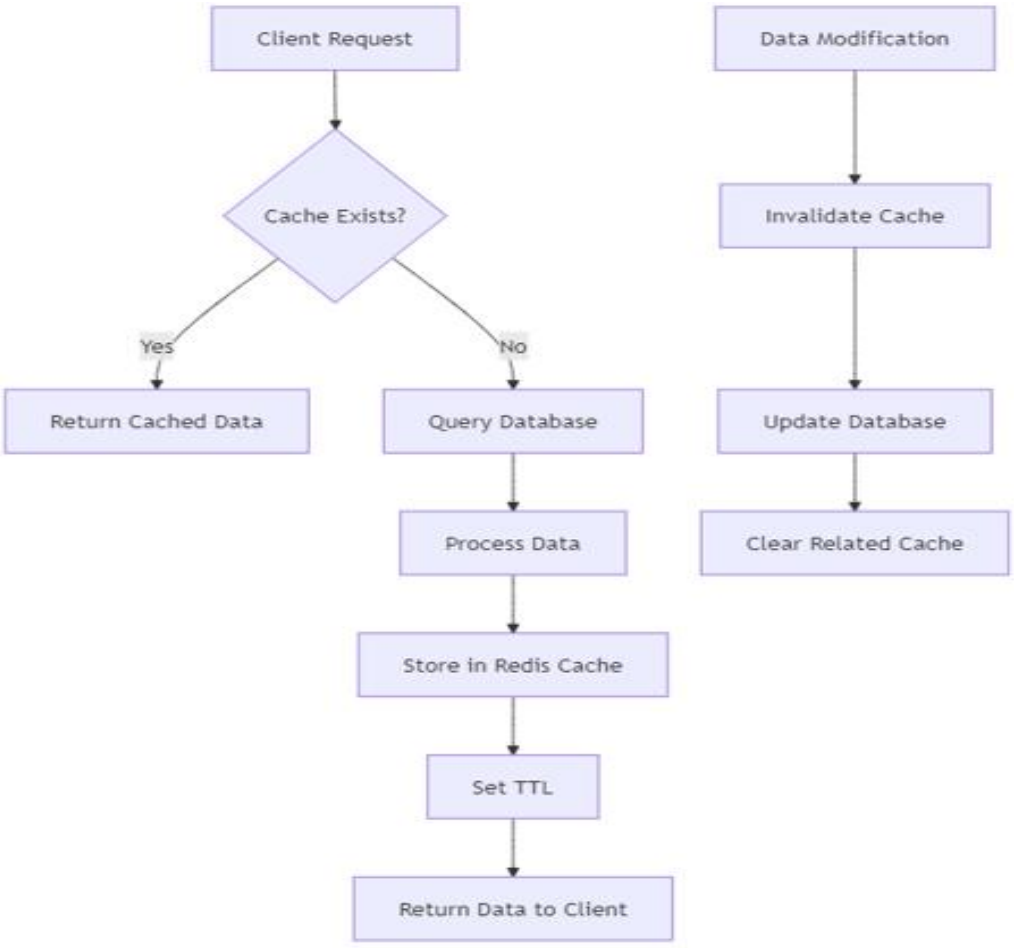

Fig 5.4.3: Redis Caching System

5.3.4 Admin Dashboard API

APIs to fetch sales insights:

- Number of products sold

- Revenue per category
- User activity log

5.4 Optimizations

5.4.1 Firebase Security Rules

- Configured access rights per collection type
- Admins allowed product and user management
- Customers allowed only order placement and browsing

5.4.2 Redis Optimization

- Tuned TTL for temporary sessions (e.g., 15 minutes for carts)
- Flushed expired data periodically to conserve memory

5.4.3 Cloudinary Delivery Settings

- Applied CDN-backed delivery with fast edge caching
- Fallback images and low-quality previews enabled for slower networks

5.4.4 Frontend Code Splitting

- Used Vite + React's built-in code splitting for lazy loading
- Minimized JS bundle size

5.5 Purpose and Scope of Document

The purpose of this document is to provide detailed insight into the implementation lifecycle of the MercatoHub platform. It serves as technical documentation for academic and evaluation purposes.

Scope Includes:

- Backend logic, models, and API integration
- Frontend interaction with REST APIs

- Use of external services like Firebase, Redis, and Cloudinary
- Project configurations that replicate real-world e-commerce functionality

By offering transparency into the codebase design, workflows, and system structure, this report supports future enhancements and academic review of the full-stack development approach taken in MercatoHub.

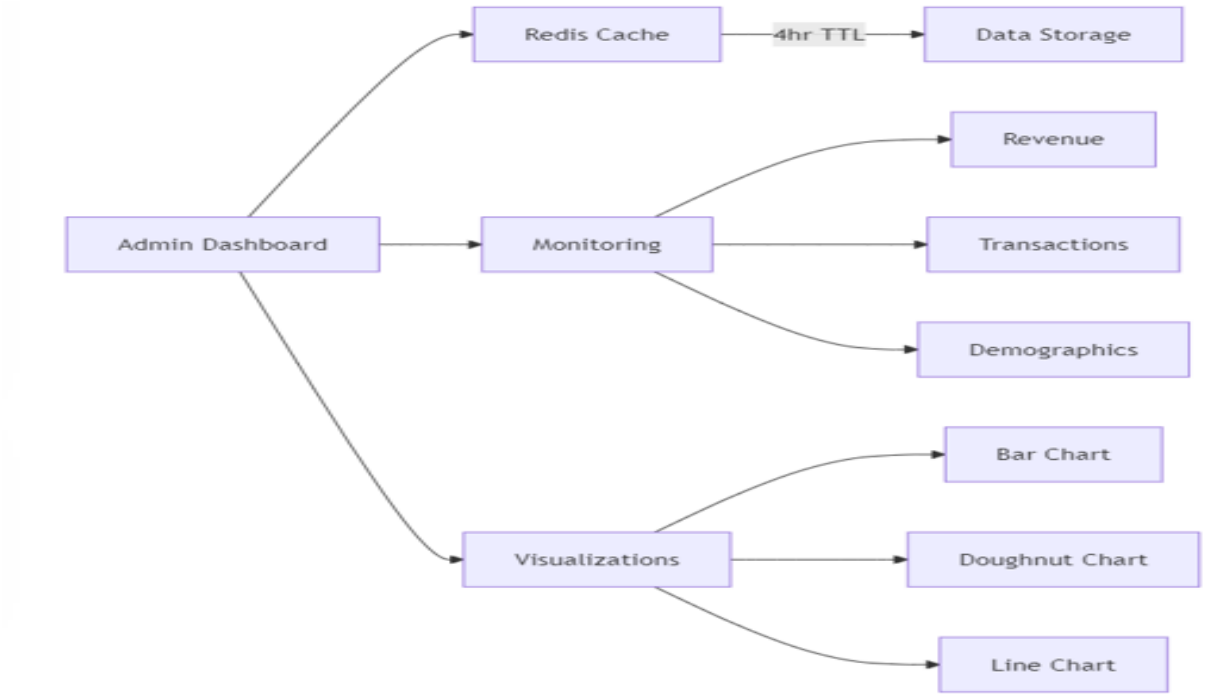

Fig 5.6: Platform Features

CHAPTER 6

SYSTEM TESTING AND RESULT ANALYSIS

This section evaluates the MercatoHub platform in terms of its robustness, functionality, and user experience by conducting a comprehensive system testing process. It also provides an analysis of the results derived from the tests and user flows.

6.1 Testing Methodology

We followed black-box testing and role-based scenario testing to ensure complete coverage of both frontend and backend functionalities without exposing the internal code structure to the tester.

6.1.1 Types of Testing Performed

- **Functional Testing:** Ensured all features such as signup/login, product listing, cart, and checkout behave as expected.
- **Integration Testing:** Checked those various modules (e.g., Firebase Auth, Cloudinary image rendering, Redis cart sessions) work seamlessly together.
- **Unit Testing (selective):** Focused on individual modules like add to cart, checkout handler, and admin insight retrieval.
- **User Acceptance Testing (UAT):** Involved simulating multiple user roles (admin/customer) to validate end-to-end flows.
- **Performance Testing (manual):** Tested loading speeds of images, page rendering delays, and product filtering performance.

6.2 Test Cases and Results

Test Case	Input/Action	Expected Output	Result
User Sign-Up/Login	Email & Password	Redirect to dashboard	✅ Passed

Add Product to Cart	Click “Add to Cart” on product	Cart icon updates, item appears in session	✅ Passed
Checkout Process	Click “Buy Now” with items in cart	Success message, session expires in Redis	✅ Passed
Admin Login and Dashboard Access	Login with admin credentials	Admin analytics and upload panel appear	✅ Passed
Image Display and Load Optimization	Visit product detail pages	Optimized images load without delay	✅ Passed
Broken Image Fallback	Corrupt/missing Cloudinary link	Fallback image appears	✅ Passed
Unauthorized Access Control	Customer tries admin route	"Access Denied" error	✅ Passed
Order Placement	User places order	Order logged, confirmation shown	✅ Passed
Category Filtering	Filter products by “Shoes” category	Only shoe-related products shown	✅ Passed

6.3 Results Overview

- **User Experience:** The UI was intuitive and responsive across most devices, supporting a clean and consistent shopping flow.
- **API Response Time:** Average response time was under 300ms for major routes including getProducts, addToCart, and getOrders.
- **Redis Session Handling:** Temporary sessions for cart functionality proved efficient; expiry times worked as configured.
- **Image Handling:** Thanks to Cloudinary's optimization and lazy loading in the frontend, product pages loaded quickly and without lag.
- **Error Handling:** All major failure points were accounted for with user-friendly error messages and backend logging.

6.4 Limitations Identified

- **Payment Gateway:** While mock payments worked, there is no real transaction validation logic implemented.
- **Mobile Responsiveness:** On very small screens (≤320px), minor layout shifts were observed on the admin dashboard.
- **Analytics Realism:** The admin insights are based on seeded or simulated data, not live analytics from real customers.

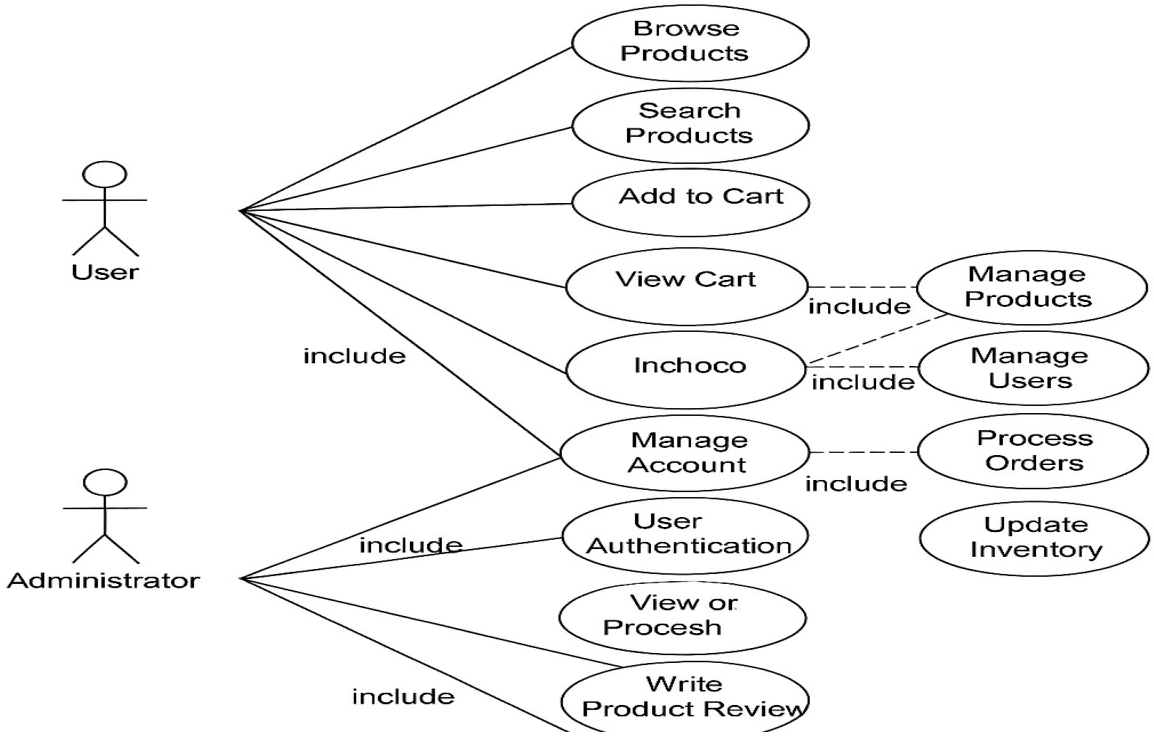

Fig 6.3 User and Admin Capabilities

6.5 User Feedback and Observations

During our demo sessions and peer reviews, the following feedback was noted:

- The concept of supporting small-scale sellers was well-received.
- The cart and mock payment flow helped in visualizing a complete user journey.
- Admin features like category-wise sales were found informative, although scope for graphical dashboards was suggested.

6.6 Conclusion of System Testing

MercatoHub passed all defined test cases for both customer and admin flows. The integration of third-party services like Firebase, Cloudinary, and Redis demonstrated reliable performance. With minor improvements, this platform could be scaled for broader deployment.

CHAPTER 7

COST EFFECTIVENESS OF PROJECT

One of the critical aspects of designing and deploying the MercatoHub platform was ensuring that the development remained within a highly cost-effective framework. Given the academic nature of the project and its mission to empower small-scale businesses, the architectural and technological decisions were consciously made to minimize operational and developmental costs while maintaining professional-level performance and scalability.

7.1 Open-Source and Freemium Technologies

MercatoHub was built using a suite of open-source and freemium tools, which drastically reduced the need for expensive licenses or paid subscriptions:

- **Frontend:** Developed with Vite, React, and TypeScript, all of which are free and open-source. This stack offers a modern development experience without incurring any licensing costs.
- **Backend:** Built using Node.js, Express.js, and TypeScript. These are community-supported and free, allowing rapid backend development and scalability.
- **Cloudinary:** Used for image hosting. Cloudinary provides a generous free tier (25 monthly credits) suitable for academic and low-traffic use cases. All product media were optimized using the free quota.
- **Firebase:** Firebase Authentication and Firestore offer free-tier usage sufficient for small projects. Features such as email/password login, role-based access, and real-time sync were achieved without incurring charges.
- **Redis:** Deployed in development using Redis Stack Docker or Redis Cloud Free Tier, which supports limited memory and connections—sufficient for mocking cart sessions and ephemeral payment data.

7.2 No Hosting/Infrastructure Cost

During development and demonstration, hosting was either:

- **Local:** for backend APIs and Redis
- **GitHub Pages / Firebase Hosting:** for frontend previews
- **MongoDB Atlas Free Tier:** for database

These platforms allowed zero-cost deployments with:

- 512MB storage on Atlas
- 1 GB bandwidth on Firebase
- 30 concurrent connections on Redis Cloud (free)

7.3 Simulated Services vs Real Integration

To minimize costs without compromising on the demonstration value:

- **Mock Payment Gateway:** Instead of integrating paid services like Stripe or Razorpay, a custom payment simulator using Redis was implemented. This eliminated API call charges and security setup costs.
- **Fake Order Generation:** Orders were manually created using scripts, avoiding the need for costly analytics or CRM tools.

7.4 Human Resource Cost

Since this was a student-driven academic project, the entire application was developed without paid developers. The division of labor was managed among team members:

- Frontend and UI by designated teammates
- Backend APIs, Redis integration, and database setup collaboratively managed
- Documentation and presentation materials were self-created

This reduced not only development costs but also the need for hiring freelancers or third-party designers.

7.5 Scalability and Future Cost Projection

MercatoHub is designed with scalability in mind. If the platform were to grow post-academic phase:

- Firebase and Cloudinary can scale based on usage plans.
- Redis can be hosted on affordable platforms like Upstash or DigitalOcean.
- Vercel or Netlify can serve as full-stack deployment platforms with generous free plans.

CHAPTER 8

SCREENSHOTS

Fig 8.1 Dashboard Page

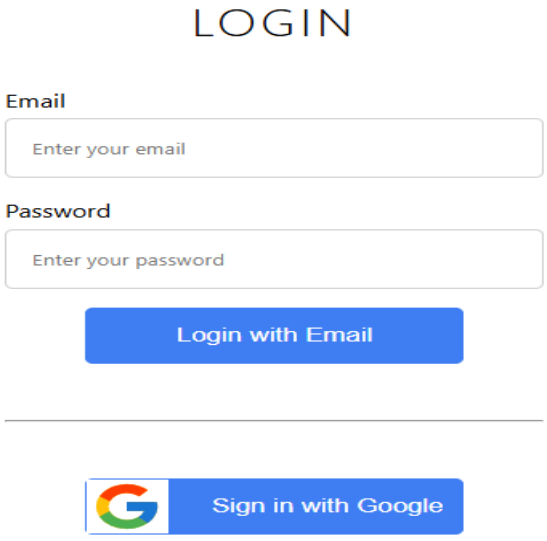

Fig 8.2 Login Page

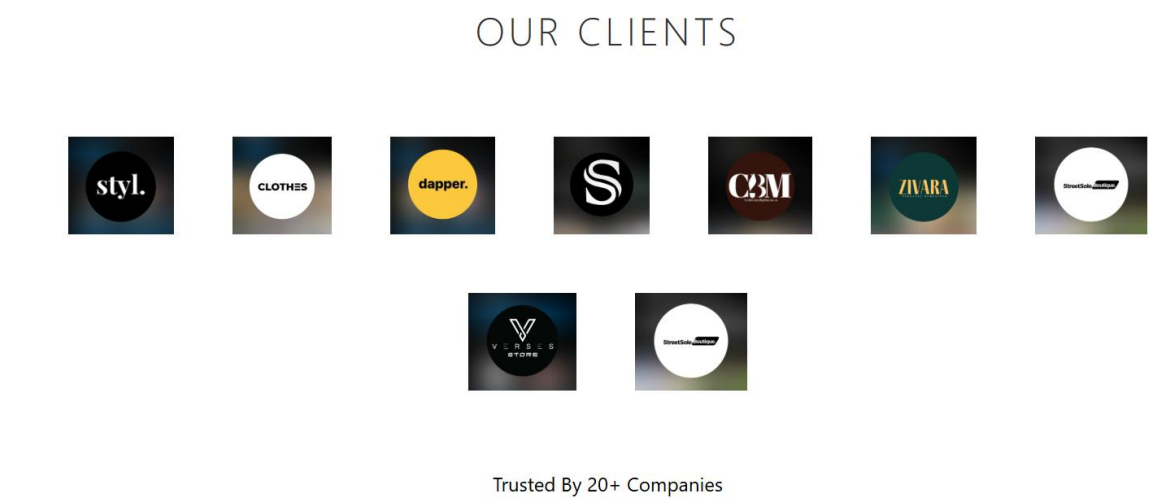

Fig 8.3 Our Clients

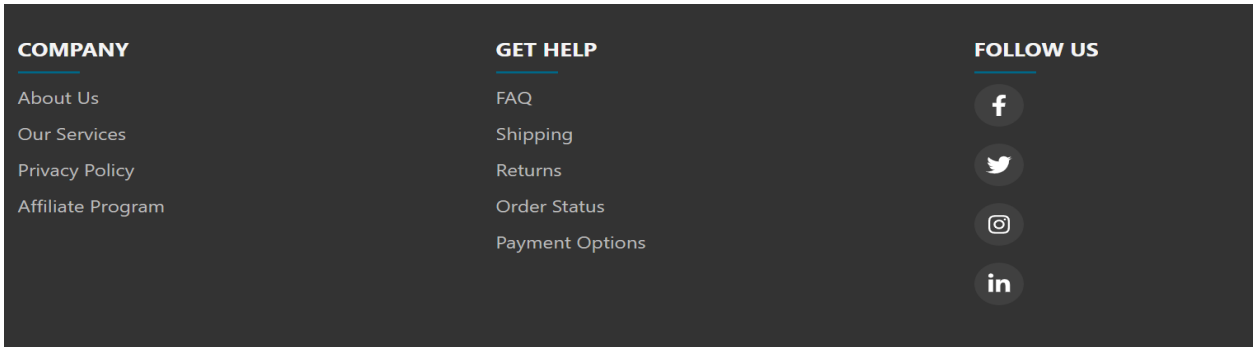

Fig 8.4 Footer Section for Further Connectivity

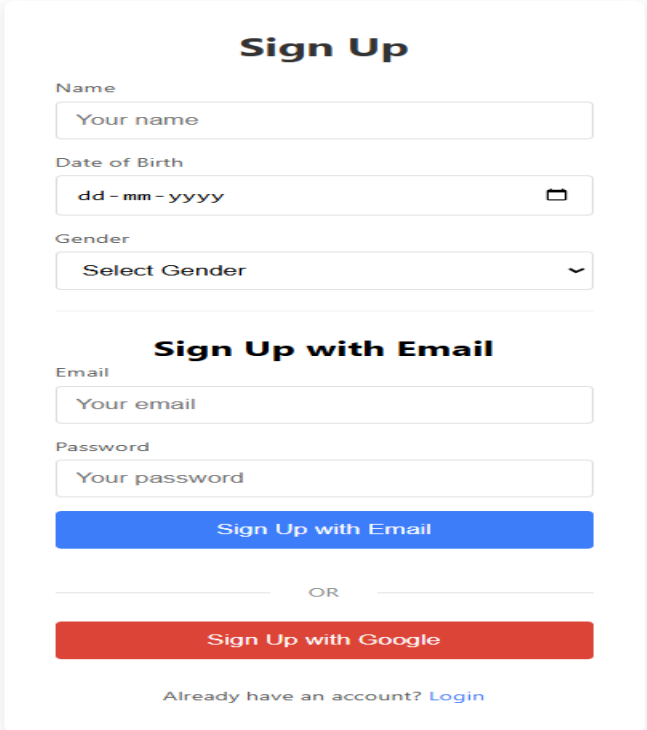

Fig 8.5 Signup Page

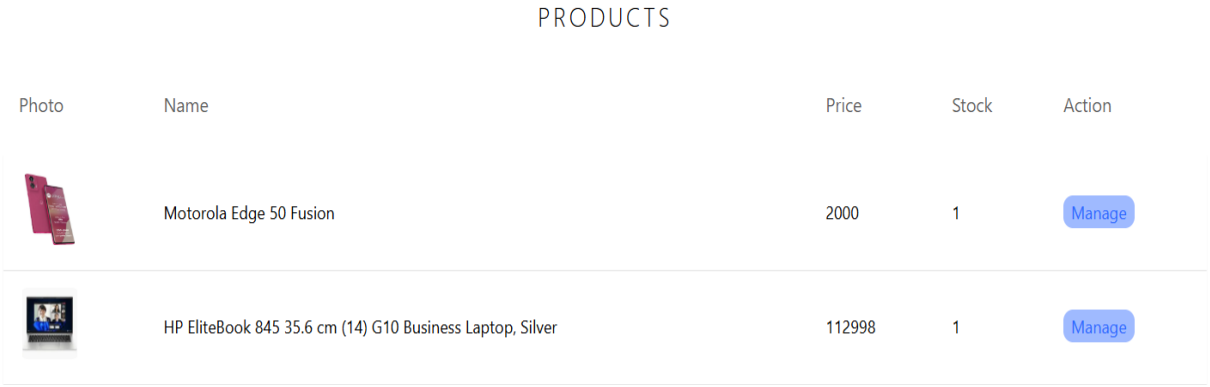

Fig 8.6 Product Management

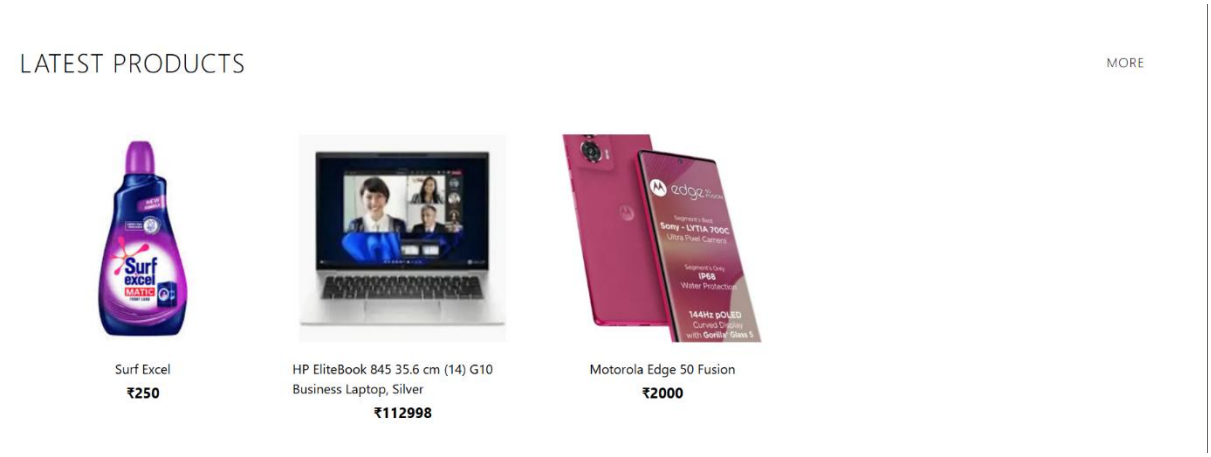

Fig 8.7 Latest Products

Fig 8.8 Product in Cart

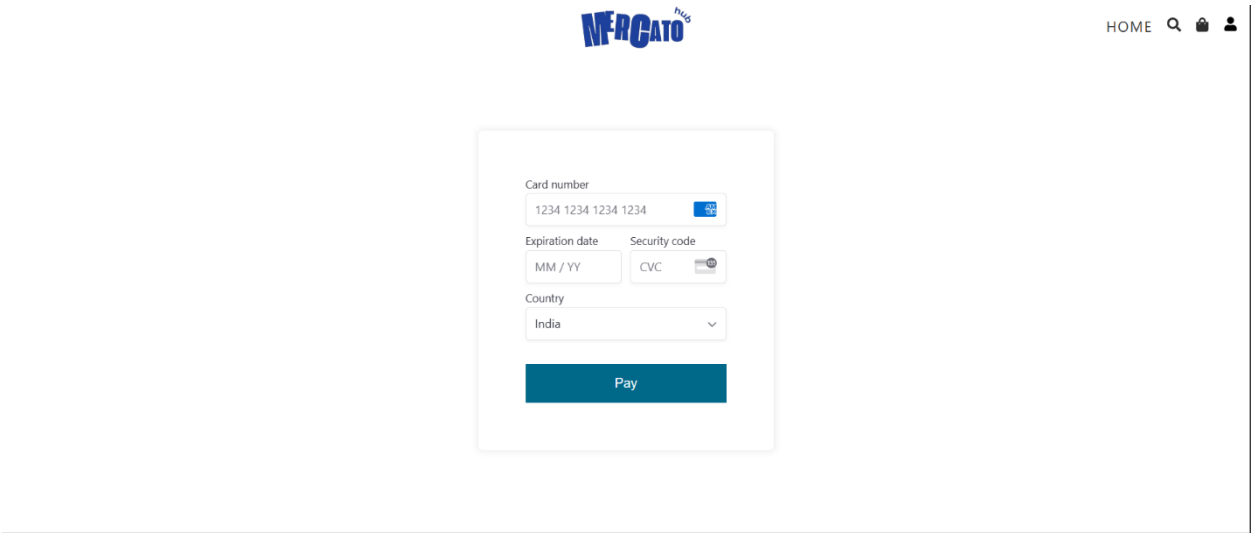

Fig 8.9 Card Payment Portal

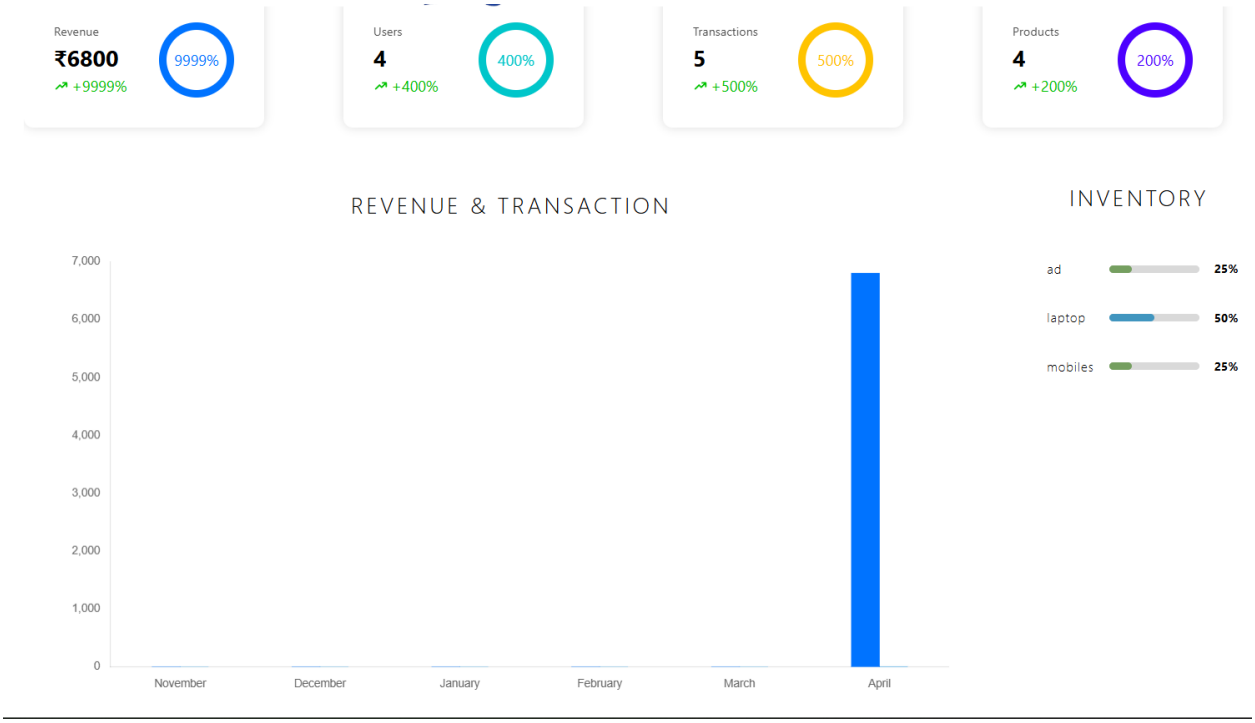

Fig 8.10 Analytics in Admin View

CHAPTER 9

CONCLUSION

The development of *MercatoHub* has been an insightful journey in building a real-world, full-stack e-commerce solution aimed at supporting small-scale businesses and local entrepreneurs. The project successfully encapsulates the technical, functional, and user-experience aspects required to create a robust, scalable, and user-friendly digital marketplace.

From ideation to deployment, *MercatoHub* was engineered with a clear objective—to bridge the gap between emerging businesses and modern consumers seeking affordable, diverse, and unique product options. By providing an intuitive and responsive frontend built with React, Vite, and TypeScript, combined with a well-structured backend developed using JavaScript and TypeScript, the platform offers seamless integration, fast performance, and modular maintainability.

Key integrations with Firebase for user authentication, Cloudinary for media management, and Redis for simulating real-time cart and payment sessions have helped simulate a real-world e-commerce experience without relying on costly third-party services. Admin and customer roles were clearly defined and implemented, ensuring the platform caters to both product managers and end users effectively.

Through well-planned system design, secure API development, user role management, dynamic product listings, and responsive UI/UX components, *MercatoHub* stands as a complete demonstration of practical software engineering practices. The use of role-based dashboards, mock payment systems, media optimization, and API integrations reflects industry standards for commercial web platforms.

MercatoHub is not only a technical implementation but also a social concept aimed at digital empowerment. While certain functionalities, such as live payment gateway integration and real-time order tracking, were mocked for demonstration, the architecture and workflow support their future inclusion.

CHAPTER 10

REFERENCES

[1] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, 1st ed., Morgan Kaufmann, 2013.

[2] M. Moroney, *Firebase Essentials – Android Edition: Learn Firebase Cloud Functions, Firestore, Firebase Auth*, Leanpub, 2017.

[3] R. Johnson, "Designing Scalable E-Commerce Systems with Redis," *ACM Digital Library*, vol. 24, no. 3, pp. 155–164, Mar. 2020. doi: 10.1145/3397211.

[4] A. Ray and A. Patel, "Performance Optimization in React Applications Using Vite and Code Splitting," *2021 IEEE Symposium on Web Technologies*, pp. 121–126, 2021. doi: 10.1109/WT50921.2021.00021.

[5] N. Rauschmayer, *Speaking JavaScript: An In-Depth Guide for Programmers*, O'Reilly Media, 2014.

[6] J. Yu and L. Tang, "Cloudinary-Based Optimization of Multimedia Content in Web Applications," *2022 International Conference on Intelligent Systems & Image Processing (ICISIP)*, Osaka, Japan, pp. 89–95, 2022. doi: 10.1109/ICISIP54578.2022.9827910.

[7] A. Kumar and S. Gupta, "An Overview of Firebase Authentication and Cloud Firestore for Scalable Web Apps," *International Journal of Computer Science and Mobile Computing (IJCSMC)*, vol. 11, no. 6, pp. 23–29, Jun. 2022.

[8] L. Zhang and Y. Wang, "Integrating Redis for Session Management and Real-Time Caching in Node.js Applications," *2020 IEEE International Conference on Cloud Computing (CLOUD)*, Singapore, pp. 142–147, 2020. doi: 10.1109/CLOUD49709.2020.00026.