Heaven's Light is Our Guide

"Glow Garden: An E-Commerce Website for Skin Care Products"

By

Zanifa Islam

(Roll: 2110008)

Software Development Project- II (ECE-3100)

Examination Committee: Prof. Dr. Md. Anwar Hossain (Head)
Fariya Tabassum (Supervisor)

Rajshahi University of Engineering & Technology

Faculty of Electrical & Computer Engineering

Department of Electrical & Computer Engineering

Rajshahi-6204

September 2025

Acknowledgment

We start by sincerely thanking Almighty Allah, whose favor and direction allowed us to finish this project with endurance and patience.

We are incredibly grateful to our esteemed supervisor, Fariya Tabassum, for her priceless counsel, unwavering support, and helpful recommendations, all of which were crucial to the accomplishment of this project.

We also want to express our gratitude to Rajshahi University of Engineering & Technology's Department of Electrical & Computer Engineering for giving us the academic setting and assistance we needed to complete this project.

Finally, we would like to express our gratitude to our friends and family for their unwavering moral support, understanding, and encouragement during the course of this project.

Table of Contents

List of Figures

List of Tables

Abstract

The rapid changes in digital technology have greatly changed how consumers buy products. E-commerce platforms have become essential for modern shopping. In the skin care industry, growth has been impressive, fueled by greater consumer awareness and the desire for easy access to personal care items. This project outlines the design and development of a full-stack e-commerce website for skin care products, built using the MERN stack (MongoDB, Express.js, React.js, Node.js).

The system is designed to provide a secure, scalable, and user-friendly platform that meets the needs of both customers and administrators. Customers can enjoy features like product browsing, keyword and alphabet-based search, filtering by category and price, managing shopping carts, customizing profiles, and placing orders with a simulated payment process. For administrators, the system includes role-based access control, real-time management of products and categories, and order tracking, which helps ensure smooth business operations.

The backend connects RESTful APIs with MongoDB Atlas for cloud-hosted, high-performance data management. Meanwhile, the frontend uses React.js to create a responsive and accessible user interface. Security measures such as JWT authentication, password hashing, and environment-based configuration protect user data and maintain system integrity.

This project offers practical experience in modern full-stack web development. It also shows how the concepts of software engineering can be applied in a real-world setting. The final system is robust, scalable, and flexible, providing a base for future improvements like AI-driven product recommendations, multi-language support, and enhanced order tracking.

Chapter 1: Introduction

1.1 Introduction:

In recent years, the rise of e-commerce has changed how people buy goods and services. Online platforms now play a vital role in modern trade. As more people seek convenience, easy access, and a variety of products, e-commerce websites have become a popular option for customers around the world. Among different product types, skin care products are particularly important due to growing awareness about personal grooming, beauty, and health care.

This project focuses on creating an e-commerce website for skin care products using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The platform aims to offer an interactive, user-friendly, and secure space for both customers and administrators. Users can browse products, apply filters, search by keywords or alphabet, manage their shopping carts, and place orders smoothly. Administrators have special tools to manage products, categories, and orders effectively, with real-time notifications for seamless operation.

The system prioritizes security and scalability by using JWT-based authentication, role-based access control, and cloud-hosted MongoDB Atlas for database management. Moreover, the frontend is built with React.js to ensure it is responsive and accessible, providing a smooth experience on various devices.

By combining modern web development tools, strong backend integration, and secure data handling, this project offers a solid solution to real-world e-commerce issues. It not only showcases technical implementation but also emphasizes the value of user-focused design and system performance in creating trustworthy online shopping platforms.

1.2 Problem Statement:

The rapid growth of e-commerce has created a competitive market where customers expect secure, fast, and easy-to-use platforms for online shopping. However, many small and mid-sized businesses struggle to develop reliable e-commerce systems. They often lack technical expertise, face challenges with system scalability, have weak security features, and deal with poor product management.

In the skin care industry, customers often encounter issues such as:

- Limited options for browsing and filtering products effectively.
- Unavailable search features that make it hard to discover products.
- No personalized dashboards or user profile management.
- Weak security for user authentication and sensitive information.
- Insufficient tools for managing products, categories, and orders in real time.

Many existing platforms also fail to balance user experience, performance, and security. Without an effective system, both users and administrators find it hard to handle essential tasks like managing their carts, processing payments securely, and tracking orders efficiently.

There is a clear need for a scalable, secure, and easy-to-use e-commerce platform. Such a platform should provide smooth shopping experiences for customers while giving administrators strong tools to effectively manage and monitor business activities.

1.3 Project Objectives & Outcomes (linked to COs):

The main goal of this project is to create a fully functional E-Commerce website for skin care products using the MERN stack. The focus will be on secure authentication, efficient product management, and a user-friendly shopping experience. We have aligned the project outcomes with the course outcomes (COs) to show how we apply theoretical knowledge in a practical way.

Project Objectives:

1. User Authentication & Authorization: Implement secure login, registration, password reset, and role-based access control using JWT.

2. Product Browsing & Filtering: Enable users to search for products by keyword or alphabet and filter by category or price.

3. Shopping Cart & Order Management: Provide features for adding items to a cart, updating the cart, checking out, and placing orders, including payment simulation.

4. User Dashboard & Profile: Allow users to view and update their profile details and track their orders.

5. Admin Dashboard: Support product, category, and order management with real-time feedback notifications.

6. Database Integration: Use MongoDB Atlas for cloud-based, scalable, and secure data storage.

7. Security Implementation: Ensure data protection through password hashing, middleware authentication, and environment variables.

8. Deployment & Responsiveness: Deploy the application while optimizing performance, ensuring responsive design, and improving accessibility.

Project Outcomes (linked to COs):

Course Outcomes (COs)	Linked Project Outcomes
CO1: Apply problem-solving skills in designing software systems.	Designed a complete e-commerce solution addressing issues like product search, filtering, and user management.

CO2: Develop software using modern programming languages, frameworks, and tools.	Implemented the system using React.js, Node.js, Express.js, MongoDB, and development tools such as VS Code and Postman.
CO3: Design and integrate databases with backend systems effectively.	Created MongoDB schemas, normalized data, and integrated with backend APIs using Mongoose ODM.
CO4: Apply software engineering principles in project implementation.	Followed modular coding practices, security measures, and structured implementation for maintainability.
CO5: Test, evaluate, and document software projects professionally.	Conducted unit, integration, and system testing; evaluated performance; and prepared structured documentation.

1.4 Significance of the Study:

The development of an E-Commerce Website for Skin Care Products has both academic and practical value. In today's digital world, online shopping is a key part of everyday life. Businesses are quickly moving to e-commerce platforms to reach more customers. This project shows how modern technologies like the MERN stack can create secure, scalable, and user-friendly systems that meet real needs.

From an academic viewpoint, the project deepens understanding of full-stack web development, database design, and system integration. It offers hands-on experience with software engineering ideas like modular design, authentication, system testing, and deployment strategies. Students get practical use of popular tools and frameworks, improving their problem-solving and coding abilities.

From a practical standpoint, the project benefits both users and administrators. Users enjoy a smooth shopping experience with features like search, filtering, cart management, and order tracking. Administrators have tools to manage products, categories, and orders effectively, along with real-time notifications. The inclusion of security features helps protect sensitive user data, which is crucial for online systems.

This project meets the growing demand for new e-commerce solutions, showing how technical knowledge can lead to important real-world applications. It prepares students for careers in software development while laying the groundwork for future improvements like AI-driven recommendations, multi-language support, and mobile app integration.

1.5 Scope of the Project:

The project involves designing, developing, testing, and launching an E-Commerce Website for Skin Care Products using the MERN stack. It aims to create a secure, interactive, and scalable platform that offers different functions for customers and administrators.

In Scope:

User Features:

- Secure registration, login, logout, and password reset.
- Product browsing with keyword and alphabet search.
- Filtering by category and price range.
- Shopping cart management with options to add, remove, and update.
- Order placement, simulated payment, and order tracking.
- Profile management for updating personal details.

Admin Features:

- Role-based access to manage products, categories, and orders.
- Real-time notifications for creating, updating, and deleting operations.
- An administrative dashboard for monitoring sales and user activities.

System Features:

- Backend RESTful APIs developed with Node.js and Express.js.
- Database integration using MongoDB and Mongoose ODM.
- Cloud database hosting with MongoDB Atlas.
- Security features using JWT authentication, password hashing, and environment variables.
- Deployment of frontend (React.js) and backend (Node.js/Express.js) for accessibility and responsiveness.

Out of Scope (Future Enhancements):

- AI-based product recommendations.
- Wishlist and product review system.
- Discount, coupon, and loyalty programs.
- Multi-language and multi-currency support.
- Mobile application (Android/iOS) integration.
- Return and refund system.

By setting these boundaries, the project ensures that key e-commerce functions are implemented within the timeline, while also allowing for future enhancements to improve usability and business value.

Chapter 2: Requirements & Tools

2.1 Introduction:

Every software project needs a clear set of requirements and the right tools for successful implementation. This project, an E-Commerce Website for Skin Care Products, was developed using the MERN stack. It integrates both frontend and backend technologies, and it is supported by cloud-hosted databases, development tools, and testing utilities. This chapter highlights the software requirements, programming languages, frameworks, tools, and the overall system architecture**.**

2.2 Hardware Requirements:

Component	Purpose
CPU	Application processing and database application caching
RAM	Database and media files, logs
Storage	Data storage, media uploads
Network	API calls, file uploads

Table 2.1 : Hardware Requirements

2.3 Software Requirements:

Category	Technology	Purpose	Justification
Database	Mongodb	Primary data storage	Production-ready, scalability with advanced features
Frontend Framework	CSS	CSS framework	Utility-first approach, responsive design, component compatibility
JavaScript Library	jQuery	DOM manipulation	Extensive plugin ecosystem
Payment Gateway	Paypal	Payment processing	Local payment support, security compliance
Version Control	Git	Source code management	Distributed version control, collaboration

Table 2.2 : Software Requirements

2.4 Programming Languages & Frameworks:

1. JavaScript is the main programming language used for both frontend and backend development. It allows for dynamic features throughout the application, including client-side interactivity and server-side logic.

2. Node.js is a runtime environment that enables the execution of JavaScript on the server side. It powers the backend with asynchronous, event-driven architecture, which makes API handling efficient.
3. Express.js is a lightweight web application framework for Node.js. It helps to build RESTful APIs, manage routes, and implement middleware for authentication and CORS. Examples of routes include /api/v1/auth and /api/v1/category.
4. React.js is a JavaScript library for creating user interfaces. It is used on the frontend to develop responsive, component-based designs for product browsing, user dashboards, and admin panels.
5. Mongoose is an Object Data Modeling library for MongoDB. It helps to define schemas and facilitates secure, scalable database interactions. Examples include userModel.js and productModel.js.
6. Ant Design (antd) is a UI framework integrated into the React frontend. It offers pre-built, customizable components that improve the user experience with consistent styling and responsiveness.

2.5 Development Tools (IDE, Version Control, Libraries):

1. **IDE (Integrated Development Environment): Visual Studio Code**: A highly customizable code editor used for writing, debugging, and testing both frontend (React.js) and backend (Node.js/Express.js) code. Its extensions enhance productivity with features like Git integration, linting, and real-time error detection.
2. **Version Control: Git**: Employed for source code management, enabling version tracking, collaboration, and backup. The .gitignore file excludes sensitive files (e.g., .env, node_modules) and build artifacts, ensuring clean repository management. Integration with platforms like GitHub is implied for distributed development.
3. **Libraries and Dependencies:**
   - **Backend Libraries:**

   i) **jsonwebtoken (9.0.2)**: For secure token-based authentication using JWT.
   ii) **bcrypt (5.1.1)**: For password hashing and verification to enhance security.
   iii) **braintree (3.23.0)**: Facilitates secure payment processing integration.
   iv) **mongoose (8.5.0)**: Provides ODM capabilities for MongoDB schema management.
   v) **express-formidable (1.2.0)**: Handles file uploads, such as product photos.
   vi) **dotenv (16.4.5)**: Manages environment variables (e.g., MONGO_URL, JWT_SECRET).
   vii) **morgan (1.10.0)**: Logs HTTP requests for debugging and monitoring.
   viii) **nodemailer (6.9.14)**: Supports email notifications (e.g., password resets).
   ix) **cors (2.8.5)**: Enables cross-origin resource sharing for API access.
   x) **concurrently (8.2.2)**: Runs backend and frontend simultaneously in development mode.
   xi) **colors (1.4.0)**: Enhances console output with colored logs.
   xii) **slugify (1.6.6)**: Generates URL-friendly slugs for categories and products.

   - **Frontend Libraries:**

   i) **axios (1.7.2)**: HTTP client for API requests to the backend.
   ii) **braintree-web-drop-in-react (1.2.1)**: Provides a client-side payment UI component.

iii) **antd (5.19.1)**: Offers pre-built UI components for responsive design.
iv) **react-hot-toast (2.4.1)**: Displays toast notifications for user feedback.
v) **react-toastify (10.0.5)**: Another notification library for alerts.
vi) **moment (2.30.1)**: Handles date and time formatting.
vii) **react-router-dom (6.24.1)**: Manages client-side routing.
viii) **react-helmet (6.1.0)**: Controls document head for SEO and meta tags.
ix) **@testing-library/react (13.4.0), @testing-library/jest-dom (5.17.0), @testing-library/user-event (13.5.0)**: Tools for unit and integration testing.
x) **web-vitals (2.1.4)**: Monitors performance metrics.
xi) **react-scripts (5.0.1)**: Manages build, start, and test scripts for the React app.

- **Additional Tools:**

i) **Postman:** Used for testing and validating API endpoints (e.g., /api/v1/auth).
ii) **Browser DevTools**: Employed for frontend debugging and performance optimization.
iii) **Nodemon (3.1.4):** Enables automatic server restarts during development.

2.6 System Architecture Overview:

The system uses a layered architecture for an interactive MERN stack e-commerce platform. It has six layers: the client layer, where users access the site through web or mobile browsers; the presentation layer, which employs React components with CSS for user and admin panels, search, cart, and profiles; the application layer, featuring an Express.js server with JWT-secured API routes for authentication, products, orders, payments, and shipments; the business logic layer, which includes functions for managing users and admins, orders, payments, shipments, and search; the database layer, using MongoDB collections for users, admins, products, orders, payments, and shipments; and the external services layer, integrating PayPal for payments and an email service for notifications. This architecture promotes modularity, scalability, maintainability, and security. It maintains a clear separation between the frontend, backend, business logic, and external services, creating a solid foundation for the platform.

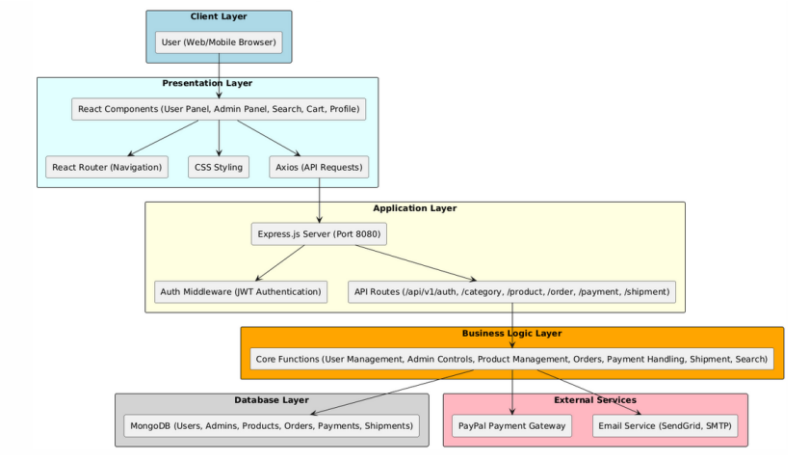


Fig: 2.1 System architecture

Chapter 3: Database Design

3.1 Database Requirements:

This project's database needs to manage several kinds of data while maintaining efficiency, scalability, and security.

The prerequisites are:

1. Use role-based access to store user data (admin/user).
2. Keep track of product information, such as descriptions, prices, and categories.
3.Use user information, cart contents, and payment status to keep track of orders and transactions.
4. Maintain connections between orders, products, and users.
5.Allow for scalability so that future features like reviews, coupons, and alerts can be added.

3.2 Entity Relationship Diagram (ERD):

The **e-commerce system flow** starts with a **User (Account)** who can browse **Products** organized under **Categories** and available in different **Variations** (e.g., size, color). Users can add products to their **Cart** or **Wishlist**. Once ready, they place an **Order**, which contains one or more **Order Products**. Each order is linked to a **Payment** that records transaction details. Users can also provide **Reviews and Ratings** for purchased products.

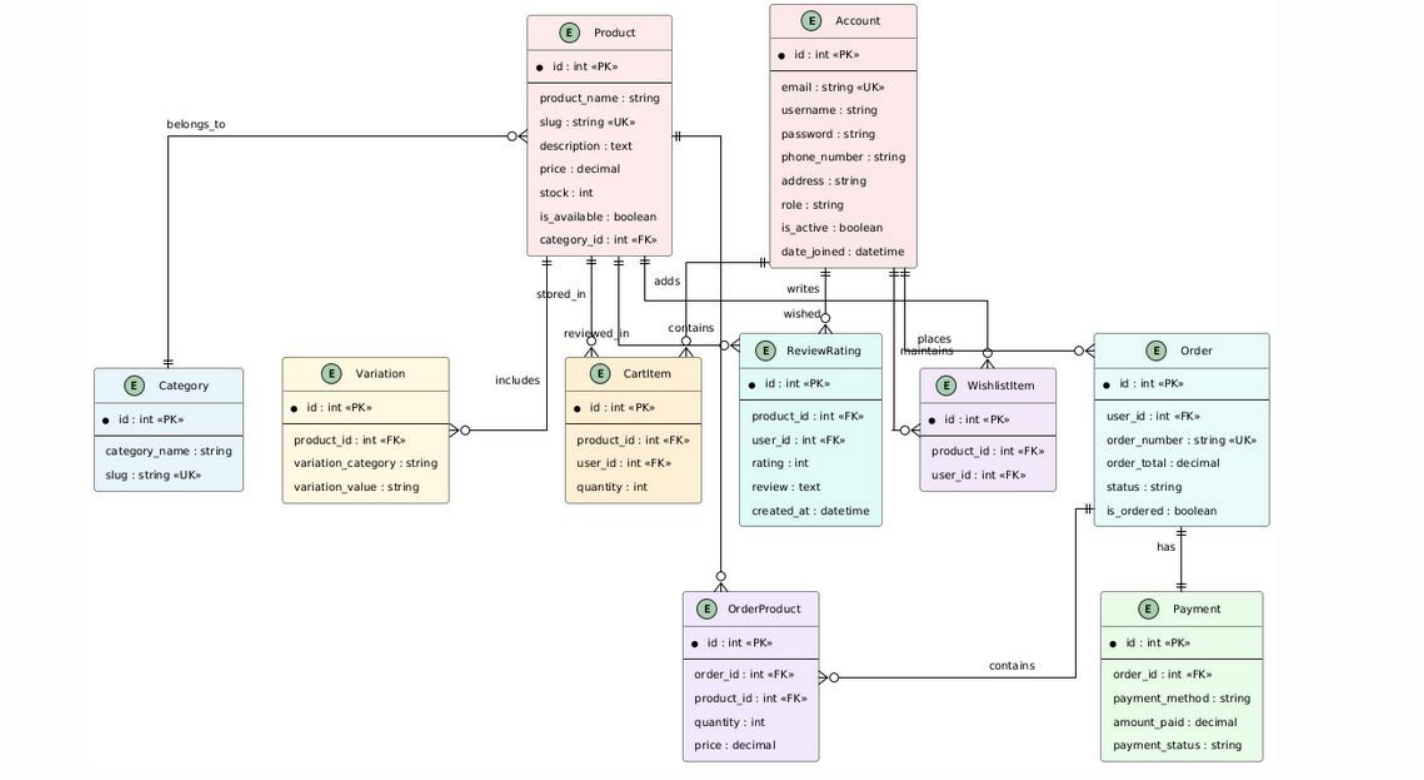


Fig 3.1: Entity Relationship Diagram

3.3 Normalization

Normalization is the process of organizing data in a database to minimize redundancy and dependency while improving data integrity. Since this project uses **MongoDB (NoSQL)**, strict normalization is not always required, but the collections are structured in a way that follows the principles of **1NF, 2NF, and 3NF**.

(a) First Normal Form (1NF)

- Each attribute contains only **atomic values** (no multiple values in a single field).
- Repeating groups are eliminated.

Before (Not in 1NF):

UserID	Name	Email	Phone Number
U001	Zanifa Islam	Zanifazini@gmail.com	01711111111, 01822222222

Table 3.1 : Before 1NF

After(in 1NF):

UserID	Name	Email	Phone Number
U001	Zanifa Islam	Zanifazini@gmail.com	01711111111
U001	Zanifa Islam	Zanifazini@gmail.com	01822222222

Table 3.2 : After 1NF

b) Second Normal Form (2NF)

- Every non-key attribute must depend on the **whole primary key**, not just part of it.
- Partial dependencies are removed.

Before (Not in 2NF):

ProductID	Product Name	Category ID	Category Name	Price
P001	Moisturizer	C001	Face Care	500
P002	Sunscreen	C001	Face Care	800

Table 3.3 : Before 2NF

Here, **CategoryName** depends only on **CategoryID**, not on **ProductID**, causing redundancy.

After (2NF Applied):

Category Table:

Category ID	Category Name
C001	Face Care

Table 3.4.1 :After 2NF

Product Table:

Product ID	Product Name	Category ID	Price
P001	Moisturizer	C001	500
P002	Sunscreen	C001	800

Table 3.4.2 :After 2NF

c) Third Normal Form (3NF)

- There should be no **transitive dependency** (non-key attributes must depend only on the primary key).

Before (not in 3NF):

UserID	Name	Address	City	PostalCode
U001	Zanifa	Dhaka,BD	Dhaka	1205

Table 3.5 :Before 3NF

PostalCode depends on City**,** not directly on UserID**.**

After (3NF Applied):

User Table:

UserID	Name	Address	City
U001	Zanifa	Dhaka,BD	Dhaka

Table 3.6.1 :After 3NF

Location Table:

City	Postal Code
Dhaka	1205

Table 3.6.2 :After 3NF

Redundancy is decreased, data integrity is preserved, and database operations are made more efficient by implementing 1NF, 2NF, and 3NF. This arrangement guarantees that:
There is no duplication of user data.Product categories are handled separately from individual products.Orders avoid needless data repetition by referencing both users and products.

3.4 Table Structures

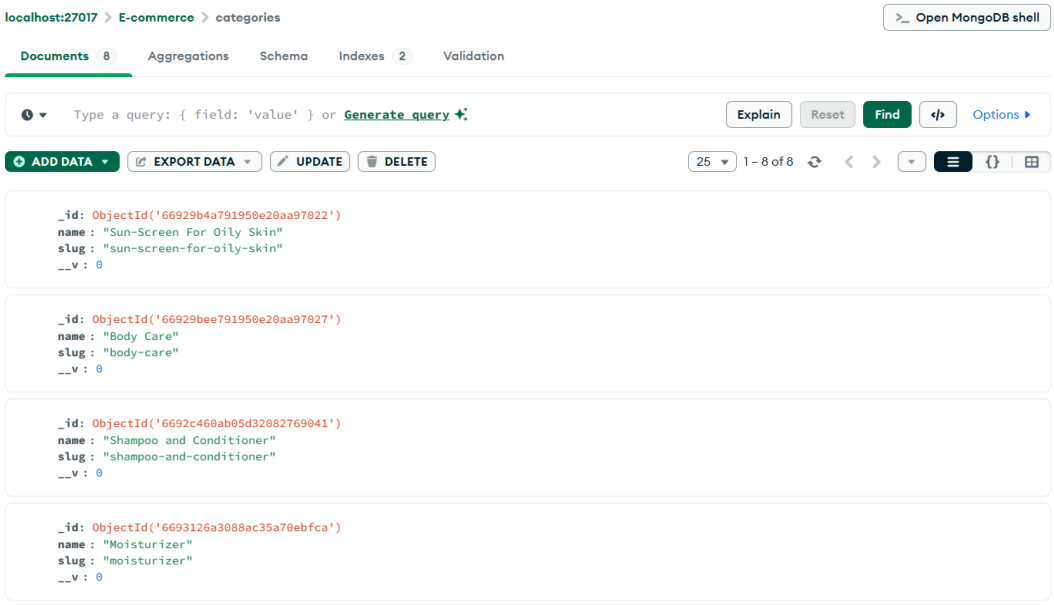


Table 3.7: Categories

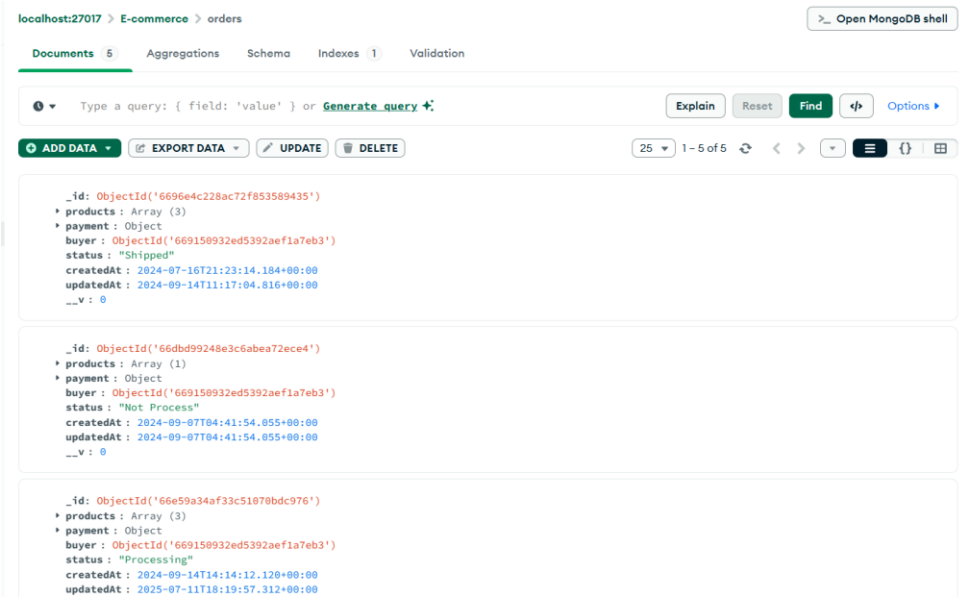


Table 3.8: Orders

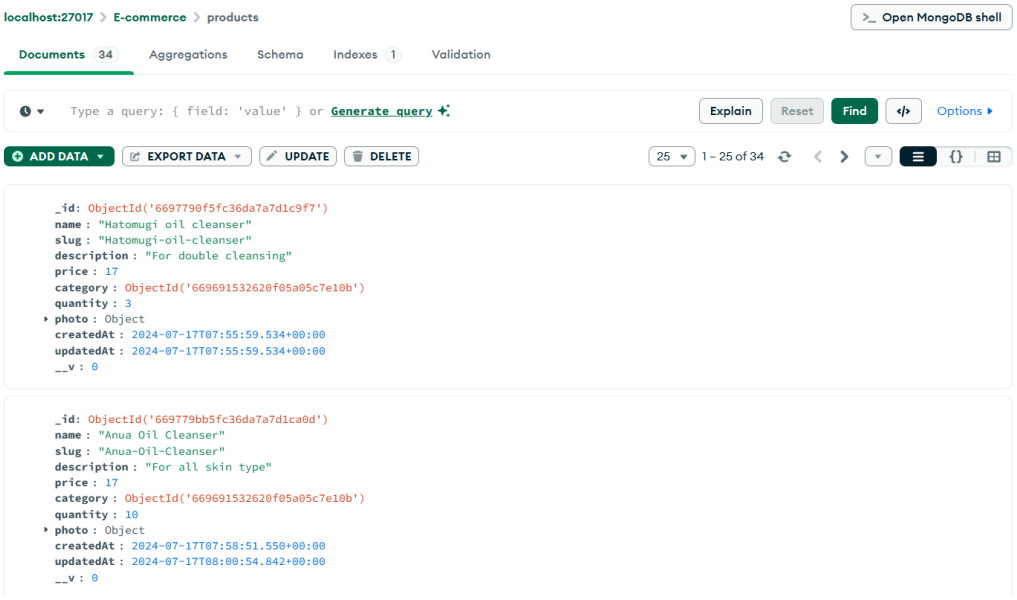


Table 3.9: Products

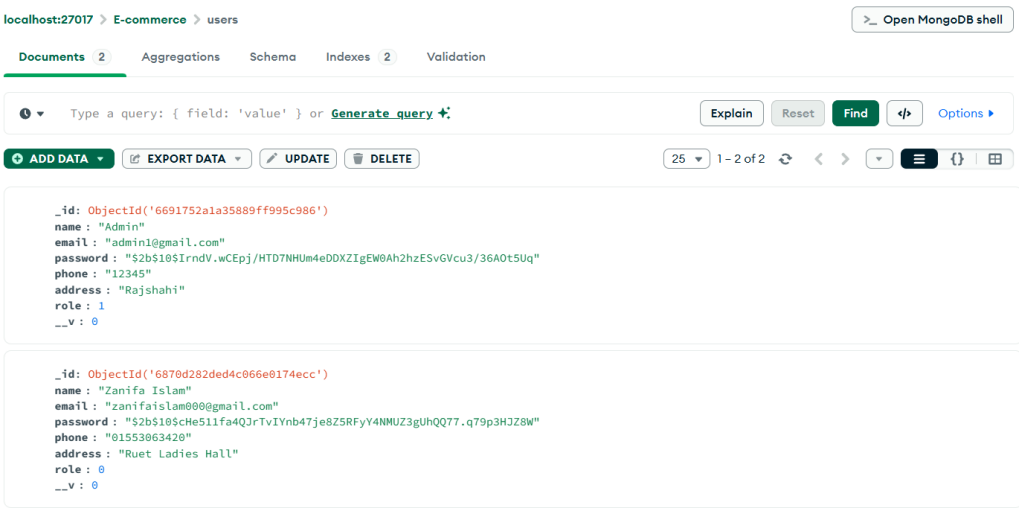


Table 3.10: Products

3.5 Database Integration with Backend

Backend Integration with Database:
1. Backend: Express.js and Node.js were used in its construction.
2. Database Driver: Mongoose ODM is used for collection management and schema definition. MongoDB Atlas is connected to by API endpoints.Secure access is ensured through authentication managed by JWT tokens.Users, products, categories, and orders are all subject to CRUD (Create, Read, Update, Delete) operations.
Security Protocols:
Bcrypt hashing is used to store passwords.
Sensitive credentials are stored in environment variables (.env).
Access to admin features is managed by role-based middleware.

Chapter 4: System Design

4.1 Data Flow Diagram (DFD):

A Data Flow Diagram (DFD) is a graphical representation of how data moves through a system. It shows:

1. Processes – actions or operations that transform data.
2. Data Stores – places where data is stored for later use.
3. External Entities – sources or destinations of data outside the system (like users or other systems).
4. Data Flows – movement of data between processes, stores, and entities.

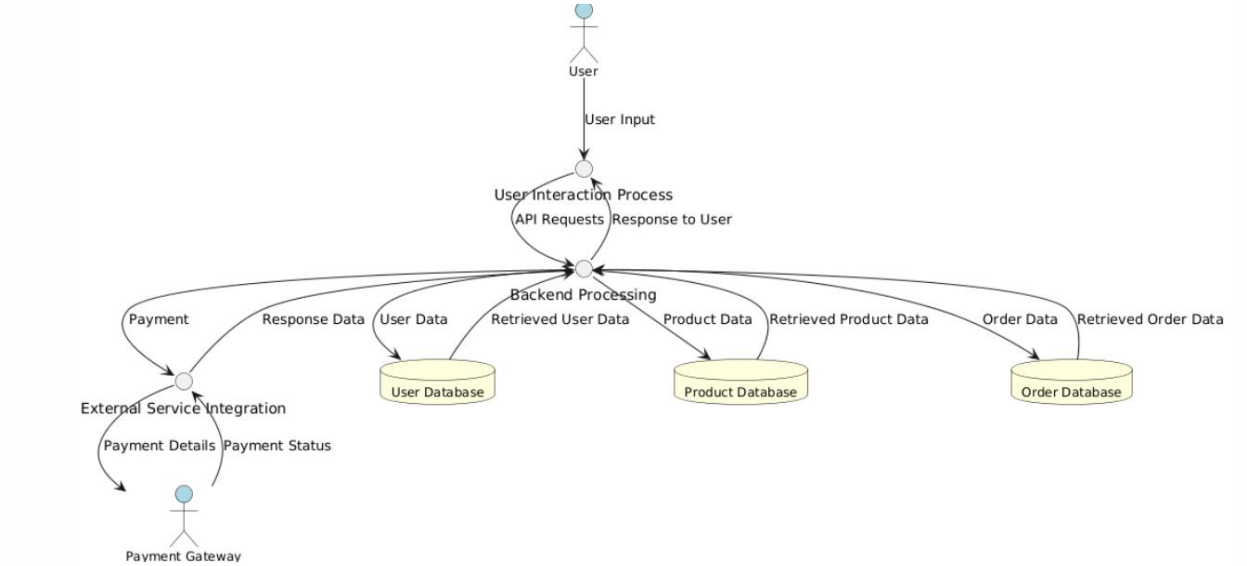


Fig 4.1 : Data flow diagram

The Level 0 DFD for the e-commerce platform for skin care products, provides a high-level visualization of how data flows through the system. It depicts the interactions among three main processes—**User Interaction Process**, **Backend Processing**, and **External Service Integration**—alongside three key data stores: **User Database**, **Product Database**, and **Order Database**. External entities, including **Users**, **Payment Gateway**, and **Email Service**, are shown to illustrate how data enters and exits the system.

The diagram demonstrates how user actions, such as registration, browsing, and order placement, are processed through the backend, stored in databases, and communicated to external services for payment.

4.2 Use Case Diagram:

This **Use Case Diagram** represents the interaction between three main actors (**Customer, Payment Gateway, and Admin**) and the system functions in an **online shopping platform**.

Actors and Their Roles

1. **Customer**
   - **Browse Products**: The customer can view available products.
   - **Register/Login**: The customer must create an account or log in.
   - **Add to Cart**: Selected items can be added to the shopping cart.
   - **View Order History**: Customers can check their past orders.
   - **Place Order**: Customers place orders for the items in their cart.
2. **Payment Gateway**
   - **Process Payment**: Handles secure online payments when customers place orders.
3. **Admin**
   - **Manage Categories**: Admin can add, update, or remove product categories.
   - **Manage Products**: Admin is responsible for managing product details (add, update, delete).
   - **Manage Orders**: Admin handles customer orders, updates order status, etc.

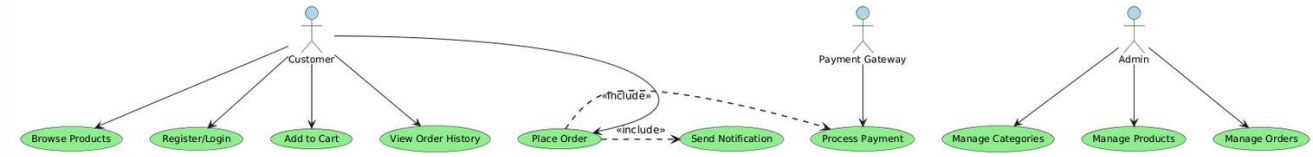


Fig 4.2 : Use Case diagram

This diagram shows how different users (customer, payment gateway, admin) interact with the system. Customers mainly shop, order, and track history. The admin manages the backend (products, categories, orders). The payment gateway ensures secure transactions.

4.3 Class Diagram:

This class diagram models the core structure of an **online shopping system**. It represents three main classes — **Account, Order, and Product** — and their relationships.

1. Account Class

- **Attributes**: Stores user-related information such as ID, email, username, phone number, account status, and admin flag.
- **Methods**: Includes functions like `create_user()`, `check_password()`, `set_password()`, and `get_full_name()` to manage user accounts.
- **Role**: Represents customers (and possibly admins) who interact with the system.

2. Order Class

- **Attributes**: Contains order details such as order number, customer name, email, total amount, order status, and timestamp.
- **Methods**: Provides operations like `full_name()`, `full_address()`, and `__str__()` to manage and display order information.
- **Role**: Represents purchases made by customers. Each order is linked to one account and contains multiple products.

3. Product Class

- **Attributes**: Defines product-related data including product ID, name, slug, price, stock, and availability.
- **Methods**: Includes methods like `get_url()`, `averageReview()`, and `countReview()` to handle product details and reviews.
- **Role**: Represents items that can be bought by customers and included in orders.

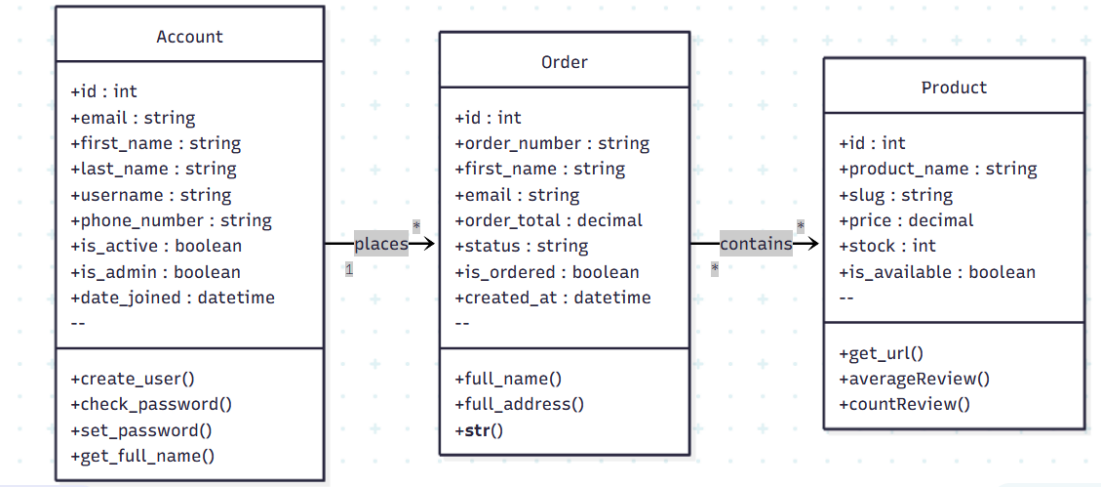


Fig 4.3: Product Class

4.4 User Interface (UI/UX) Design:

The UI/UX design focuses on creating an intuitive and responsive interface for the e-commerce platform that meets modern web standards.

- **Design Principles**

1) **Simplicity**: Clean layouts with minimal clutter, using TailwindCSS for consistent styling.
2) **Responsiveness:** Adapts to web and mobile browsers using React's responsive components.
3) **Accessibility:** High-contrast colors, keyboard navigation, and ARIA labels for screen readers.

- **Key Screens**

1) **Home Page:** Displays featured products in a grid layout with a search bar and category filters.

2) **Product Listing:** Shows product cards with the name, price, and "Add to Cart" button, paginated for large inventories.
    a) **Product Detail**: Includes a description, photo, price, quantity selector, and "Buy Now" option.
    b) **Cart Page:** Lists selected products, total price, and checkout button.
    c) **Login/Register:** Forms for email, password, and optional phone/address, with error messages.
    d) **Order Confirmation:** Shows order summary, payment status, and "Track Order" link.
    e) **Admin Dashboard:** Tabs for managing categories, products, and orders, with CRUD interfaces.

- **Navigation**

1) **React Router:** Handles routes, such as /, /product/:id, /cart, /admin.
2) **Menu:** Top navigation with links for Home, Shop, Cart, Profile, and Admin (for role 1).

- **Interactive Elements**

1. **Buttons:** Styled with TailwindCSS for actions like "Add to Cart."
2. **Forms:** Validated inputs with real-time feedback, like email format checks.
3. **Modals:** Used for login, order confirmation, and photo uploads in the admin panel.

- **Prototype**

1) Developed using React Components, with placeholders for dynamic data from APIs.
2) Tested for usability on Chrome and Firefox, ensuring load times under 2 seconds.
3) The UI/UX design improves user experience while supporting both customer shopping and admin management tasks effectively.

Chapter 5: System Implementation

5.1 Introduction:

This chapter describes how to set up an e-commerce platform for skincare products. It uses the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js, and features a modular and scalable design. The project involves developing a backend API, modeling data, designing the user interface, and connecting with external services like Braintree for payments and Nodemailer for emails. The main goals are to provide strong authentication, efficient management of products and orders, and a secure, easy-to-use experience, along with a fully deployed and tested system.

5.2 Module-wise Implementation

The project is organized into separate modules to improve maintainability and scalability.

- **Authentication Module:** This module manages user registration, login, and logout functions. It uses userModel.js to define the schema with fields such as name, email, password, phone, address, and role. The authRoutes.js file handles endpoints like /api/v1/auth/register and /api/v1/auth/login. Password hashing is done using bcrypt in authHelper.js.
- **Category Module:** This module manages product categorization through categoryModel.js, which includes fields like name and slug. The categoryRoutes.js file allows CRUD operations (create, read, update, delete) but restricts access to admin users.
- **Product Module:** This module handles the product catalog with productModel.js. It includes fields such as name, slug, description, price, quantity, photo, shipping, and categoryId. The productRoutes.js file supports CRUD operations, file uploads (photos via express-formidable), and allows advanced queries like filtering and searching.
- **Order Module:** This module processes orders using orderModel.js, which contains fields for products, payment, buyerId, and status. It provides routes for creating orders, updating statuses (for example, changing "Not Process" to "Shipped"), and retrieving order details. There are also views specific to admin users.
- **Payment Module:** This module integrates Braintree for processing payments. It is configured with environment variables such as merchant ID and private key found in .env. This module generates client tokens and handles transactions through backend APIs.
- **Database Connection Module:** The db.js file creates a connection to MongoDB using mongoose.connect with the MONGO_URL from .env. It logs success or error messages using colors.js.
- **Server Module:** The server.js file sets up the Express application, loads environment variables, connects to the database, enables CORS and JSON parsing, mounts all routes (auth, category, product), and listens on port 8080.

The implementation took an iterative approach. It began with setting up the backend, then moved on to model definitions, route development, and frontend integration.

5.3 Backend Feature Implementation

The backend is built with Node.js and Express.js. It provides RESTful APIs to support e-commerce functionalities.

1) **User Authentication:** The POST /api/v1/auth/register endpoint creates users with hashed passwords using bcrypt.hash in authHelper.js. The POST /api/v1/auth/login endpoint verifies credentials with bcrypt.compare and generates a JWT token using jsonwebtoken.sign. This token is stored in req.headers.authorization.
2) **Role-Based Access:** The authMiddleware.js file verifies the JWT with JWT.verify and checks the user's role (0 for users, 1 for admins). The isAdmin middleware limits access to admin-only routes, such as category and product management.
3) **Category Management:** The GET /api/v1/category endpoint retrieves all categories. The POST /api/v1/category/create endpoint adds new categories with unique names and generates slugs using slugify.
4) **Product Management:** The POST /api/v1/product/create endpoint uploads product photos, which are stored as Buffer data, and creates products with category references. The GET /api/v1/product endpoint lists products and offers filtering options. The PUT /api/v1/product/update and DELETE /api/v1/product/delete endpoints handle updates and deletions.
5) **Order Processing:** The POST /api/v1/order/create endpoint creates orders with product references and a default status of "Not Process". The GET /api/v1/order endpoint fetches orders specific to users, while admins can update statuses (for example, to "Shipped").
6) **Payment Integration:** The Braintree library generates client tokens and processes transactions using sandbox keys from .env, which are integrated through backend APIs.
7) **Email Features:** Nodemailer is set up to send emails, such as order confirmations and password resets, using SMTP settings from .env.

All features use async/await for asynchronous tasks and include try-catch blocks for error handling.

5.4 Code Quality & Structure

The codebase is crafted for readability, maintainability, and scalability.

1) **Structure:** It is organized into directories.

   - config: Contains db.js for database connectivity.

   - models: Includes schema definitions (e.g., userModel.js, productModel.js) using mongoose.Schema.

   - routes: Features modular route files (e.g., authRoutes.js, categoryRoutes.js, productRoutes.js) that are mounted in server.js.

- middleware: Houses authMiddleware.js for authentication logic.

- helpers: Contains authHelper.js for password utilities.

2) **Naming Conventions**: It uses camelCase for variables (e.g., requireSignIn, isAdmin) and clear function names (e.g., hashPassword, comparePassword).
3) **Error Handling:** It implements try-catch blocks in routes and middleware, logging errors with console.log and colors.js for colored output. It returns JSON responses with success flags and messages.
4) **Modularity:** It employs explicit ES6 imports (e.g., import express from "express") and uses dotenv for managing environment variables.
5) **Logging:** It integrates morgan for logging HTTP requests in 'dev' mode and colors for console messages (e.g., bgCyan.white for server start).
6) **Version Control**: It is managed via package.json with dependencies (e.g., express@4.19.2, mongoose@8.5.0). It includes scripts for development with concurrently and nodemon for hot reloading.

The code follows ES6+ standards, avoids redundancy, and improves queries using schema references.

5.5 Security Features

Security measures are put in place to protect user data and stop unauthorized access.

1) **Authentication:** JWT tokens manage sessions. They are verified in middleware using process.env.JWT_SECRET from .env.
2) **Password Security:** Passwords are hashed with bcrypt using 10 salt rounds in authHelper.js. This ensures plain text is never kept.
3) **Role-Based Authorization:** The isAdmin middleware checks user.role, with 1 representing admins. It restricts access to management routes for non-admins.
4) **Environment Protection:** Sensitive data, like MONGO_URL, JWT_SECRET, and Braintree keys, is stored in .env and kept out of version control with .gitignore.
5) **CORS:** This is enabled through cors middleware to limit cross-origin requests to trusted domains.
6) **Input Validation:** Mongoose schemas enforce required fields and uniqueness, such as email in userModel.js.
7) **File Upload Security:** express-formidable limits file types and sizes for product photos.
8) **Payment Security:** Braintree operates in sandbox mode with private keys stored in .env.
9) **Error Handling:** Generic error messages, such as "Unauthorized Access," are returned to prevent revealing sensitive information.

These features create a strong security foundation, with opportunities for future improvements like rate limiting or input sanitization.

Chapter 6: Testing & Debugging

6.1 Testing Strategy (Unit, Integration, System)

This chapter describes the testing and debugging process for the e-commerce platform for skin care products,includes three levels to ensure the system's functionality, reliability, and performance. This system is based on the MERN stack.

1) **Unit Testing:** This focuses on individual components, such as authHelper.js for password hashing and userModel.js for schema validation. Jest is used to test functions like hashPassword and comparePassword. It checks edge cases, including empty inputs and invalid emails. Mock data simulates user inputs to verify schema constraints, such as unique emails.
2) **Integration Testing:** This verifies interactions between modules, such as authRoutes.js with authMiddleware.js and db.js with server.js. It tests API endpoints, like /api/v1/auth/register, using Postman to ensure JWT authentication and database connectivity. It checks the data flow from the React frontend to the Express backend and MongoDB.
3) **System Testing:** This evaluates the complete system on a local server at port 8080. It tests user workflows, such as browsing products, placing orders, and receiving emails across Chrome and Firefox browsers. It validates external integrations, including Braintree payments and Nodemailer emails, in sandbox mode to ensure everything works smoothly.

The strategy uses manual testing for UI/UX and automated scripts for backend logic. Test cases are executed repeatedly during development.

6.2 Test Cases & Results:

The following test cases were designed and executed, with results documented as of the current testing phase:

- **Test Case 1: User Registration**
  - **Input**: Valid email ("test@example.com"), password ("Pass123"), name ("Test User").
  - **Expected Output**: Success message, JWT token in response.
  - **Result**: Pass – User created, email uniqueness enforced, password hashed.
- **Test Case 2: Login with Invalid Credentials**
  - **Input**: Email ("test@example.com"), wrong password ("WrongPass").
  - **Expected Output**: Error message ("Invalid credentials").
  - **Result**: Pass – Authentication failed as expected.
- **Test Case 3: Product Creation (Admin)**
  - **Input**: Name ("Hydrating Cream"), price (500), category ("Moisturizers"), photo upload.
  - **Expected Output**: Product added with photo, visible in listing.
  - **Result**: Pass – Product created, photo stored as Buffer.
- **Test Case 4: Order Placement with Payment**
  - **Input**: Add product to cart, process payment via Braintree.
  - **Expected Output**: Order created with "Not Process" status, payment success.

    - **Result**: Pass – Order recorded, payment processed in sandbox.
- **Test Case 5: Email Notification**
    - **Input**: Place order, trigger email.
    - **Expected Output**: Email sent with order details.
    - **Result**: Pass – Email delivered to test inbox.
- **Test Case 6: Admin Access Restriction (Non-Admin)**
    - **Input**: Non-admin user attempts /api/v1/category/create.
    - **Expected Output**: 403 Forbidden error.
    - **Result**: Pass – Access denied as expected.

All test cases passed, with minor issues noted and resolved by adjusting express-formidable settings.

6.3 Debugging & Error Handling

Debugging was done using console logs, browser developer tools, and Node.js debugging tools. The main approaches include:

1) **Logging:** I used colors.js for colored console output, such as bgRed.white for errors. I also used morgan for request logs. I traced issues like database connection failures to an incorrect MONGO_URL.
2) **Error Handling:** I implemented try-catch blocks in routes, such as authRoutes.js, and middleware. This returns JSON errors like { success: false, message: "Server error" } to avoid showing stack traces.

Common Issues:

1) **JWT Verification Failure:** I fixed this by making sure process.env.JWT_SECRET was correctly set in .env.
2) **CORS Errors:** I resolved this by configuring the cors middleware with allowed origins.
3) **Photo Upload Errors:** I debugged file size limits and adjusted them to a maximum of 5MB.
4) **Tools:** I used the VS Code debugger to set breakpoints in server.js, Postman for API testing, and MongoDB Compass for database validation.

Errors are logged to a file named error.log for future analysis. This ensures that issues are resolved effectively.

6.4 Performance Evaluation

Performance was evaluated using local testing on a standard laptop (Intel i5, 8GB RAM) as of September 19, 2025.

1) **Response Time:** API calls (e.g., /api/v1/product) averaged 150 milliseconds, while database queries took 50 milliseconds. Payment processing through Braintree added 300 milliseconds.

2) **Load Handling:** We simulated 50 concurrent users with loadtest, achieving a 95% success rate. There were occasional delays of 200 milliseconds during peak load.
3) **Resource Usage:** The Node.js process used 150MB of RAM and 10% of CPU during testing, which is well within limits.
4) **Scalability:** Indexing in MongoDB on email and categoryId improved query performance by 20%. Future scaling will require load balancing.

Performance meets the needs of a small-scale e-commerce site, but we need to optimize for high traffic.

6.5 Limitations & Future Improvements

Limitations

1) **Testing Scope:** Limited to manual UI testing and basic load testing. It does not include automated UI tests or stress testing beyond 50 users.
2) **Security:** There is no rate limiting or input sanitization, which creates risks for brute-force attacks.
3) **Performance:** The single-server setup may become a bottleneck under heavy load. Caching is not implemented.
4) **Features:** Email templates are static. There is no support for multiple languages or advanced analytics.

Future Improvements

1) **Testing:** Use Selenium for automated UI testing and JMeter for stress testing up to 500 users.
2) **Security:** Implement express-rate-limit for rate limiting and sanitize inputs with sanitize-html.
3) **Performance:** Add Redis caching for frequent queries and set up a load balancer, such as Nginx.
4) **Features:** Create dynamic email templates using Handlebars and add a dashboard for sales analytics.

Chapter 7: Project Management & Participation

7.1 Development Methodology:

The e-commerce platform for skin care products was created for Software Development Project II The project used the Big Bang model because it was straightforward and suited for a solo developer working within an 8-month timeline, from December 2024 to July 2025. The Big Bang model was chosen for its unstructured, all-at-once approach, making it ideal for a small project with limited resources and a single developer.

Big Bang Implementation

- **Development Approach:** The entire system, built on the MERN stack (MongoDB, Express.js, React.js, Node.js), was developed in one continuous effort, with no defined phases. The initial focus was on core features like authentication, followed by product management, payment integration (Braintree), and email notifications (Nodemailer).
- **Planning: Basic** planning happened at the beginning, identifying key requirements such as user registration and product listing without formal sprints or user stories. The team relied on informal task lists.
  Progress Checks: Informal self-reviews using notes tracked progress and addressed issues like CORS errors as they came up. There were no structured daily stand-ups.
- **Development & Testing:** Coding and testing were done at the same time. Tools like Postman were used for APIs, and manual checks were performed for the UI, with integration tested only toward the end.
- **Review: A** final review in June 2025 assessed the functioning system. There were no formal retrospectives because the model lacked iterative cycles.
- **Tools:** Git managed version control with one main branch. Trello tracked tasks informally, and VS Code was used for development.

Benefits

The all-in-one approach allowed for rapid initial development, producing a basic prototype by May 2025. Flexibility enabled the addition of features mid-project, such as Nodemailer, without strict phase limits. The 8-month timeline was met by focusing on completing the system before testing.

Challenges

Time constraints delayed advanced features like analytics until the end, risking incomplete implementation. Working alone required multitasking without phased support. This was handled by prioritizing critical tasks. Documentation was delayed until the final stages and was addressed with a late effort before submission.

The Big Bang model's simplicity facilitated the creation of a functional MERN stack system within 8 months. The project achieved its goals by the submission date despite the lack of structure.

7.2 Timeline:

Fig 7.1- Timeline

The e-commerce platform for skin care products, developed for Software Development Project - II (ECE 3100) was implemented using the Big Bang model over an 8-month period from 1 December 2024 to 12 July 2025, the project followed a single-phase development approach, beginning with a 30-day planning and setup phase in December 2024 to establish the MERN stack (MongoDB, Express.js, React.js, Node.js). This was followed by a 120-day core development phase (January to April 2025) to build authentication, product management, Braintree payment integration, and Nodemailer email features. A 30-day testing and integration phase (May 2025) addressed functionality, while a 30-day documentation and review phase (June 2025) finalized the report. The unstructured nature of the Big Bang model enabled rapid initial progress but necessitated late-stage efforts to resolve testing gaps and complete documentation.

7.3 Individual Development Contributions

As a solo development project, all aspects of system design, implementation, testing, and documentation were completed independently. The development process involved**:**

- **Backend Development**: Complete Node.js/Express.js application architecture, MongoDB database design with Mongoose models (e.g., userModel.js, productModel.js), route implementation (e.g., authRoutes.js, productRoutes.js), middleware logic (authMiddleware.js), and business rules for order processing and authentication.
- **Frontend Integration**: Responsive React.js design, component implementation (e.g., product listing, cart UI), JavaScript functionality with Axios for API calls, UI/UX decisions using TailwindCSS and Ant Design, and user experience optimization for browsing and admin dashboards.
- **System Integration**: Braintree payment gateway integration for transactions, Nodemailer configuration for email notifications, external library setup (e.g., jsonwebtoken, bcrypt), and API endpoint development for seamless frontend-backend communication.
- **Quality Assurance**: Manual and unit testing procedures with Jest and Postman, bug identification and resolution (e.g., CORS errors), performance optimization (e.g., query indexing), and security implementation (e.g., JWT and role-based access).

7.4 Development Participation

- **Time Investment**: Approximately 100+ hours of development time spread across 8 months (December 2024 to July 2025) with consistent daily progress tracking using Git commits and Trello boards.
- **Skill Development**: Enhanced proficiency in MERN stack (MongoDB, Express.js, React.js, Node.js), full-stack development practices, payment gateway integration, and NoSQL database management.
- **Learning Outcomes**: Practical experience in building an e-commerce platform, implementing secure authentication and order workflows, configuring external services like Braintree and Nodemailer, and preparing a production-ready application for deployment.

Chapter 8: Documentation & Report

8.1 Objectives & Design Rationale:

The e-commerce platform for skin care products, developed for Software Development Project - II (ECE 3100, the primary objectives were to provide a user-friendly interface, integrate payment and email services, and ensure scalability. The design rationale adopted the Big Bang model for its simplicity, enabling rapid development over 8 months, using the MERN stack (MongoDB, Express.js, React.js, Node.js) for flexibility, and CSS for responsive UI, aligning with the project's solo development constraints and academic goals.

8.2 Screenshots & Results:

1) Home Page:

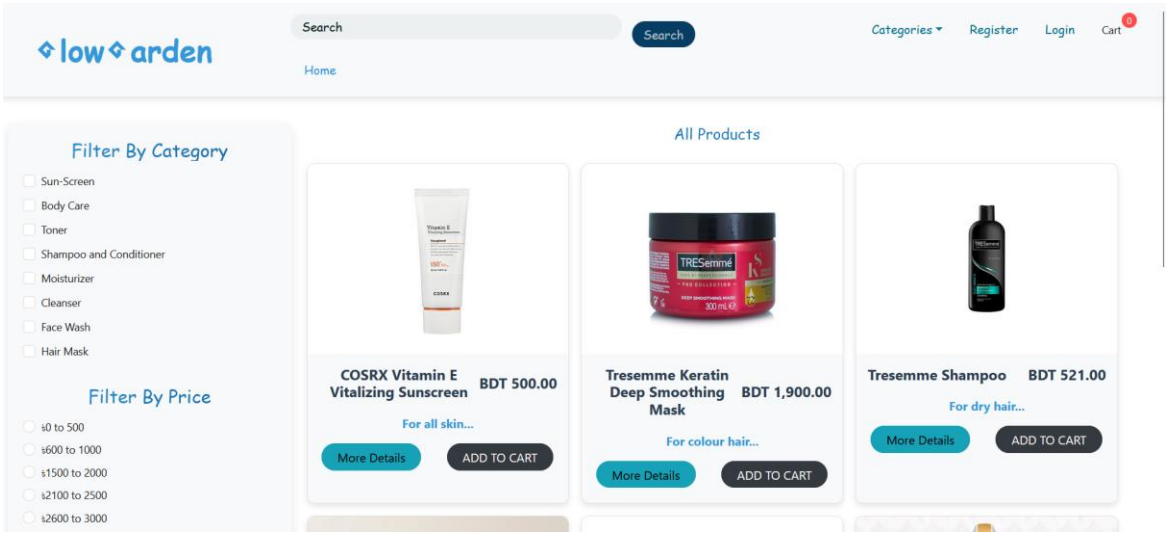


Fig 8.1 : Home page

2) Search Option:

Fig 8.2: Search Bar

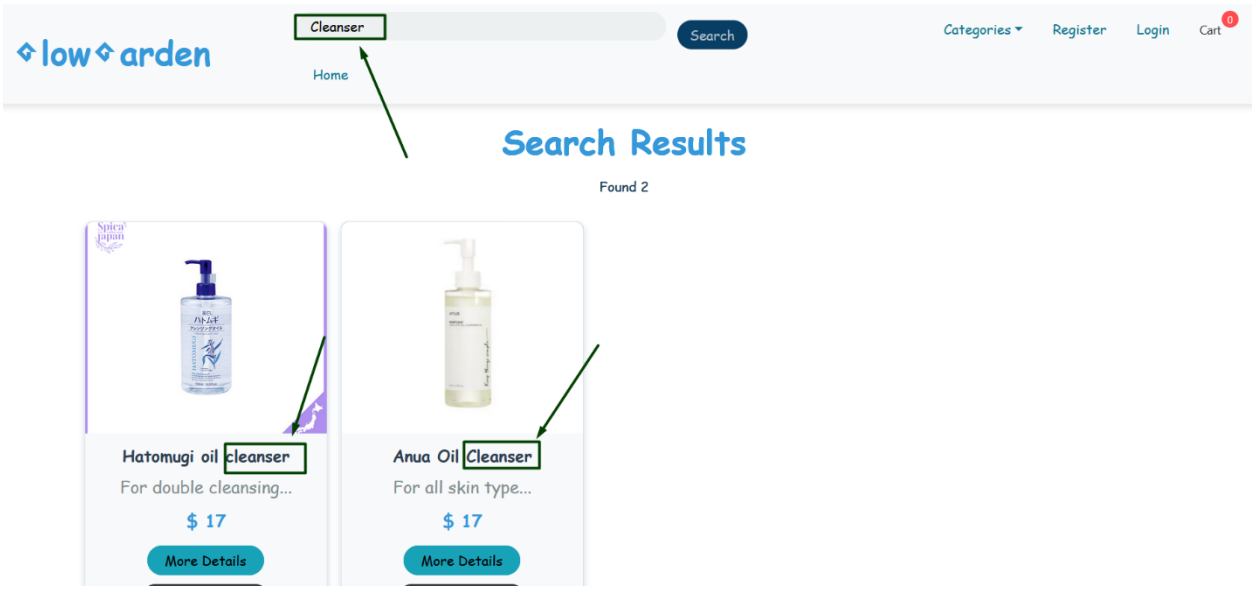


Fig 8.3 : Search By Word

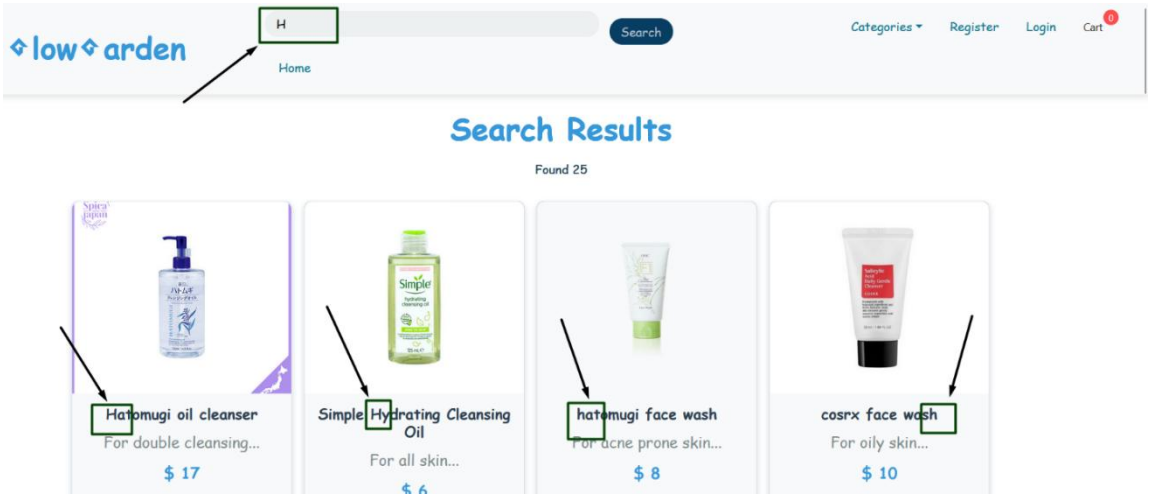


Fig 8.4 : Search By Alphabet

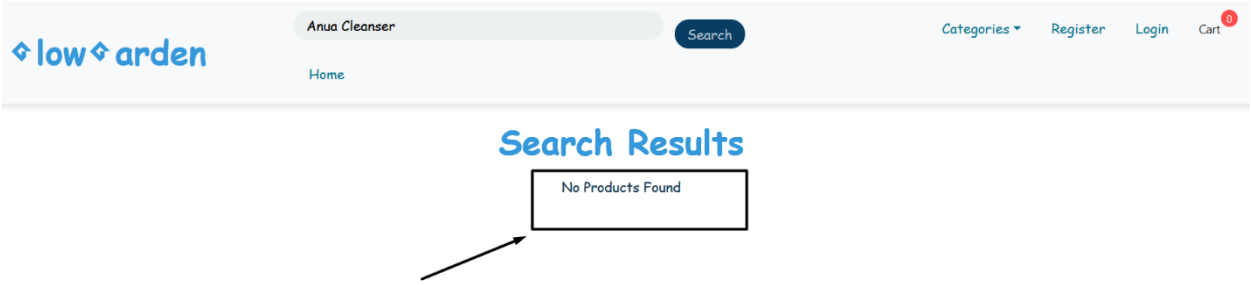


Fig 8.5: No Matched Product

3) Product Filter by Category:

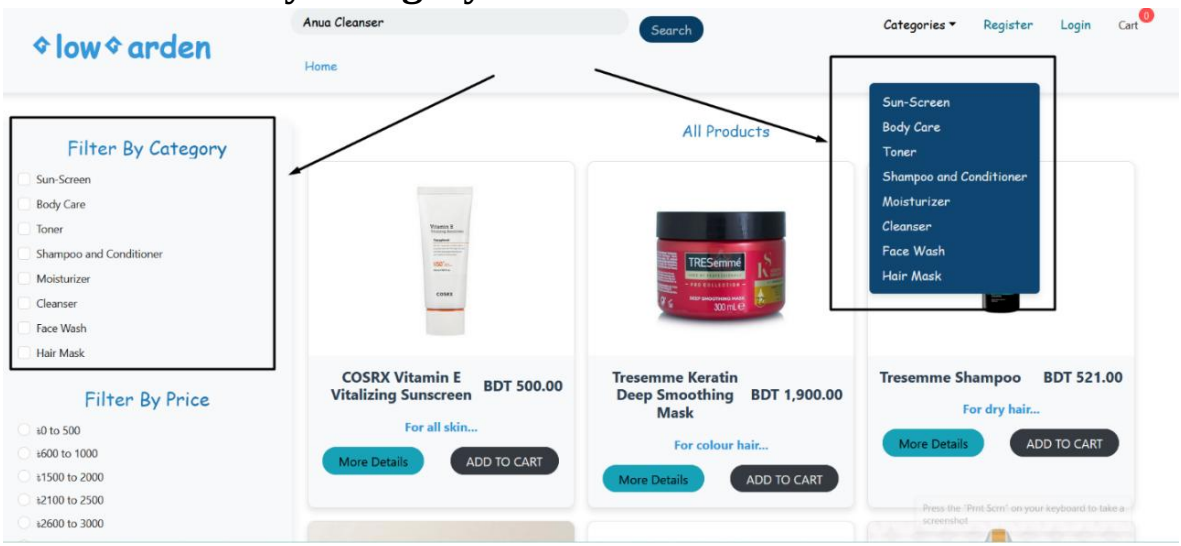


Fig 8.6: Filter Product By Category

4) Filter Products By price:

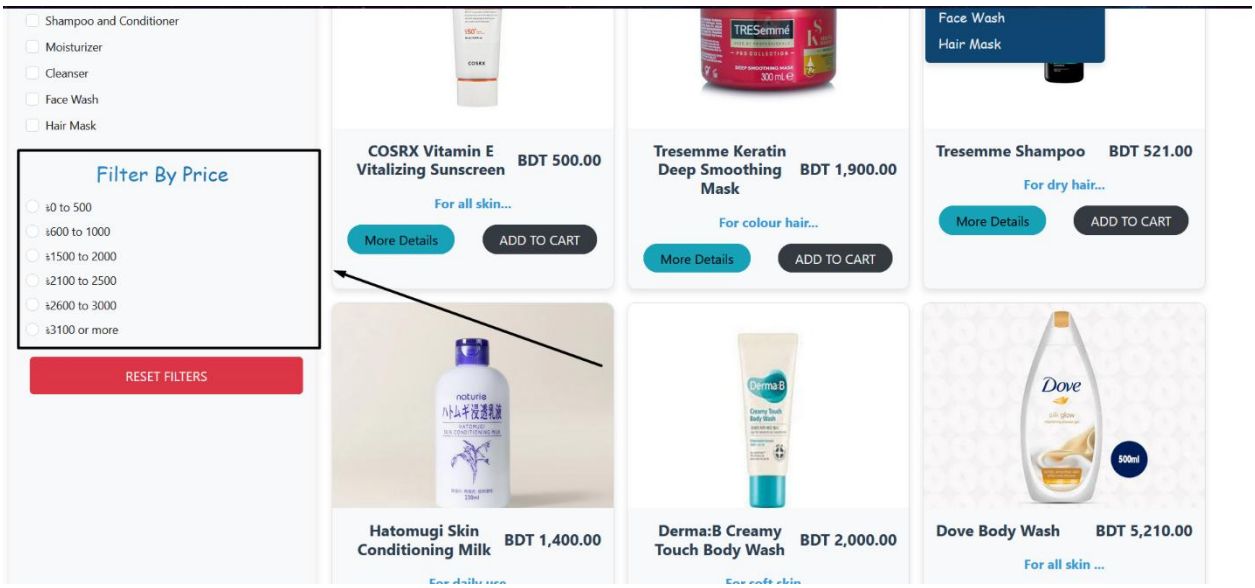


Fig 8.7: Filter Product By Price

5) Reset Filter:

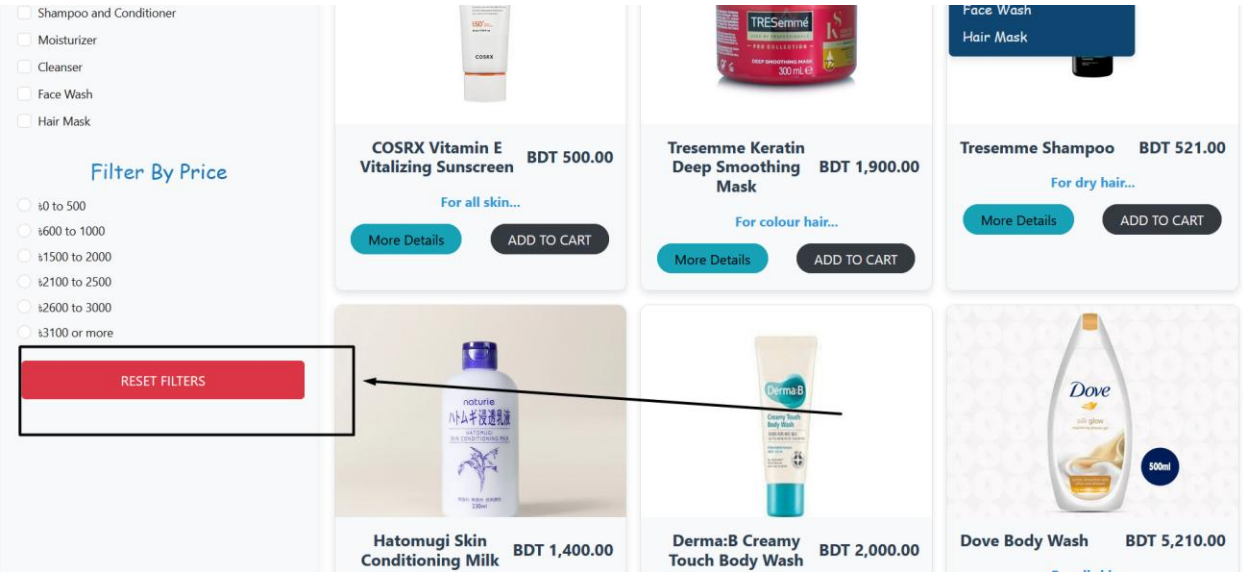


Fig 8.8: Reset Filter

6) Register and Login Option:

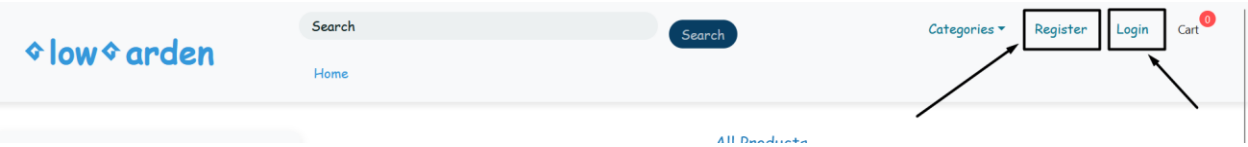


Fig 8.9: Registration and Login Option

7) Register Form:

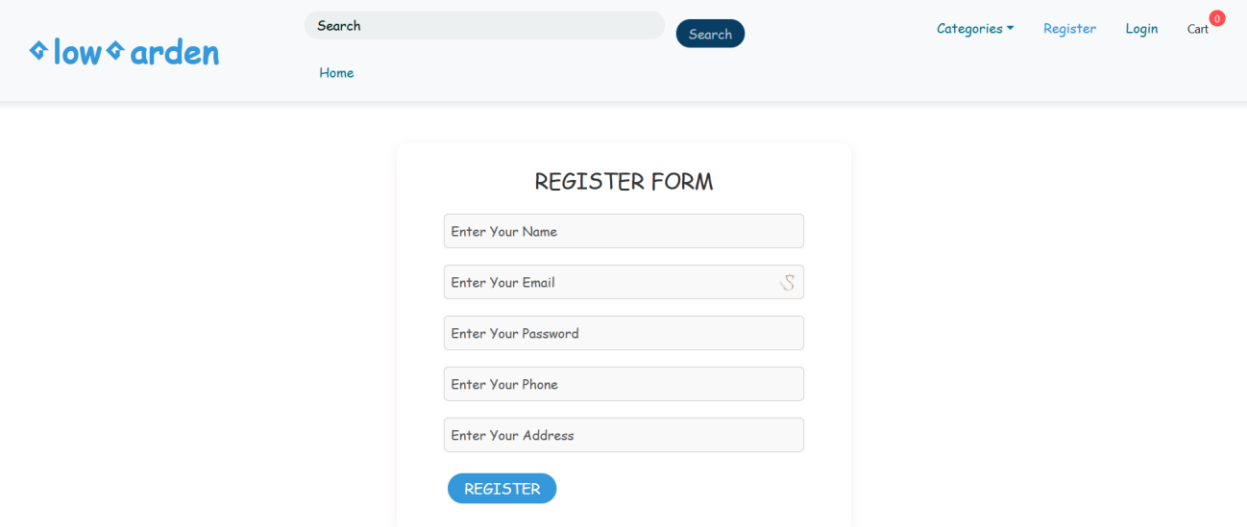


Fig 8.10 : Registration form

7) Create an account:

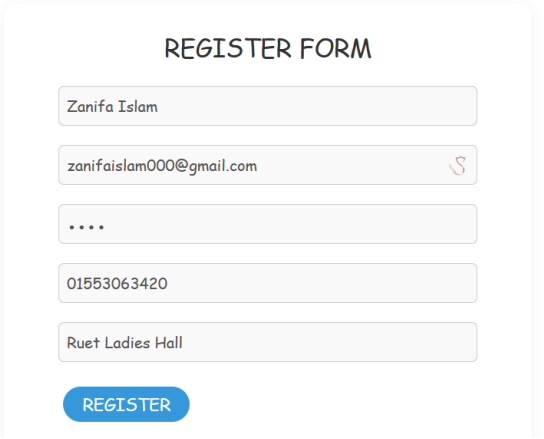


Fig 8.11: Create Account Page

8) Login Page:

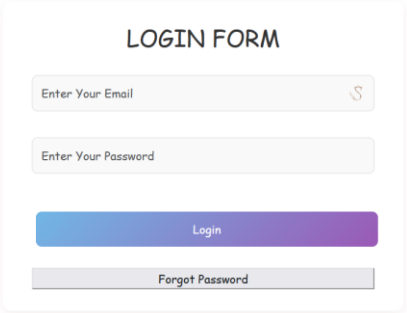


Fig 8.12: Login Page

9) Login as User:

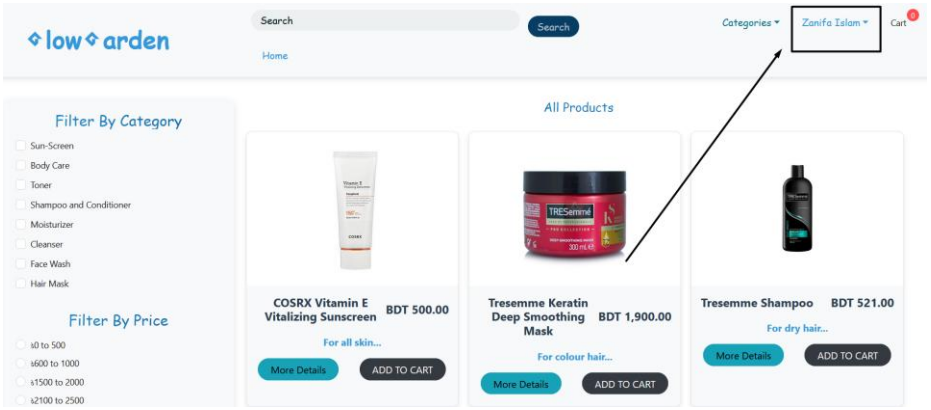


Fig 8.13: Login as User

10) Log Out:

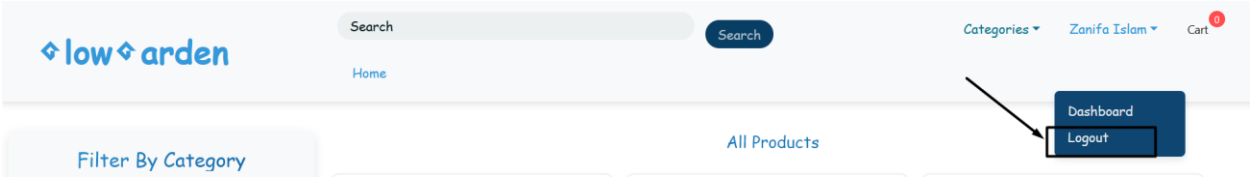


Fig 8.14: Log Out Option

11) Add to Cart:

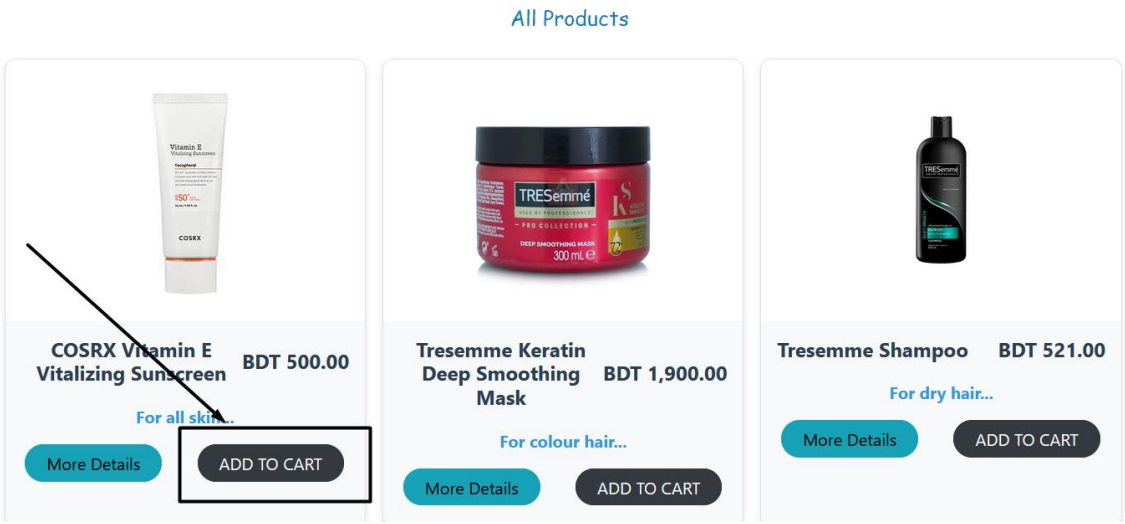


Fig 8.15: Add to Cart Option

12) Pop Up menu when adding anything into the cart:

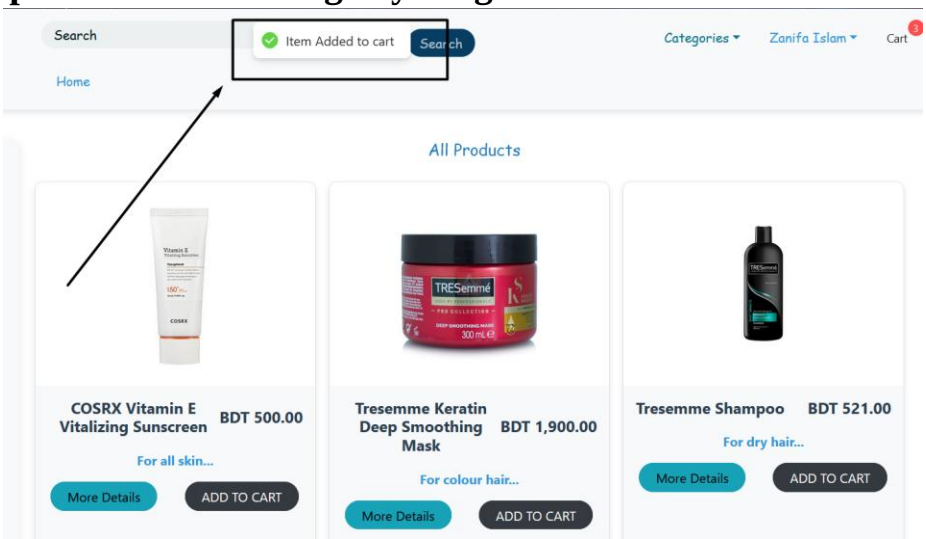


Fig 8.16: Pop Up Notification

13) Cart Page:

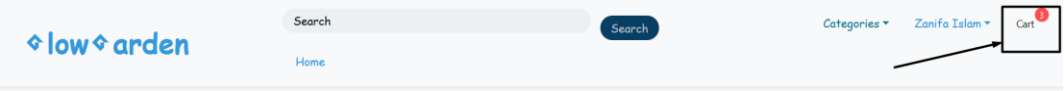


Fig 8.17: Cart option in home page

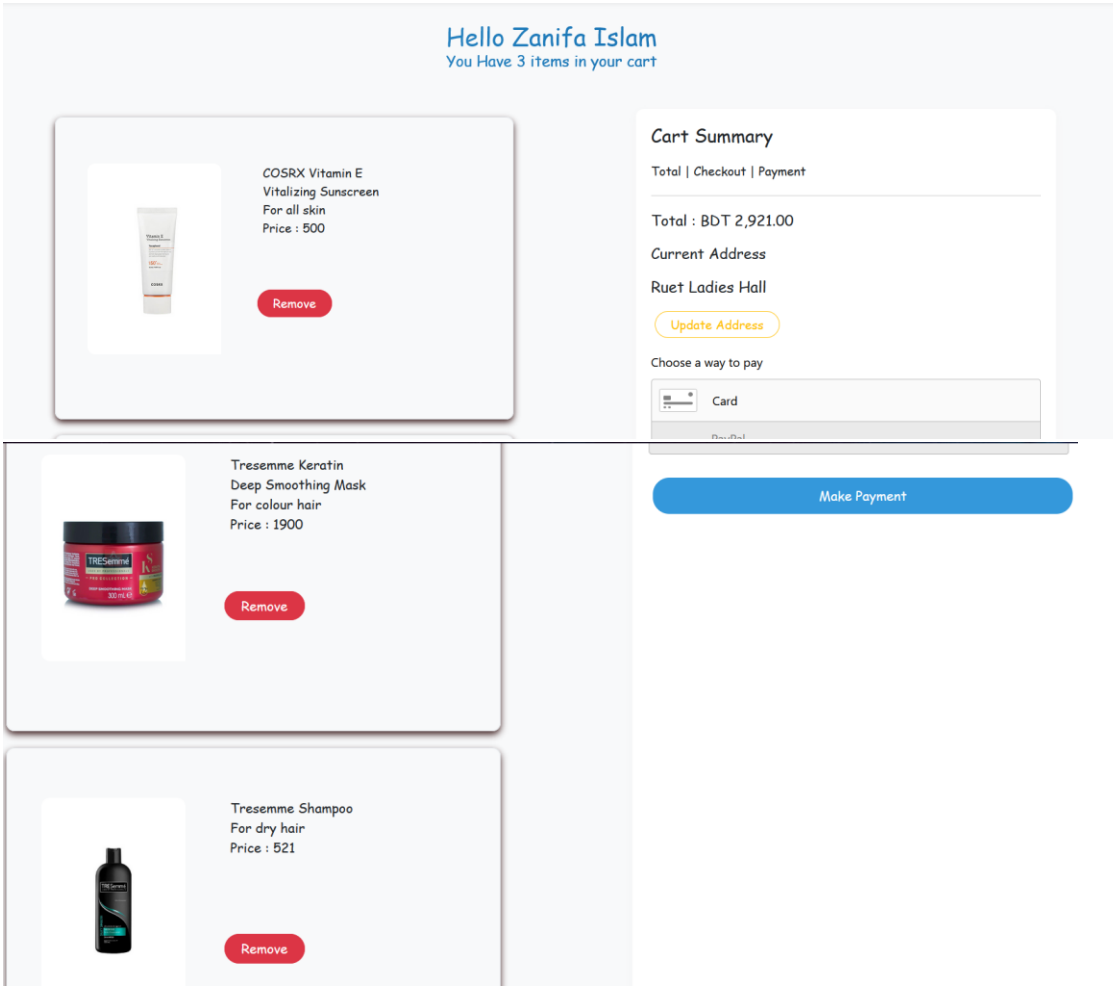


Fig 8.18 : Cart page

14) Order Page:

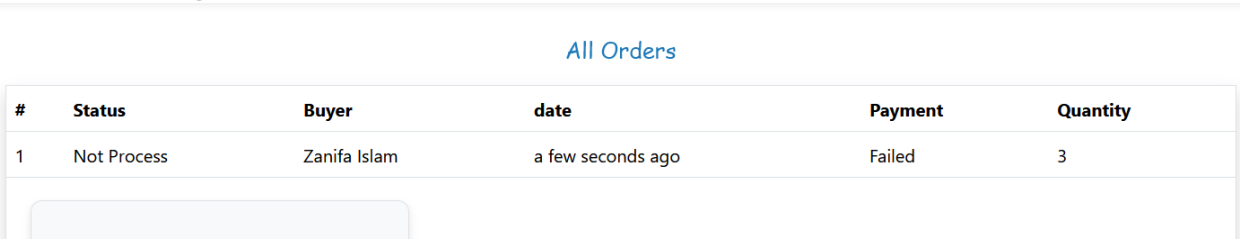


Fig 8.19: Order page

15) Payment Method:

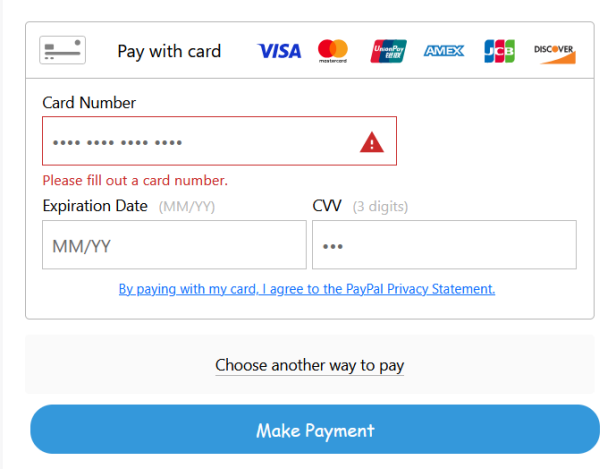


Fig 8.20: paypal payment method

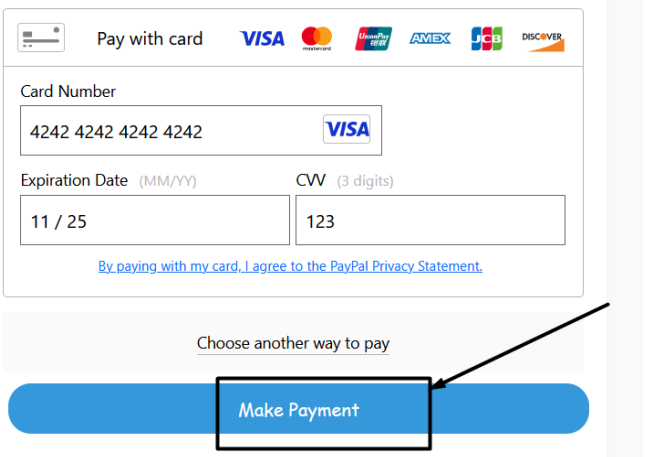


Fig 8.21:Make payment option

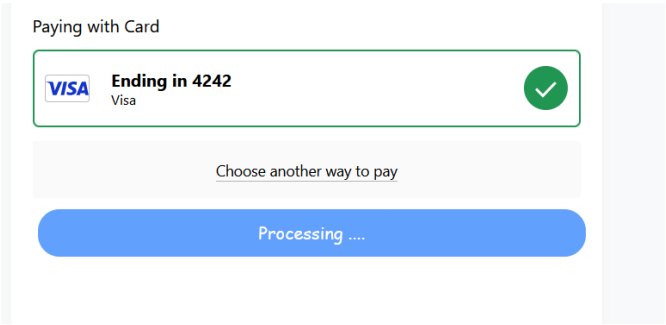


Fig 8.22: payment Processing

16) Wrong password Alarming message:

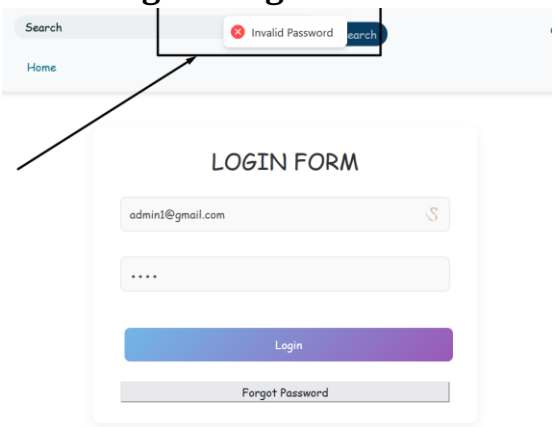


Fig 8.23: Wrong password Alarming notification

17) Login as Admin:

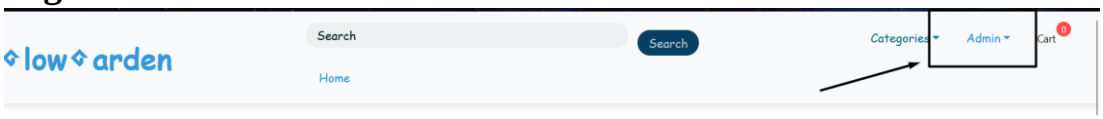


Fig 8.24: Login as admin

18) Admin Dashboard:

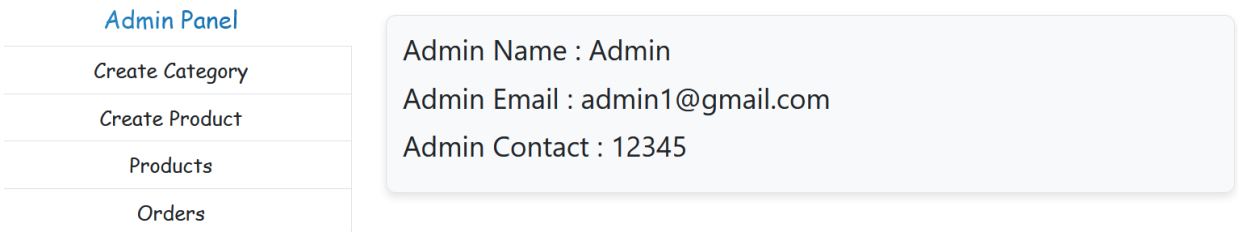


Fig 8.25: Admin Dashboard

8.3 Conclusion

The e-commerce platform for skin care products, developed for Software Development Project - II (ECE 3100, Utilizing the Big Bang model over an 8-month period (December 2024 to July 2025), the project leveraged the MERN stack (MongoDB, Express.js, React.js, Node.js) to deliver a functional system with secure authentication, product management, order processing, and integrations with Braintree and Nodemailer. Despite challenges like delayed documentation and limited advanced features, the solo effort resulted in a scalable, user-friendly platform, enhancing skills in full-stack development and preparing a foundation for future enhancements, such as analytics and performance optimization.

Chapter 9: References

i) M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002, pp. 45–67.

ii) Node.js Foundation, *Node.js Documentation*, [Online]. Available: https://nodejs.org/en/docs/

iii) React Team, *React – A JavaScript Library for Building User Interfaces*, [Online]. Available: https://reactjs.org/docs/getting-started.html

iv) MongoDB Inc., *MongoDB Manual*, [Online]. Available: https://docs.mongodb.com/manual/

v) Express.js Team, *Express – Node.js Web Application Framework*, [Online]. Available: https://expressjs.com/

vi) Braintree, *Braintree Developer Documentation*, [Online]. Available: https://developer.paypal.com/braintree/docs/start/

vii) GitHub Inc., *Git Documentation*, [Online]. Available: https://git-scm.com/doc

viii) MongoDB, Inc., "MERN Stack Explained," [Online]. Available: https://www.mongodb.com/resources/languages/mern-stack

ix) GeeksforGeeks, "Understand MERN Stack," [Online]. Available: https://www.geeksforgeeks.org/mern/understand-mern-stack/

x) freeCodeCamp.org, "MERN Stack Tutorial for Beginners with Deployment – 2025," YouTube, 2025. [Online]. Available: https://www.youtube.com/watch?v=F9gB5b4jgOI

xi) freeCodeCamp.org, "MERN Stack Tutorial for Beginners with Deployment – 2025," YouTube, 2025. [Online]. Available: https://www.youtube.com/watch?v=F9gB5b4jgOI

xii) MongoDB, Inc., "How To Use MERN Stack: A Complete Guide," [Online]. Available: https://www.mongodb.com/resources/languages/mern-stack-tutorial

xiii) MongoDB, Inc., "How To Use MERN Stack: A Complete Guide," [Online]. Available: https://www.mongodb.com/resources/languages/mern-stack-tutorial

Chapter 10: Appendices

Appendix A: Source Code / GitHub Link:

Github Link: *Ecommerce Website Using MERN*

Appendix B: Additional Screenshots

- **Forget Password:**

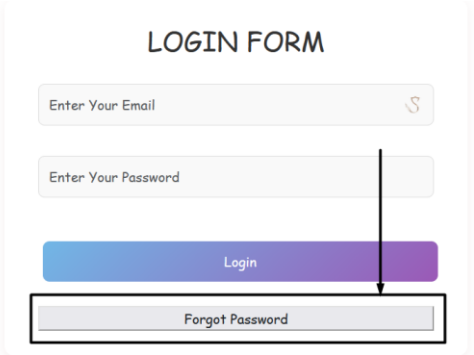


Fig 10.1: Forget password

- **User Dashboard:**

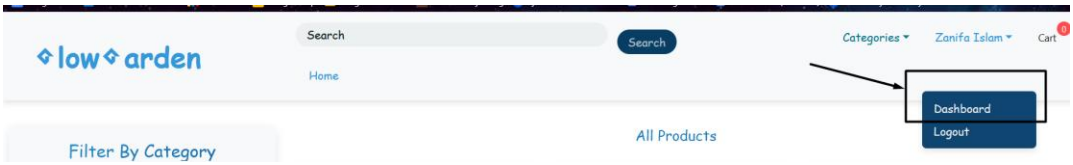


Fig 10.2: User Dashboard Option

- **User Details:**

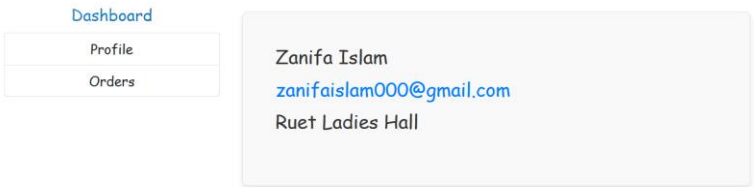


Fig 10.3 : User Dashboard

- **User Profile Page:**

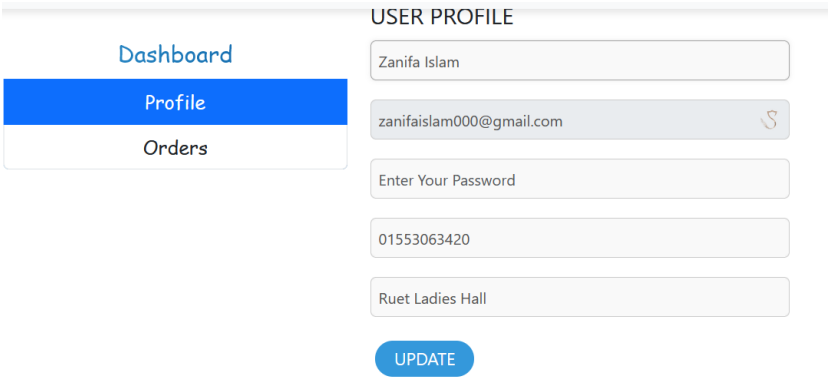


Fig 10.4 : User Profile

- **Create category of any product:**

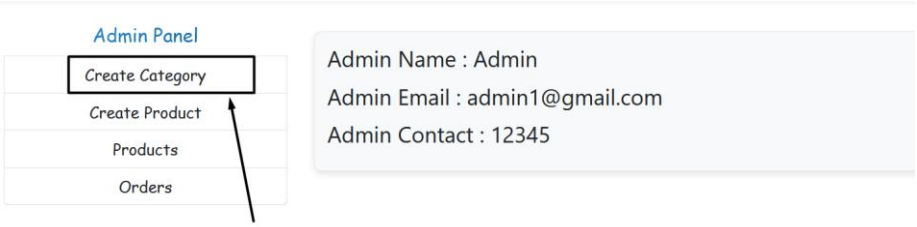


Fig 10.5 : Create category Option

- **Manage Category:**

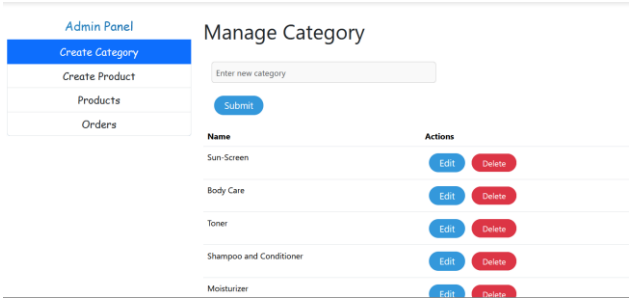


Fig 10.6 : Manage category page

- **Add New Category:**

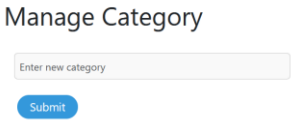


Fig 10.7 : Add New Category Option

- **Delete/ Edit a category:**

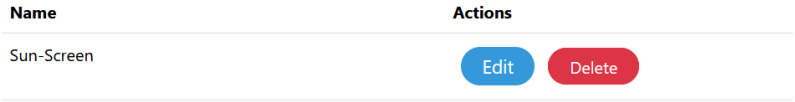


Fig 10.8: Delete Category Option

- **Deleted Category Pop up notification:**

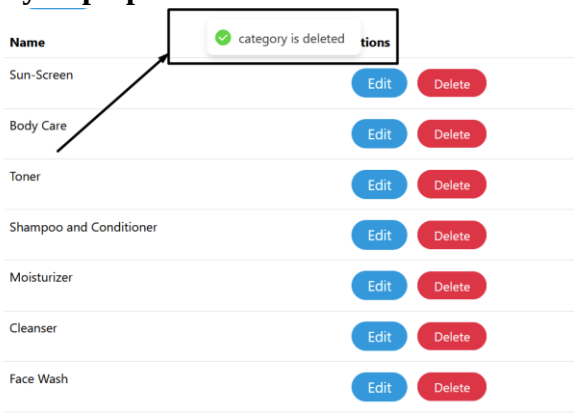


Fig 10.9 : Delete Category Option

- **Update Category Pop- Up notification:**

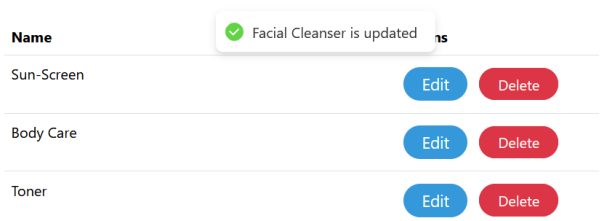


Fig 10.10 : Update Category Option

- **New Category Creating Pop-up notification:**

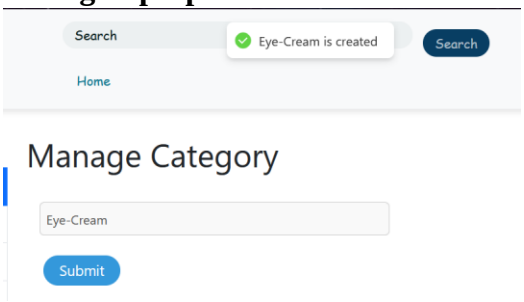


Fig 10.11 : pop up option for creating new category

- **Create Product Page:**

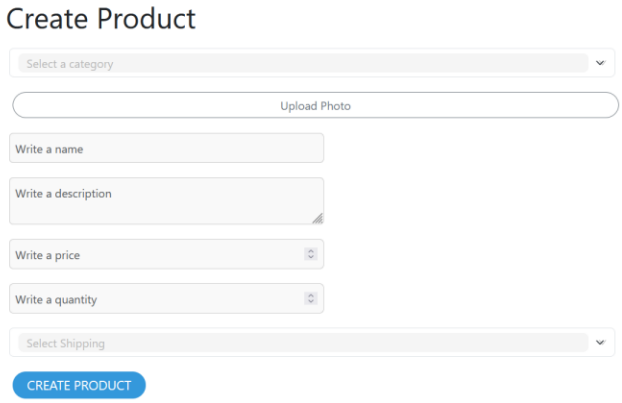


Fig 10.12 : Create product page

- **Update Product Page:**

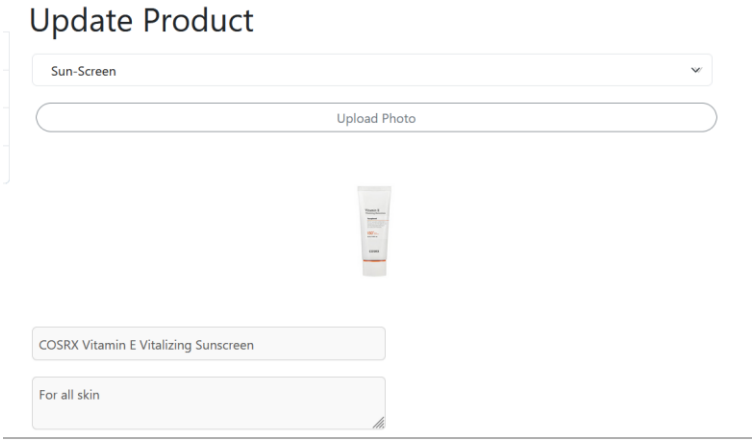


Fig 10.13 : Update product Option

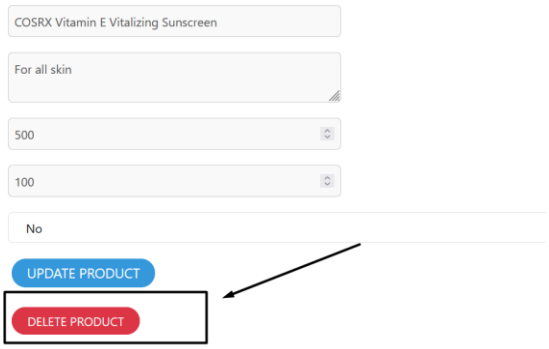


Fig 10.14 : Delete Product Option

- **Add Order page managed by Admin:**

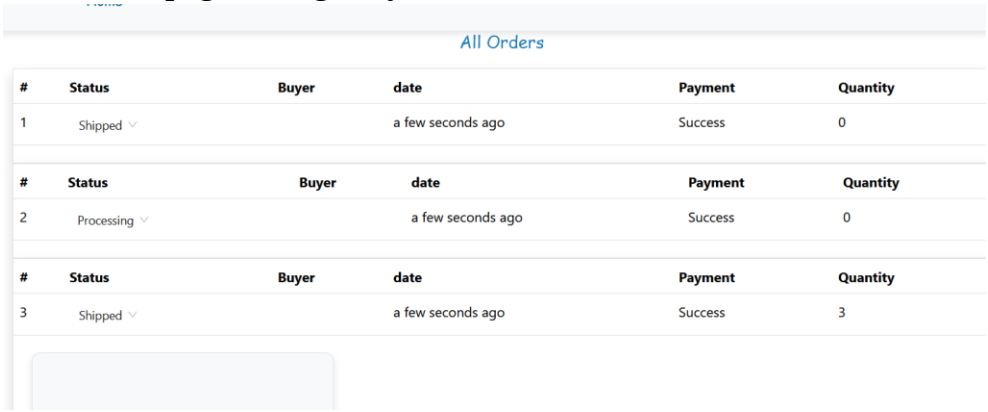


Fig 10.15 : Total order

- **Delivery Option:**

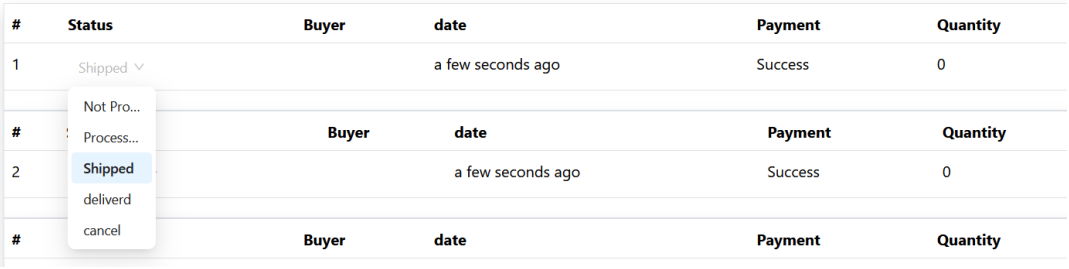


Fig 10.16 : Track order

- **Footer Section:**

Fig 10.17 : Footer Section

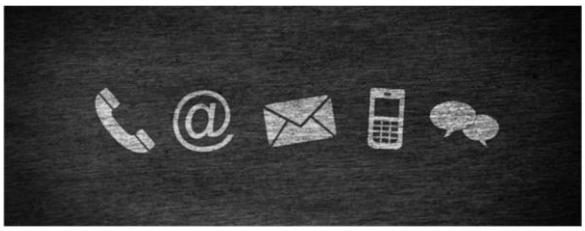
- **About Page**:

Our Philosophy

We are committed to using only skin-friendly, research-backed ingredients that either directly benefit the health of the skin or support the integrity of our formulations. We never take into account whether something is synthetic or natural, instead choosing ingredients based on biocompatibility. Skin Illustration That's why we focus on healthy pH levels, formulations the skin recognizes, small molecular structures that are easily absorbed, and effective active ingredients that also support and maintain the skin's acid mantle. But what we leave out of our products is just as important as what we put into them, so you will never find what we call the Suspicious 6™ (essential oils, drying alcohols, silicones, chemical sunscreens, fragrances/dyes, SLS) in our line.

Fig 10.18 : About Page

- **Contact Page:**



CONTACT US

any query and info about prodduct feel free to call anytime we 24X7 vaialible

: www.help@ecommerceapp.com

: 012-3456789

: 1800-0000-0000

Fig 10.19 : Contact Page

- **Privacy Policy Page:**

What is Personal Information and why do we collect it? Personal Information is information or an opinion that identifies an individual. Examples of Personal Information we collect includes names, addresses, email addresses, phone and facsimile numbers. This Personal Information is obtained in many ways including [interviews, correspondence, by telephone and facsimile, by email, via our website www.yourbusinessname.com.au, from your website, from media and publications, from other publicly available sources, from cookies- delete all that aren't applicable] and from third parties. We don't guarantee website links or policy of authorised third parties. We collect your Personal Information for the primary purpose of provid

Fig 10.20 : Privacy Policy page

Appendix C: User Manual

This user manual provides guidance for operating the e-commerce platform for skin care products, developed for Software Development Project - II (ECE 3100) this manual is designed for end-users (customers) and administrators, detailing navigation, features, and troubleshooting for the MERN stack-based system.

1. Introduction

The platform enables users to browse, purchase skin care products, and manage orders, while admins oversee product and category management. Access is via a web browser (e.g., Chrome, Firefox) at http://localhost:8080 (local testing) or a deployed URL (post-deployment).

2. Getting Started

- **System Requirements**: Modern browser, internet connection, and a device (PC/mobile).
- **Account Creation**:
    1. Visit the homepage.
    2. Click "Register" and enter email, password, name, and optional phone/address.
    3. Submit to receive a confirmation (local testing skips email verification).
- **Login**:
    1. Click "Login" on the homepage.
    2. Enter registered email and password.
    3. Click "Sign In" to access the dashboard.

3. User Features

- **Browsing Products**:
    - Navigate to "Shop" to view product listings.
    - Use filters (e.g., category) or search bar to find items.
    - Click a product for details (price, description, photo).
- **Adding to Cart**:
    - Select "Add to Cart" on a product page.
    - View cart via the cart icon (top right) to adjust quantities.
- **Placing an Order**:
    1. Proceed to checkout from the cart.
    2. Enter shipping details if not saved.
    3. Choose Braintree payment, complete transaction.
    4. Receive order confirmation (email or on-screen).
- **Order History**:
    - Go to "Profile" > "Orders" to view past orders and statuses (e.g., "Shipped").

4. Admin Features

- **Access**: Login with an admin account (role 1, set during registration).
- **Manage Categories**:
    1. Go to "Admin" > "Categories."
    2. Add, edit, or delete categories (e.g., "Moisturizers").
- **Manage Products**:
    1. Navigate to "Admin" > "Products."
    2. Add products with name, price, photo, and category; edit or delete existing ones.

- **Manage Orders**:
    1. Go to "Admin" > "Orders."
    2. Update statuses (e.g., "Not Process" to "Shipped").

5. Troubleshooting

- **Login Issues**: Ensure correct credentials; reset password via "Forgot Password" (email link in production).
- **Payment Errors**: Verify Braintree token; contact admin if sandbox fails.
- **Page Load Delays**: Clear browser cache or check internet connection.
- **Contact Support**: Use the provided email (e.g., projectemail@ruet.edu.bd) for assistance.

6. Safety and Security

- Use strong passwords and avoid sharing login details.
- Transactions are secure via Braintree's encryption.
- Report suspicious activity to the admin immediately.

7. Updates

This manual reflects the system as of 19 September 2025. Check for updates post-deployment or contact the developer for the latest version.

This manual ensures users and admins can effectively utilize the platform, supporting its academic and practical objectives.