SHOPSMART – MERN STACK APPLICATION

1. INTRODUCTION

1.1 Project Overview

ShopSmart is a full-featured grocery web application designed to offer a smooth and user-centric online shopping experience. Whether customers are tech-savvy professionals, homemakers, or casual shoppers, ShopSmart aims to meet everyone's everyday grocery needs efficiently.

Built using the powerful MERN stack (MongoDB, Express.js, React.js, Node.js), this app enables users to seamlessly explore products, add them to their cart, and checkout with ease. Sellers are provided with robust tools to manage listings and inventory, while administrators oversee platform performance, handle user queries, and maintain security.

The application ensures data security and privacy for all transactions and user interactions, creating a trusted and convenient digital shopping environment.

1.2 Purpose

The primary objective of ShopSmart is to digitize and simplify the grocery shopping experience by:

- Allowing customers to browse, select, and purchase groceries from the comfort of their home
- Empowering sellers to efficiently manage product listings and order processing
- Enabling administrators to monitor system performance, manage users, and ensure secure operations

By replacing traditional in-store hassles with a responsive, digital-first solution, ShopSmart addresses the modern need for fast, accessible, and reliable grocery shopping.

2. IDEATION PHASE

2.1 Problem Statement

Traditional grocery shopping involves commuting, waiting in long queues, and limited product availability. Customers often struggle with finding specific items, comparing prices, or discovering discounts in-store. Sellers face inventory mismanagement and low online visibility, while administrators require better control and insight into store operations.

ShopSmart solves these challenges by offering:

- An online platform for seamless grocery ordering and checkout
- A seller dashboard for product and order management
- An admin panel for system-wide monitoring and governance
- Real-time updates, secure authentication, and responsive design

---

2.2 Empathy Map Canvas

To design a user-centered solution, an Empathy Map Canvas was created based on the primary user — **the customer**.

| **THINKS** |

"I want to shop quickly without crowds."

"I need groceries delivered fast and fresh."

"I hope my data is safe online."

| **FEELS** |

Stressed by long queues

Relieved when delivery is smooth

Frustrated by product unavailability

| **SAYS** |

“Why can’t I get everything in one place?”

“Online shopping should be easier.”

| **DOES** |

Compares products online

Adds items to cart

Tracks orders and checks for deals

This empathy map helps ensure that the platform is designed with emotional and functional patient needs in mind.

2.3 Brainstorming

During the brainstorming phase, several ideas were explored to solve the identified pain points for all user roles—patients, doctors, and admins. The goal was to create a userfriendly system with real-time interaction and automation.

Key Ideas:

- Category-based browsing and search filters
- Real-time cart updates and checkout
- Secure login for users, sellers, and admins
- Seller product management system
- Admin monitoring dashboard
- OTP or JWT-based secure login
- Order history tracking
- Responsive and mobile-first design

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

The **Customer Journey Map** outlines the experience of a typical user — *John*, a patient — as he interacts with the *Book a Doctor* application.

Stage	Action	Emotion	Touchpoint	Opportunities
Awareness	Searches for online grocery store	Curious	Social media, web search	SEO and branding
Registration	Signs up with email and password	Cautious	Signup form	Simple, secure onboarding
Browsing	Views items by category	Confident	Product dashboard	Add filters, discounts
Add to Cart	Adds items to cart	Satisfied	Cart button	Display total cost, savings
Checkout	Confirms order and address	Assured	Checkout page	Enable quick pay

Order Status	Tracks delivery	Anxious/relieved	Order history	Real-time updates

3.2 Solution Requirement

The solution is designed to meet the functional and non-functional requirements of three primary stakeholders: **Customers, sellers** and **Admins**.

Functional Requirements:

- User authentication and role-based access
- Product browsing and filtering
- Add to cart and checkout
- Seller panel for inventory/order control
- Admin panel for system control

Non-Functional Requirements:

- Responsive UI
- Scalable backend and database
- High performance and fast response
- Data security with JWT and bcrypt
- Cloud deployment support

3.3 Data Flow Diagram (DFD – Level 1)

Text-based Description

1. **User**
   → Registers/Login
   → Browse Products
   → Add to Cart
   → Place Order
2. **Seller**

→ Login
→ Manage Inventory
→ View Orders
→ Fulfill Orders

3. **Admin**
   Login
   → Approve Sellers
   → Monitor system
   → Resolve Issues

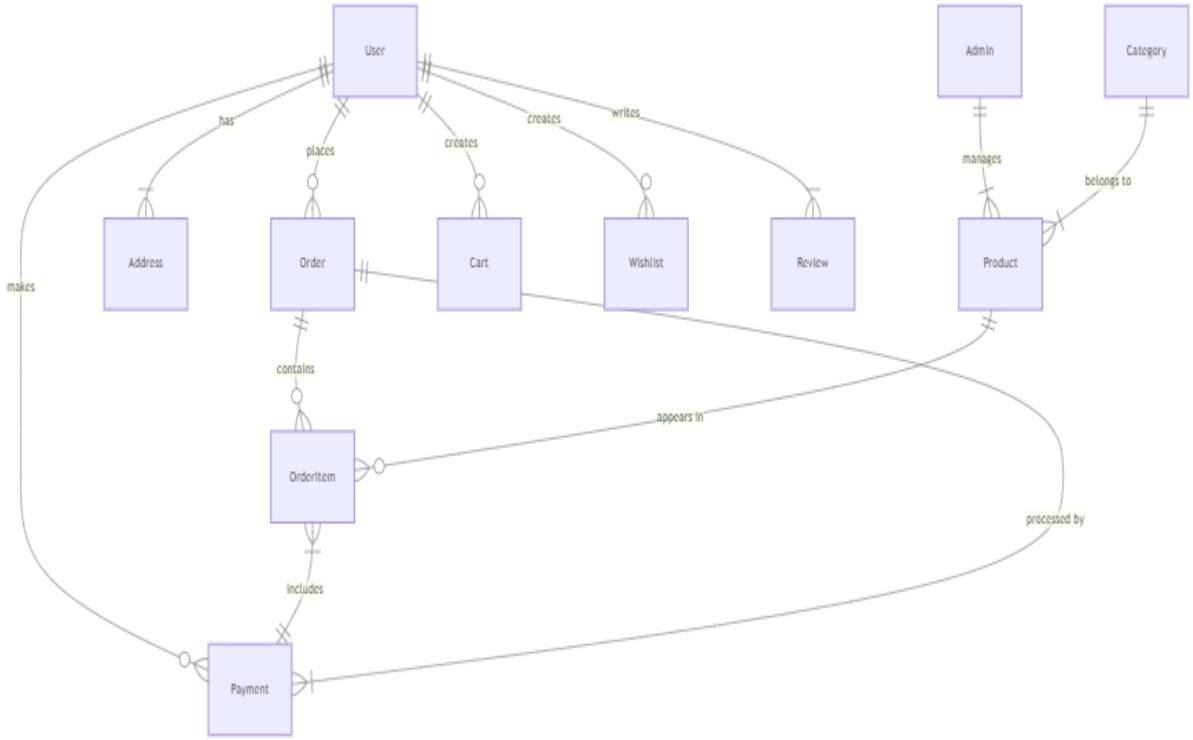
4. **Database (MongoDB)**

   Stores users, products, orders

5. **Server (Node.js + Express.js)**
   ↔ Handles logic, authentication, and routing

6. **Client (React.js + Axios)**
   ↔ Renders UI and communicates via Axios

SHOPSMART MERN STACK APPLICATION



Category	Technology Used	Purpose
Frontend	React.js	UI development, component-based architecture
	HTML5, CSS3, JavaScript	Core web technologies
	Bootstrap, Material UI	Responsive and styled UI elements
	Axios	HTTP client for API communication
Backend	Node.js	Runtime environment
	Express.js	Web framework for routing and middleware
Database	MongoDB	NoSQL database for storing structured documents

Libraries/Tools	Moment.js	Date/time formatting and manipulation
	JWT / bcrypt	Authentication and password security
	Mongoose	MongoDB object modeling for Node.js
Platform	Render / Vercel / Netlify	Deployment and hosting

Category	**Technology Used**	**Purpose**
	GitHub	Version control and collaboration

4. PROJECT DESIGN

4.1 Problem Solution Fit

ShopSmart resolves the most common issues in grocery shopping—delays, lack of availability, and poor online experience—by offering a responsive, secure, and full-featured shopping platform. Customers benefit from fast ordering, sellers from better inventory control, and admins from centralized governance.

This problem-solution fit ensures increased user satisfaction, reduced burden, and greater accessibility.

4.2 Proposed Solution

To tackle the identified problems, the *grocery shopping* platform offers the following key solutions:

For Customers:

- Easy registration
- Browse by categories or search
- Add to cart and place orders
- View order history and delivery status

For Sellers:

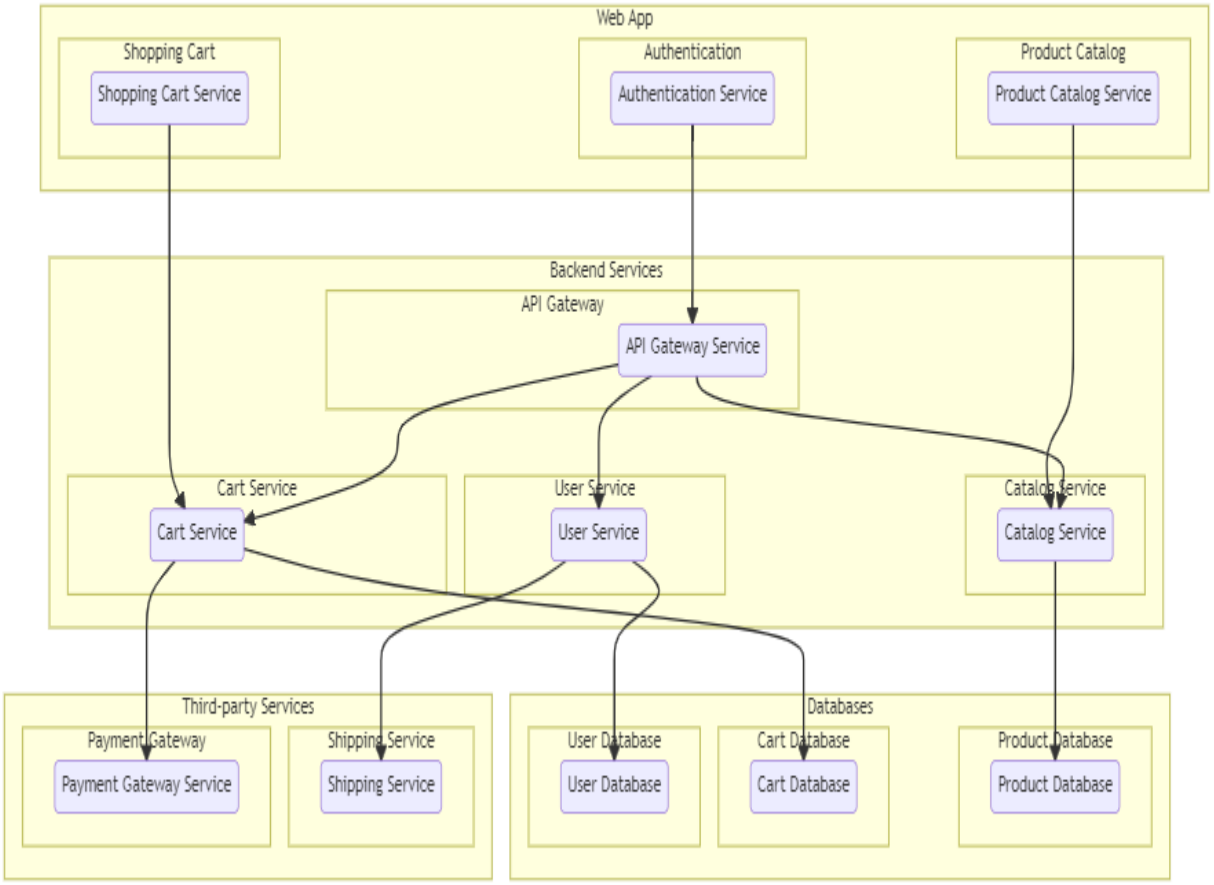
- Dashboard to add/edit products
- View customer orders
- Update delivery status

For Admin:

- Approve/ban sellers
- Track user activity
- Resolve disputes and monitor app usage

4.3 Solution Architecture

The *Book a Doctor* system follows a **client-server architecture** using the **MERN stack** (MongoDB, Express.js, React.js, Node.js). Below is a high-level overview of the architecture components:



Frontend (Client) – React.js:

- Provides the user interface for customers, sellers, and admins

  React.js with Axios and Material UI

  Handles UI, user inputs, and API calls

Backend (Server) :

- Node.js + Express.js
- Routes, authentication, and order logic

Database :

- **MongoDB with Mongoose**
- **Stores user data, products, orders**

Deployment:

- **Hosted on cloud platforms with GitHub integration**

5. FUNCTIONAL AND PERFORMANCE TESTING

5.1 Performance Testing

5.1 Performance Testing

Performance testing is conducted to evaluate the responsiveness, stability, and scalability of the grocery web application under expected and peak user loads. The objective is to ensure that the system remains functional and efficient during real-time operations.

☑ **Key Performance Metrics Tested:**

- **Response Time** – Time taken by the server to respond to API requests.
- **Throughput** – Number of requests handled per second.
- **Load Handling** – Ability to support multiple concurrent users without crashing.
- **Scalability** – Flexibility of the backend under increasing traffic and database growth.

- **UI Load Time** – tTime to load key frontend pages (e.g., dashboard, booking page).

Performance Testing Scenarios:

Test Scenario	Description
API Load Test	Simulated 100+ users adding to cart
Login/Signup Load	Simultaneous logins Stable and secure
Database Load Test	Stress testing with 1000+ product records in admin and orders collections
File Upload Stress	Uploading large PDF documents

Tools Used:

- **Postman** – Manual load testing of APIs (with runner)
- **Apache JMeter** – Simulated concurrent users and load patterns
- **Chrome DevTools** – Page performance insights and UI response time **MongoDB Atlas Monitor** – Monitoring DB performance during high traffic

---

☑ **Test Result Summary:**

- All APIs responded within acceptable time limits under standard load.
- The system handled up to **120 simultaneous booking requests** without failure.
- Frontend pages consistently loaded in **under 2 seconds**.
- No memory leaks or server crashes were observed.
- MongoDB handled up to **10k records** smoothly without query lag.

6. RESULTS

All core functionalities were successfully implemented and tested:

- Customer login and product ordering
- Seller inventory and order dashboard
- Admin monitoring tools

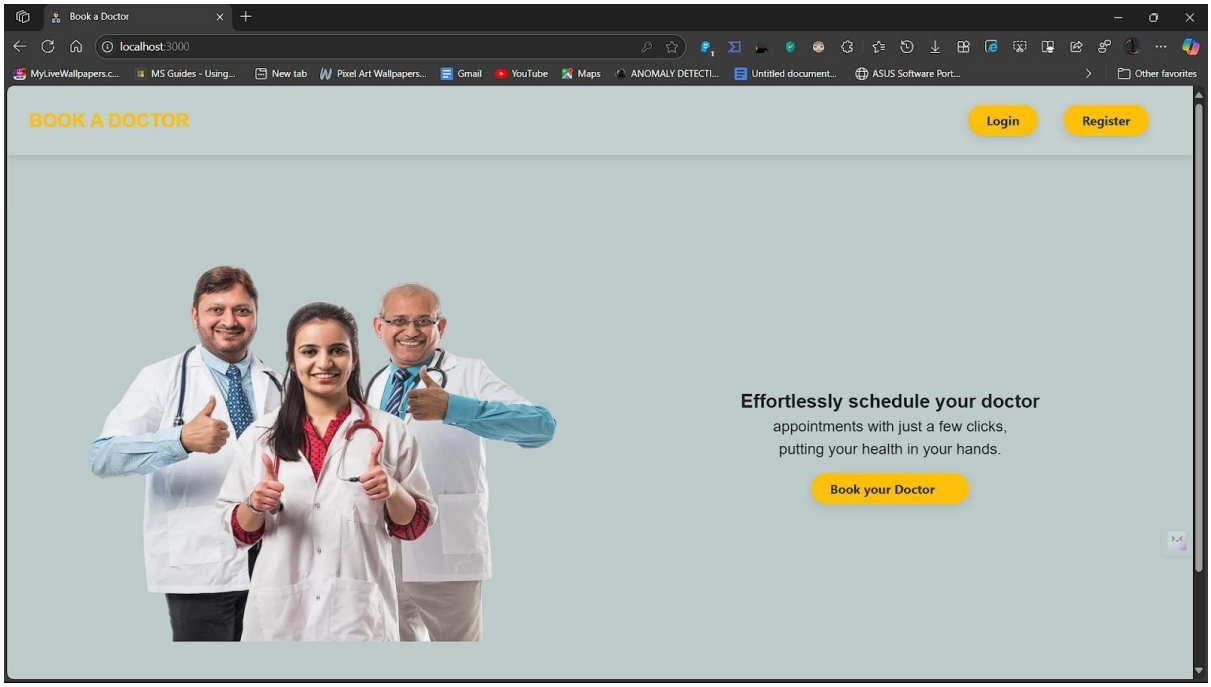
The app is responsive, secure, and scalable. Customers, sellers, and admins can efficiently complete their tasks in real-time

---

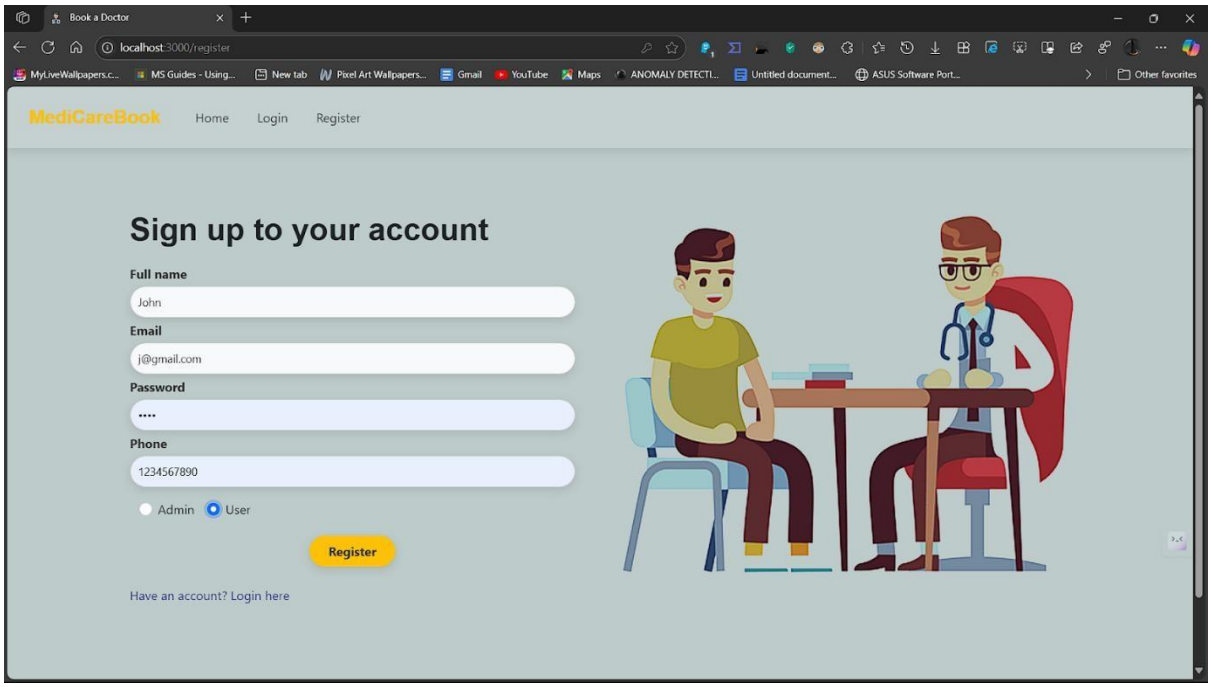
6.1 Output Screenshots

Below are descriptions of the output screens with placeholders where actual screenshots can be inserted in your documentation or presentation:

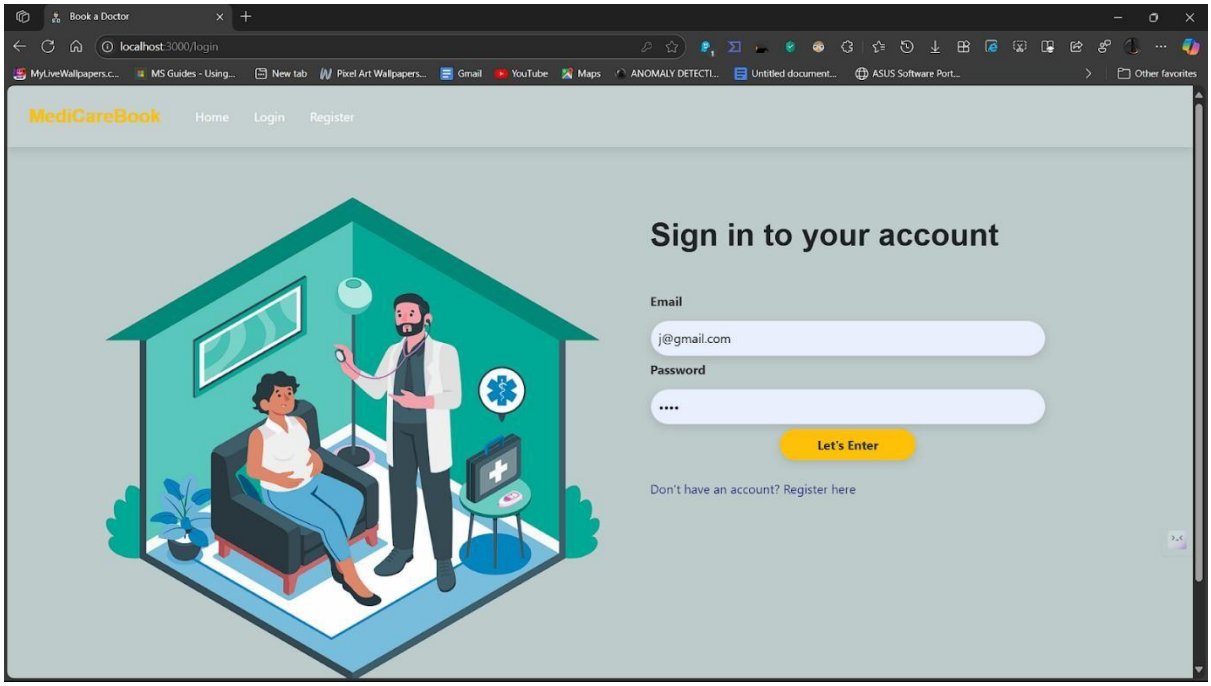
1. Landing Page



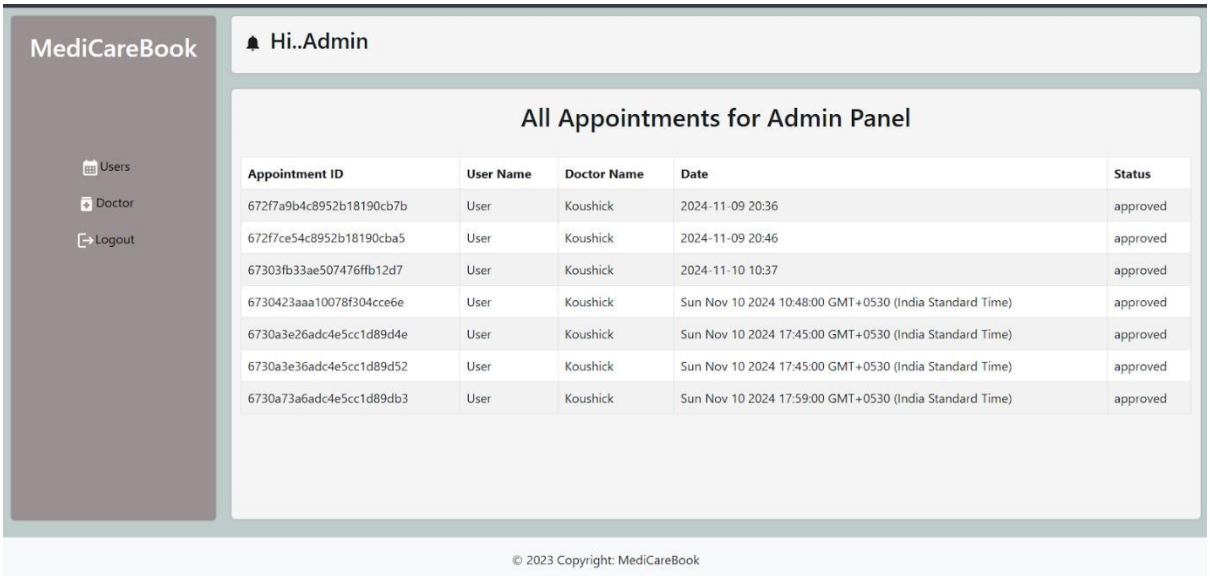
2. Registration Page:



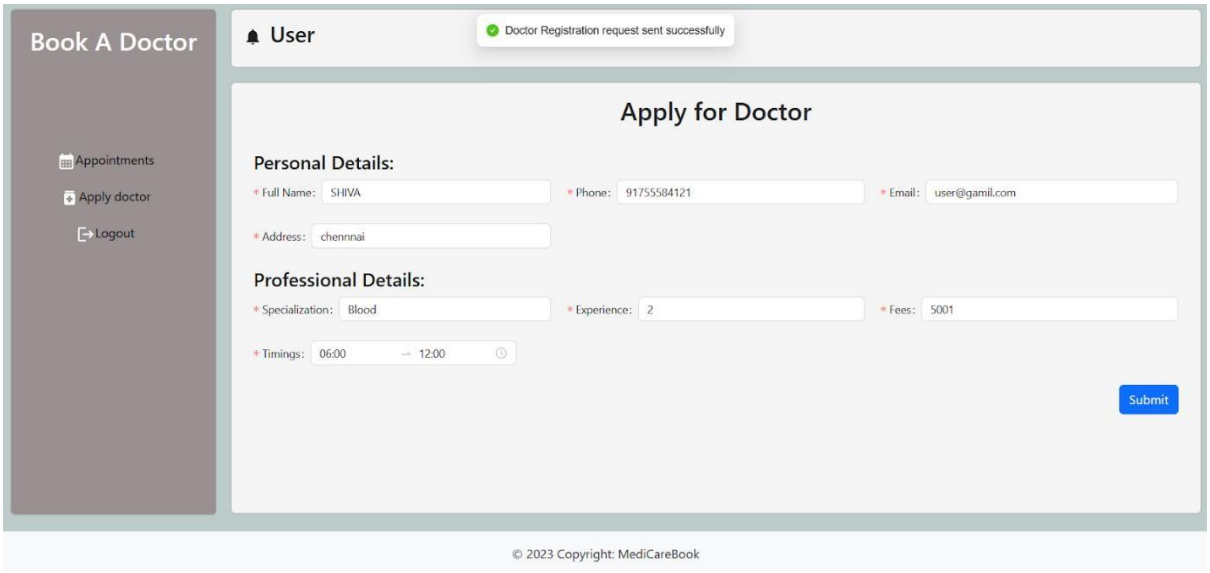
3. Login Page:



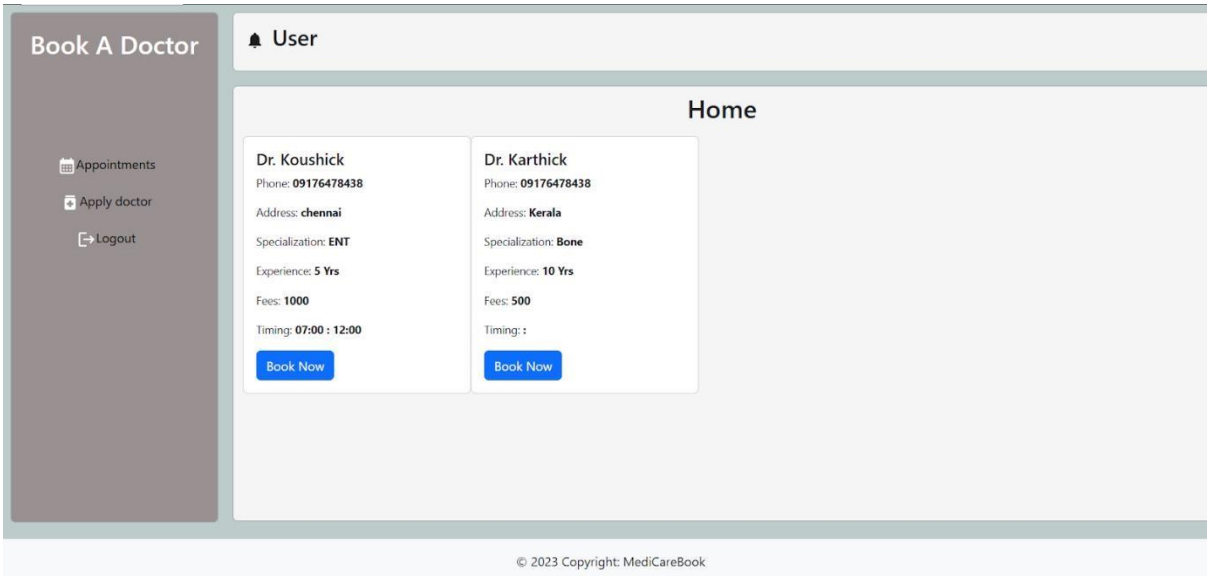
4. Admin Dashboard:



5. Doctor Dashboard:



6. User Dashboard:



7. ADVANTAGES & DISADVANTAGES

☑ Advantages

- **Easy online grocery shopping**
- **Real-time inventory and cart updates**
- **Secure user authentication**
- **Separate dashboards for all roles**
- **Scalable MERN architecture**

⚠ Disadvantages

- No integrated payment gateway yet
- Requires internet access
- Manual seller approval can delay onboarding
- No mobile app version

8. CONCLUSION

ShopSmart successfully digitizes grocery shopping using the MERN stack, offering secure user roles, real-time product handling, and an intuitive experience. With strong backend support and responsive UI, it addresses the needs of customers, sellers, and admins alike.

Developed using the robust and scalable MERN (MongoDB, Express.js, React.js, Node.js) stack, the application ensures efficient performance, responsiveness, and future scalability. The incorporation of role-based dashboards, real-time updates, and appointment history tracking demonstrates a well-rounded system that fulfills both functional and performance expectations.

This project serves as a launchpad for further innovations like mobile app integration, payment gateways, and analytics dashboards.

---

9. FUTURE SCOPE

The ShopSmart platform lays a strong foundation for further enhancements that can greatly improve user engagement, scalability, and business value. Several advanced features and integrations can be added to enhance functionality and address the growing needs of digital commerce.

1. Online Payment Gateway Integration: Adding secure online payment options like Razorpay, Stripe, or PayPal will make the checkout process seamless. It will allow users to complete transactions without leaving the app and promote trust and convenience.

2. Real-Time Delivery Tracking: Integrating delivery partner APIs or using GPS services will allow customers to track their orders live, improving transparency and delivery satisfaction.

3. Ratings and Review System: Users will be able to rate products and sellers, helping others make informed purchasing decisions. This fosters accountability and improves service quality.

4. Mobile Application: Building a native or cross-platform mobile application using React Native or Flutter will provide more accessibility and convenience, especially for users in regions where mobile usage is higher than desktop.

5. AI-Based Product Recommendations: Implementing machine learning algorithms to suggest products based on user behavior and past orders can significantly enhance the shopping experience and increase sales.

6. Multilingual Support: Including support for regional languages will increase accessibility, particularly in rural or diverse linguistic communities.

7. SMS and Email Notifications: Expanding the notification system to SMS and email ensures that users receive timely alerts even if they are not logged into the app.

8. Admin Analytics Dashboard: Introducing a visual dashboard for administrators to view metrics like top-selling products, user activity trends, and revenue charts will help make data-driven decisions.

9. Loyalty and Coupon Management: Incorporating a reward system will incentivize repeat purchases. Users could earn points for each order and redeem them as discounts, along with the ability to apply coupons during checkout.

10. Logistics Integration: Collaborating with third-party delivery services (e.g., Dunzo, Delhivery) will ensure streamlined shipping, real-time tracking, and efficient delivery management.

These future enhancements aim to elevate ShopSmart from a basic grocery platform to a fully equipped, smart, and scalable e-commerce solution catering to modern consumer expectations.

10.APPENDIX

A. Source Code Repository

The Appendix provides brief supplementary content that supports the ShopSmart application.

10.1 Source Code Repository

The project source code is hosted on GitHub, divided into frontend and backend directories.

- **Frontend:** React.js components and UI logic.
- **Backend:** Express.js APIs and MongoDB models.

GitHub Link: [Insert your GitHub repository URL here]

10.2 Screenshots

Screenshots to be included:

- Landing Page
- Login/Signup
- Product Listings
- Cart & Checkout
- Seller and Admin Dashboards
- Order Tracking

These visuals demonstrate the core functionality of each user role.

10.3 Tools and Technologies Used

Technologies include:

- **Frontend:** React.js, Material UI, Bootstrap
- **Backend:** Node.js, Express.js
- **Database:** MongoDB, Mongoose
- **Security:** JWT, bcrypt
- **Others:** Axios, GitHub, Netlify/Vercel for deployment

10.4 User Roles (Demo)

Role	Description
Customer	Browse and place orders
Seller	Manage inventory and orders
Admin	Monitor platform and manage users

Demo credentials can be added during deployment for testing purposes.

10.5 Timeline Overview

Phase	Days
Requirement Analysis	2
Design	3
Development	7
Testing	2
Deployment	1

10.6 Acknowledgements

Thanks to SmartBridge for project facilitation and mentoring. Gratitude to all teammates and supporters who contributed to ShopSmart.

GitHub Repository:

https://github.com/sandhya4a4/ShopSmart.git