BookNest – An Online Bookstore Platform using MERN

INTRODUCTION:

The BookNest is a full-stack online bookstore application developed using the MERN stack – MongoDB, Express.js, React.js, and Node.js. It provides a user-friendly platform for book lovers to browse, search, and buy books across various categories. The application ensures a smooth and responsive experience with features like book previews, wishlist, shopping cart, and secure order placement.

BookNest supports three types of users: **Customers**, **Sellers**, and **Admins**. Customers can register, log in, explore the available books, add them to wishlist or cart, and place orders. Sellers can manage their book inventory, view order analytics, and track sales. Admins oversee the entire system, including managing users, sellers, books, and system-wide reports.

The project is designed to replicate a real-world e-commerce bookstore experience. With role-based access, interactive dashboards, and real-time order tracking, BookNest combines powerful backend functionality with a modern, engaging frontend. It is ideal for showcasing skills in full-stack web development, database management, user authentication, and real-time data handling.

Key Features :

- **User-Friendly Interface**
  - The platform offers an intuitive and responsive design, making it easy for users to browse, search, and purchase books with minimal effort.
- **Secure Authentication System**
  - BookNest provides secure login and registration functionality for users, sellers, and admins using encrypted passwords and token-based authentication.
- **Role-Based Access Control**
  - The application supports three different roles – User, Seller, and Admin – each with access to specific features based on their role.
- **Book Browsing and Wishlist**
  - Users can explore a wide variety of books, view detailed information, and add their favorite titles to a wishlist for future reference.
- **Shopping Cart and Order Placement**

        - Users can add books to their cart, update quantities, and place orders. Total amount calculation is handled dynamically before checkout
    - **Order History Tracking**
        - Every user can view their past orders, including book details, order date, total amount, and status updates.
    - **Seller Dashboard**
        - Sellers can manage their books, view orders placed for their items, update inventory, and track analytics like revenue and top-selling books.
    - **Admin Panel**
        - Admins have complete control over the platform. They can manage users, sellers, books, and view platform-wide analytics such as revenue, order count, and user distribution.
    - **Book Search Functionality**
        - Users can easily search for books by title or author, helping them quickly find the content they're looking for.
    - **Order Status Management**
        - Sellers and admins can update the order status (e.g., Processing, Shipped, Delivered), and users can track these updates in real time.
    - **Analytics and Reporting**
        - Both sellers and admins have access to detailed analytics including total orders, total revenue, and most popular books.

DESCRIPTION:

The BookNest is a full-stack web application designed to simulate a real-world online bookstore platform. The project provides a digital space where users can browse, search, and purchase books with ease. It features a smooth and responsive interface built using React, allowing for seamless user interaction across all devices. Whether it's a student looking for academic books or a reader exploring fiction, BookNest caters to all.

The platform supports three types of users – general users (customers), sellers (organizers), and admins. Each user role has its own dashboard and access controls. Customers can view books, add them to a cart or wishlist, and place orders. Sellers can list books, manage their inventory, and fulfill orders. Admins oversee the entire system by managing users, sellers, books, and analytics. Role-based access ensures security and clarity in navigation.

The BookNest includes a secure sign-up and login system for all roles. Once logged in, users get access to features relevant to their role. For instance, a customer can view their order history and track delivery status, while a seller can see order stats and update stock levels. The application also includes functionalities such as order status updates, secure checkout, and analytics dashboards for both sellers and admins.

The backend is powered by Node.js and Express, with MongoDB used for data storage. Mongoose models are used to define schemas and maintain relationships such as user orders, book-seller associations, and reviews. Features like JWT authentication, protected routes, data validation, and API error handling have been integrated to ensure the project is production-ready and secure. Overall, BookNest delivers a complete and real-world experience of an online book-selling platform.

SCENARIO-BASED CASE STUDY :

User Case – *Arjun (End User)*:

- Arjun is a B.Tech final-year student and an avid reader.
- He signs up on BookNest and logs into his user dashboard.
- He browses through various genres, authors, and books.
- He adds selected books to the **wishlist** and then to the **cart**.
- He places an order using the platform and receives a success message.
- Arjun can track his order status from the **"My Orders"** section.
- He receives timely updates as the seller changes the status (e.g., Processing → Shipped → Delivered).

Seller Case – *Swathi (Registered Seller)*:

- Swathi registers as a seller and gets access to the **Seller Dashboard**.
- She lists new books with stock, price, and image details.
- She receives a new order notification from Arjun.
- She views the order in the **"Seller Orders"** section.
- Swathi updates the **order status** to “Shipped” once dispatched.
- She views her **analytics** to track top-selling books and revenue generated.

Admin Case – *Ravi (System Administrator)*:

- Ravi logs in as the admin using admin credentials.
- He can view and manage all users, books, sellers, and orders from the **Admin Dashboard**.
- He has access to a powerful **Admin Analytics Panel** that shows:
    - Total registered users
    - Total sellers
    - Total books listed
    - Total revenue

    - Top-selling books
- Ravi ensures smooth functioning by taking actions like removing inappropriate books or handling complaints.

TECHNICAL ARCHITECTURE :

The **BookNest** project is built using a **MERN stack architecture**, enabling a full-stack JavaScript environment. The system follows a modular client-server model with a RESTful API interface, promoting scalable and maintainable development. The **frontend** is developed in React.js with Bootstrap for responsive design, while the **backend** uses Node.js and Express to manage API requests, business logic, and database operations.

MongoDB serves as the primary database with Mongoose for schema modeling. Authentication is handled using JSON Web Tokens (JWT), with role-based access control for Admins, Users, and Sellers.

All components are organized to support role-based dashboards, dynamic book listings, secure transactions, and user activity tracking. The architecture is optimized for extensibility, allowing future integration of third-party services like payment gateways and logistics APIs. The system supports secure login, book management, order tracking, and admin governance from a central admin panel.

Frontend Technologies:

- The frontend is developed using **React.js** for component-based architecture and fast rendering. **React Router DOM** is used for client-side routing, and **Axios** is used to communicate with backend APIs. **Bootstrap** and **React-Bootstrap** ensure a consistent, mobile-responsive UI.

Backend Technologies :

- The backend is powered by **Node.js** and **Express.js**, enabling asynchronous and scalable API handling. RESTful APIs are used for communication between client and server. Modular route definitions manage user actions, seller operations, and admin controls efficiently.

Database and Authentication :

- MongoDB is used as the NoSQL database, allowing flexibility in managing books, orders, users, and reviews. Mongoose is used for defining schemas. Authentication is handled using JWT, with hashed passwords via bcryptjs and middleware for protecting private routes.

Admin Panel and Governance :

- The admin panel allows centralized control over users, sellers, orders, and books. Admins can activate or block users/sellers, manage listings, and generate reports. Governance is role-based, ensuring each user accesses only permitted features and resources.

Scalability and Performance :

- BookNest is built using scalable components, allowing horizontal scaling of both frontend and backend. **Load balancing** can be implemented using Nginx or a cloud provider. API endpoints are optimized using pagination and indexing for fast query performance.

Time Management and Scheduling :

- Scheduled operations like inventory updates, order status changes, or notifications can be managed using **Node Cron**. This allows automated background jobs to ensure timely execution without manual intervention.

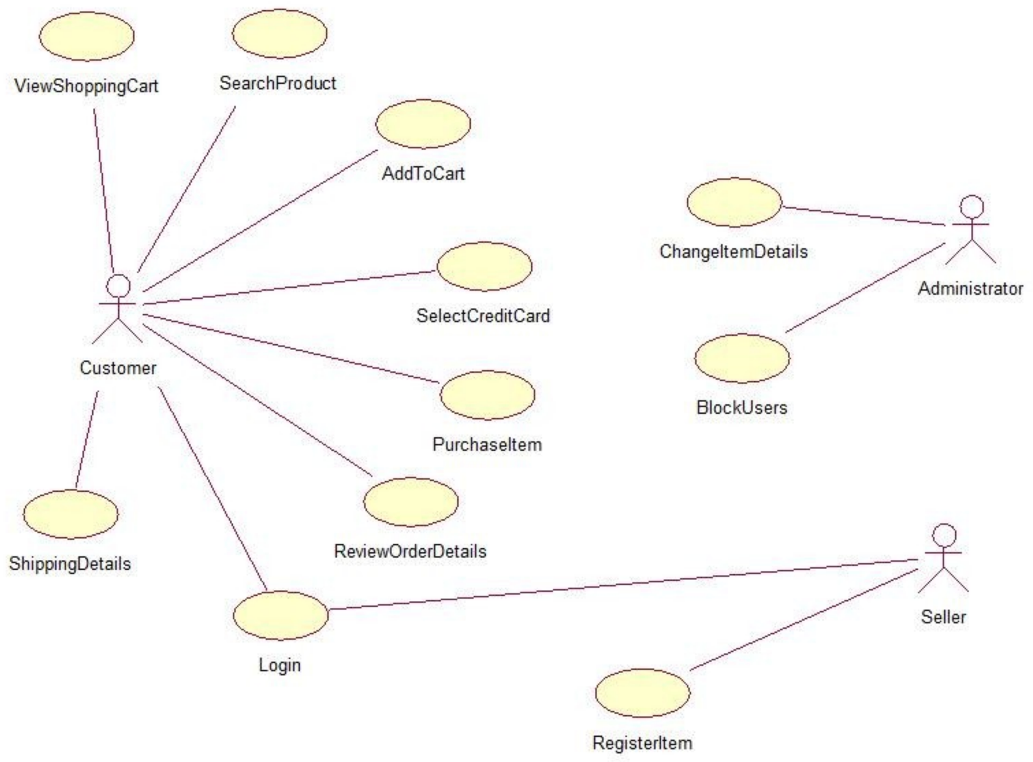
Security Features :

- The app includes robust security measures such as password hashing, JWT token verification, CORS policy, rate-limiting, and input validation to prevent common web vulnerabilities like XSS, CSRF, and brute-force attacks.

Notifications and Reminders :

- The system can integrate **email** or **SMS notifications** using services like Nodemailer or Twilio. Notifications are triggered for registration confirmation, order placement, status updates, and reminders for pending deliveries or feedback.

ER DIAGRAM :



PRE REQUISITES :

NODE.JS AND NPM :

- Node.js is a JavaScript runtime that allows you to run JavaScript code on the server-side. It provides a
- scalable platform for network applications.
- npm (Node Package Manager) is required to install libraries and manage dependencies.
- Download Node.js: Node.js Download
- Installation instructions: Installation Guide
- Run npm init to set up the project and create a package.json file.

EXPRESS.JS:

- Express.js is a web application framework for Node.js that helps you build APIs and web applications with features like routing and middleware.
- Install Express.js to manage backend routing and API endpoints.
- Install Express:
- Run npm install express

MONGODB:

- MongoDB is a NoSQL database that stores data in a JSON-like format, making it suitable for storing data like user profiles, doctor details, and appointments.
- Set up a MongoDB database for your application to store data.
- Download MongoDB: MongoDB Download
- Installation instructions: MongoDB Installation Guide

REACT.JS:

- React.js is a popular JavaScript library for building interactive and reusable user interfaces. It enables the development of dynamic web applications.
- Install React.js to build the frontend for your application.

HTML, CSS, AND JAVASCRIPT:

- Basic knowledge of HTML, CSS, and JavaScript is essential to structure, style, and add interactivity to the user interface.

DATABASE CONNECTIVITY (MONGOOSE):

- Use Mongoose, an Object-Document Mapping (ODM) library, to connect your Node.js backend to
- MongoDB for managing CRUD operations.
- Learn Database Connectivity: Node.js + Mongoose + MongoDB

FRONT-END FRAMEWORKS AND LIBRARIES:

- React.js will handle the client-side interface for managing doctor bookings, viewing appointment
- statuses, and providing an admin dashboard.
- You may use Material UI and Bootstrap to enhance the look and feel of the application.

SETUP AND INSTALLATION INSTRUCTIONS :

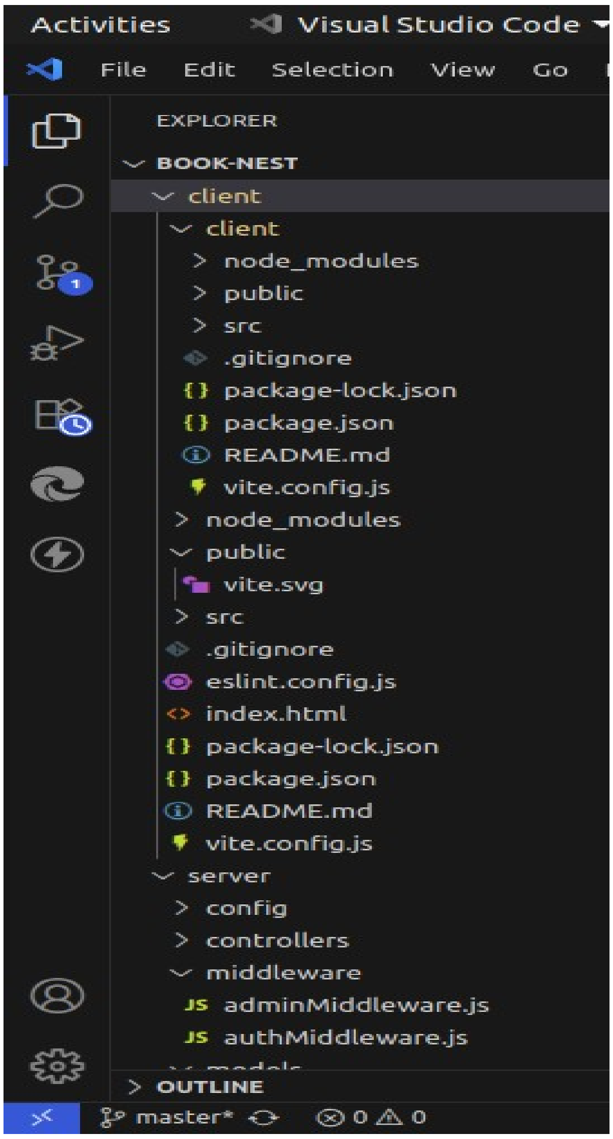
CLONE THE PROJECT REPOSITORY:

- Download the project files from GitHub or clone the repository using Git.

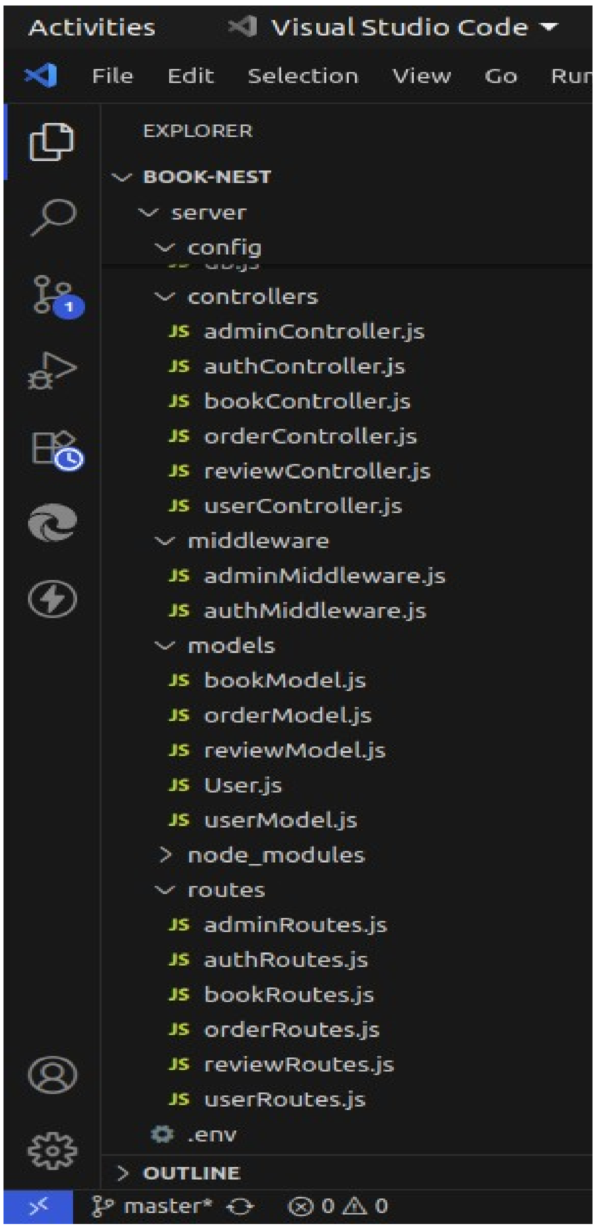
INSTALL DEPENDENCIES:

- Navigate to the frontend and backend directories and install all required dependencies for both parts of the application.
- **Frontend:**
  - Navigate to the frontend directory and run npm install.
- **Backend:**
  - Navigate to the backend directory and run npm install.
- **START THE DEVELOPMENT SERVER:**
- After installing the dependencies, start the development server for both frontend and backend.
- Frontend will run on http://localhost:3000.
- Backend will run on http://localhost:8001 or the specified port

PROJECT STRUCTURE :



```
EXPLORER
BOOK-NEST
  client
    client
      node_modules
      public
      src
      .gitignore
      package-lock.json
      package.json
      README.md
      vite.config.js
    node_modules
    public
      vite.svg
    src
    .gitignore
    eslint.config.js
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
  server
    config
    controllers
    middleware
      adminMiddleware.js
      authMiddleware.js
```



```
EXPLORER
BOOK-NEST
  server
    config
    controllers
      adminController.js
      authController.js
      bookController.js
      orderController.js
      reviewController.js
      userController.js
    middleware
      adminMiddleware.js
      authMiddleware.js
    models
      bookModel.js
      orderModel.js
      reviewModel.js
      User.js
      userModel.js
    node_modules
    routes
      adminRoutes.js
      authRoutes.js
      bookRoutes.js
      orderRoutes.js
      reviewRoutes.js
      userRoutes.js
    .env
```

The project is structured to include both frontend and backend components,each responsible for distinct tasks in the development of the Book Nest- Where stories Nestle webpage .Below is the flow ,describing the role and responsibilities of the users,sellers,and admin within the system .

FRONTEND PART :

The frontend of the **BookNest** application is developed using **React.js**, a powerful JavaScript library for building interactive and responsive user interfaces. It follows a **component-based architecture**, allowing efficient reuse of code, modular design, and ease of maintenance. The frontend communicates with the backend through **Axios** to send and receive data via RESTful APIs.

Technologies Used:

- **React.js**: For building dynamic and modular UI components.
- **React Router DOM**: For handling client-side routing and navigation between pages.
- **Axios**: For making HTTP requests to backend APIs.
- **Bootstrap & React-Bootstrap**: For building responsive, mobile-friendly layouts and styling UI elements.
- **HTML5 & CSS3**: For structuring and styling the application.
- **JavaScript (ES6+)**: For dynamic behavior, form validations, and state handling.

Key Components:

- **Landing Page**: Welcomes users and provides an overview of the platform.
- **Login/Register Pages**: Allow users, sellers, and admins to sign in or register securely.
- **Home Page**: Displays featured books and categories.
- **Books Listing Page**: Shows all available books with filters by genre, author, or price.
- **Book Details Page**: Displays in-depth information about a selected book.
- **User Dashboard**: Shows order history, wishlist, and profile management options.
- **Seller Dashboard**: Allows sellers to manage book listings, inventory, and view orders.
- **Admin Dashboard**: Admins can view and manage users, sellers, books, and reports.

Functionality Highlights:

- **Client-Side Routing**: React Router is used to navigate between components without page reloads.

- **State Management**: React's `useState` and `useEffect` hooks are used for managing component-level state and side effects.
- **Form Handling**: User inputs are validated, submitted to APIs, and provide real-time feedback.
- **Authentication Integration**: Token-based login persists user sessions and displays role-specific dashboards.
- **Responsive Design**: Bootstrap grid system ensures the app works well on desktop, tablet, and mobile devices.

BACKEND PART :

The backend of the **BookNest** application is developed using **Node.js** with the **Express.js** framework. It acts as the core engine of the application, handling business logic, routing, database communication, authentication, and secure data operations. The backend is designed to be **RESTful**, scalable, and secure, supporting seamless communication with the frontend through APIs.

Technologies Used:

- **Node.js**: JavaScript runtime for building fast and scalable backend services.
- **Express.js**: Minimal and flexible web application framework for handling routes, middleware, and API endpoints.
- **MongoDB**: NoSQL database for storing structured collections like users, books, orders, inventory, and reviews.
- **Mongoose**: ODM library for modeling and interacting with MongoDB data.
- **bcrypt.js**: Used for securely hashing passwords before storing them in the database.
- **jsonwebtoken (JWT)**: Used for generating and verifying tokens for secure user authentication.
- **dotenv**: For loading environment variables from the `.env` file into the application.

Key Modules and Features:

- **User Authentication**: Includes secure registration, login, and logout functionality using JWT. Role-based access is implemented for Users, Sellers, and Admins.
- **Book Management APIs**: CRUD operations for books, including adding, editing, deleting, and listing books by sellers or admins.
- **Order Management**: Handles placing orders, updating statuses, tracking orders, and storing order history for each user.
- **Inventory Handling**: Maintains stock levels of books and updates inventory automatically after purchases.
- **Review & Rating System**: Users can submit reviews and ratings for purchased books.

- **Admin Controls**: Admin can manage users, view system analytics, and monitor platform activities.

API Endpoints Overview:

- `POST /api/register` – Register a new user or seller
- `POST /api/login` – User login and token generation
- `GET /api/books` – Fetch all books (user view)
- `POST /api/books` – Add a new book (seller/admin)
- `PUT /api/books/:id` – Update book details
- `DELETE /api/books/:id` – Remove a book
- `POST /api/orders` – Place an order
- `GET /api/orders/user/:id` – Get user-specific order history

Security and Middleware:

- **JWT Middleware**: Protects private routes and verifies user roles.
- **Input Validation**: Ensures only valid data is accepted and stored.
- **CORS Handling**: Enables frontend and backend communication across different origins.
- **Error Handling**: Centralized error middleware handles unexpected errors and sends proper responses.

APPLICATION FLOW :

This project has three main types of users: Customer,Seller, and Admin .Each has specific roles and responsibilities that are definedby the ApI endpoints.

User Flow (Customer)

1. **User Registration/Login**
    - A new user signs up or logs in through the user authentication system.
    - On successful login, a JWT token is issued and stored securely.
2. **Browsing and Searching Books**
    - Users can view books by category, genre, author, or title using the homepage and filter options.
    - Clicking on a book displays detailed information (description, price, reviews, etc.).
3. **Adding to Wishlist or Cart**

- Users can add books to their wishlist or directly to the cart.

4. **Placing an Order**
    - Users proceed to checkout and place an order.
    - Order details are stored in the database and linked to the user’s profile.
5. **Order History and Reviews**
    - Users can view their previous orders, track order statuses, and give reviews/ratings for books.

Seller Flow (Organizer/Vendor) :

1. **Seller Registration/Login**
    - Sellers register or login through a dedicated seller portal.
    - Upon login, they receive a token and are redirected to the seller dashboard.
2. **Managing Books and Inventory**
    - Sellers can add, update, or delete their book listings.
    - They can manage inventory, set prices, and upload book details with cover images.
3. **Order Fulfillment**
    - Sellers receive notifications when users place orders for their books.
    - They can update the order status (e.g., processing, shipped, delivered).
4. **View Seller Reports**
    - Sellers can view analytics such as best-selling books, stock levels, and monthly orders.

Admin Flow (System Administrator) :

1. **Admin Login**
    - Admins log in through the admin portal with secure credentials.
2. **Managing Users and Sellers**
    - Admins can view, approve, block, or delete user and seller accounts.
    - They maintain control over platform safety and user behavior.
3. **Book and Order Monitoring**
    - Admins can see all books listed on the platform and flag inappropriate listings.
    - Admins oversee all orders, disputes, and platform-wide statistics.
4. **Dashboard Analytics**

- Admins have access to graphical reports on daily users, sales, inventory levels, and active sellers.

Navigation and Access Control :

- The application uses **React Router DOM** for page-level navigation.
- **Role-based access control (RBAC)** ensures that each user role only accesses their permitted pages and actions.
- Protected routes use **JWT verification middleware** to secure private pages (like dashboards and order management).

SETUP & CONFIGURATION :

Setting up the BookNest Webpage involves configuring both the Frontend (React.js) and Backend(Node.js, Express.js, MongoDB) to ensure the application runs smoothly. Below are the steps to set up and configure the environment for your project.

FRONTEND CONFIGURATION :

INSTALLATION :

Clone the Repository:

- Clone the project from GitHub to your local machine:
- bash
- Copy code
- git clone <your-repository-url>
- Replace <your-repository-url> with the URL of your project repository.

Navigate to Frontend Directory:

- After cloning, navigate to the frontend folder where the React.js app is located:
- bash
- Copy code
- cd booknest/client

Install Dependencies:

- Use npm (Node Package Manager) to install the necessary dependencies:
- bash
- Copy code
- npm install
- This will install all the required libraries, such as React, React Router, Ant Design, and others.

Run the React Development Server:

- To start the frontend server and run the React application:
- bash
- copy code
- npm start
- The application will be available at http://localhost:3000 in your browser

BACKEND CONFIGURATION :

INSTALLATION :

Navigate to Backend Directory:

- Move to the backend folder of your project:
- bash
- Copy code
- cd booknest/server
- Install Dependencies:
- Install the necessary backend dependencies using npm:
- bash
- Copy code
- npm install
- This will install all required libraries for Node.js and Express.js, such as express, mongoose, bcryptjs,
- jsonwebtoken, etc.

Configure MongoDB :

- Ensure you have MongoDB installed on your local machine or use a cloud-based MongoDB service like
- MongoDB Atlas.
- In the backend, open the configuration file (e.g., config/db.js or .env) and configure the connection URL
- for MongoDB:
- bash
- Copy code
- MONGO_URI=mongodb://localhost:27017/booknest
- If you are using MongoDB Atlas, replace the localhost part with your MongoDB Atlas connection
- string.
- **Set Up Environment Variables:**

- Create a .env file in the backend directory to store environment-specific variables such as:
- bash
- Copy code
- PORT=5000
- JWT_SECRET=your_jwt_secret
- Make sure to replace your_jwt_secret with a strong secret key for JWT authentication.

Run the Backend Server:

- Start the backend server by running:
- bash
- Copy code
- npm start
- The backend server will be running at http://localhost:5000

DATABASE CONFIGURATION (MONGODB) :

Install MongoDB (Local Installation):

- If you are using a local MongoDB instance, download and install it from the official MongoDB website:
- Download MongoDB
- **Set Up MongoDB Database:**
- After installation, start the MongoDB service:
- bash
- Copy code
- mongod
- This will run MongoDB on the default port 27017.

MongoDB Atlas (Cloud-based MongoDB):

- If you prefer to use MongoDB Atlas, create an account on MongoDB Atlas and create a cluster.
- Once the cluster is set up, get the connection string and replace it in the backend’s .env file:
- bash
- Copy code
- MONGO_URI=<your-mongodb-atlas-connection-string>

FINAL CONFIGURATION & RUNNING THE APP :

Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function
- correctly.
- You can open two terminal windows or use tools concurrently to run both servers together.
- To install concurrently, run:
- bash
- Copy code
- npm install concurrently --save-dev
- In your package.json file, add a script to run both servers:
- json
- Copy code

```
"scripts": {
 "start": "concurrently \"npm run server\" \"npm run client\"",
 "server": "node backend/server.js",
 "client": "npm start --prefix frontend"
}
```

Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash
- Copy code
- npm start
- The backend will be available at http://localhost:5000, and the frontend at http://localhost:3000.

VERIFYING THE APP :

Check Frontend:

- Open your browser and go to http://localhost:3000. The React.js application should load with the list of doctors,bookingforms and status updates.

Check Backend:

- Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as
- user login, doctor registration, and appointment creation.

ADDITIONAL SETUP :

Version Control:

- If you haven't already done so, initialise a Git repository in the root of your project:
- bash
- Copy code
- git init
- Add your project files and commit them:
- bash
- Copy code
- git add .
- git commit -m "Initial commit"
- Deployment (Optional):
- If you want to deploy your app, consider using cloud platforms like Heroku, AWS, or Vercel for frontend hosting and backend API deployment

Project Root Structure :

The **BookNest** project follows a modular structure with separate folders for **frontend** and **backend**, making the application easy to maintain and scale.

```
bash
CopyEdit
BookNest/
│
├── client/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── routes/
│   │   └── App.jsx
│   ├── package.json
│   └── vite.config.js
│
├── server/
│   ├── config/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── utils/
│   ├── .env
│   ├── server.js
│   └── package.json
│
├── README.md
└── .gitignore
```

Backend Setup (Node.js + Express + MongoDB) :

Steps:

1. Navigate to the backend folder:

```
bash
CopyEdit
cd server
```

2. Create a `.env` file:

```
env
CopyEdit
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_jwt_secret_key
```

3. Start the server:

```
bash
CopyEdit
npm run dev
```

Backend Packages to Install:

```
bash
CopyEdit
npm install express mongoose dotenv cors bcryptjs jsonwebtoken
npm install --save-dev nodemon
```

Frontend Setup (React.js + Vite + Bootstrap) :

Steps:

1. Navigate to the frontend folder:

```
bash
CopyEdit
cd client
```

2. Initialize React app using Vite:

```
bash
CopyEdit
npm create vite@latest
```

3. Install dependencies:

```
bash
CopyEdit
```

```
npm install
```

4. Start the development server:

```
bash
CopyEdit
npm run dev
```

Frontend Packages to Install:

```
bash
CopyEdit
npm install react-router-dom axios bootstrap react-bootstrap
```

PROJECT FLOW :

PROJECT DEMO :

- Before diving into development ,you can view a demo of the project to understand its functionality and user interactions.
- Project Demo Video
- https://drive.google.com/file/d/1uys96BqoLra1kjWmKNHzwDLq_hcPWX3Y/view?usp=sharing
- project code Repository

  https://github.com/tagore3589/Book-Nest.git
- The source code for this project can be accessed and cloned from github,providing a base structure and example code for further customization and understanding .

MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

1. Install required tools and software:

- Node.js.
- MongoDB.
- Create-react-app.

2. Create project folders and files:
- Client folders.
- Server folders.

3. Install Packages:
Frontend npm Packages
- Axios.
- React-Router –dom.
- Bootstrap.

- React-Bootstrap.

Backend npm Packages

- Express.
- Mongoose.
- Cors.

PROJECT FOLDERS:

- Frontend Folder: Contains all code related to the user interface, written in JavaScript using frameworks and libraries like React, Material UI, and Bootstrap. This setup helps maintain a clear boundary between UI logic and server logic.
- Backend Folder: Manages the server, API routes, and database interactions, typically handled through Node.js and Express.js. Using separate folders enables a modular structure, allowing changes in one area without affecting the other.

LIBRARY AND TOOL INSTALLATION:

Backend Libraries:

- Node.js: Provides a runtime environment to run JavaScript code on the server side.
- MongoDB: A NoSQL database, perfect for flexible and schema-less data storage, ideal for applications needing frequent updates and various data types.
- Bcrypt: Encrypts passwords for secure authentication, helping protect user data from potential breaches.
- Body-parser: Parses incoming request bodies, making it easy to access data in various formats like JSON.
- Frontend Libraries:
- React.js: Manages component-based UI creation, providing the flexibility to build reusable UI components.
- Material UI & Bootstrap: Provides styling frameworks, ensuring a consistent,responsive, and visually appealing design.
- Axios: Facilitates easy HTTP requests, allowing the frontend to communicate with the backend effectively.

MILESTONE 2: BACKEND DEVELOPMENT :

- The backend forms the core logic and data management for the project. A well-structured backend ensures efficient data handling, security, and scalability.

EXPRESS.JS SERVER SETUP:

- Express Server: Acts as a hub for all requests and responses, routing them to appropriate endpoints. It's
- Essential for managing incoming requests from the frontend, processing them, and sending responses back.

Middleware Configuration:

- Middleware like body-parser parses JSON data in requests, while cors enables cross-origin communication between the frontend and backend. Middleware makes it easy to add additional functionality, like error handling or data validation, without interfering with core application logic.

API ROUTE DEFINITION:

- Route Organization: Organizing routes by functionality (e.g., authentication, appointments, complaints) keeps the codebase readable and easy to maintain. For instance, all authentication-related routes can reside in an auth.js file, ensuring each file has a single purpose.
- Express Route Handlers: Route handlers manage the flow of data between client and server, such as fetching, creating, updating, and deleting records.
- **DATA MODELS (SCHEMAS) WITH MONGOOSE:**
- User Schema: Defines structure for user data and includes fields for personal information and role-based access (e.g., doctor, customer, admin). Using a schema ensures consistent data storage.
- Appointment and Complaint Models: Models like Appointment and Complaint manage complex data interactions, including relationships between users and appointments, allowing efficient querying.
- CRUD Operations: CRUD functionalities (Create, Read, Update, Delete) provide a standard interface for handling data operations. They simplify data manipulation in a structured, predictable way.
- **USER AUTHENTICATION:**

- JWT Authentication: JSON Web Tokens securely handle session management, ensuring that only verified users access protected routes. JWT tokens are embedded in request headers, verifying each request without storing session data on the server.
- **ADMIN AND TRANSACTION HANDLING:**
- Admin Privileges:
- Administrators oversee user registrations, approve appointments, and manage doctor applications. Admin-specific routes ensure that these actions are isolated and secure.
- Transaction Management:
- This functionality allows customers to interact with appointments, booking history, and cancellations in real-time.
- ERROR HANDLING:
- Middleware for Error Handling: Error-handling middleware catches issues and sends meaningful error responses with HTTP status codes (like 404 for Not Found or 500 for Server Error), enabling better debugging and user feedback.

MILESTONE 3: DATABASE DEVELOPMENT

- Using MongoDB as the database provides a flexible, schema-less structure, perfect for handling different types of user and appointment data.

SCHEMAS FOR DATABASE COLLECTIONS:

User Schema:

- Defines fields for user information like name, email, password, and userType. This
- schema allows fine-grained control over user data and easy retrieval of information.
- Complaint and Assigned Complaint Schemas:
- These schemas manage complaint data, with fields linking complaints to users and statuses. They allow efficient tracking of complaints and status updates by linking agents to users.
- Chat Window Schema:
- This schema organises messages between users and agents, storing them by
- complaint ID for a streamlined user-agent communication flow.

- DATABASE COLLECTIONS IN MONGODB:
- MongoDB collections, such as users, complaints, and messages, provide a structured, NoSQL approach to data management, making it easy to scale as data grows.

MILESTONE 4: FRONTEND DEVELOPMENT

- Frontend development focuses on creating an interactive, intuitive user experience through a React-based user interface.
- REACT APPLICATION SETUP:
- Folder Structure and Libraries: Setting up the initial React app structure and libraries ensures a smooth development workflow. By organising files into components, services, and pages, the project becomes easy to navigate and maintain.
- UI Component Libraries: Material UI and Bootstrap offer pre-built components, enabling rapid UI development and consistent design across all screens.
- UI COMPONENTS FOR REUSABILITY:
- Reusable Components:
- Each UI element, like forms, dashboards, and buttons, is designed as a reusable component. This modularity allows efficient reuse across the app, reducing development time and ensuring consistency.

Styling and Layout:

- Styling and layout components maintain a cohesive look and feel, contributing to the user experience with clean, intuitive visuals.

FRONTEND LOGIC IMPLEMENTATION:

- API Integration: Axios is used to make API calls to the backend, connecting UI components with data from the server.
- Data Binding and State Management: React’s state management binds data to the UI, automatically updating it as the user interacts with the app.

MILESTONE 5: PROJECT IMPLEMENTATION AND TESTING:

- After completing development, running a final set of tests is crucial for identifying any bugs or issues.

VERIFY FUNCTIONALITY:

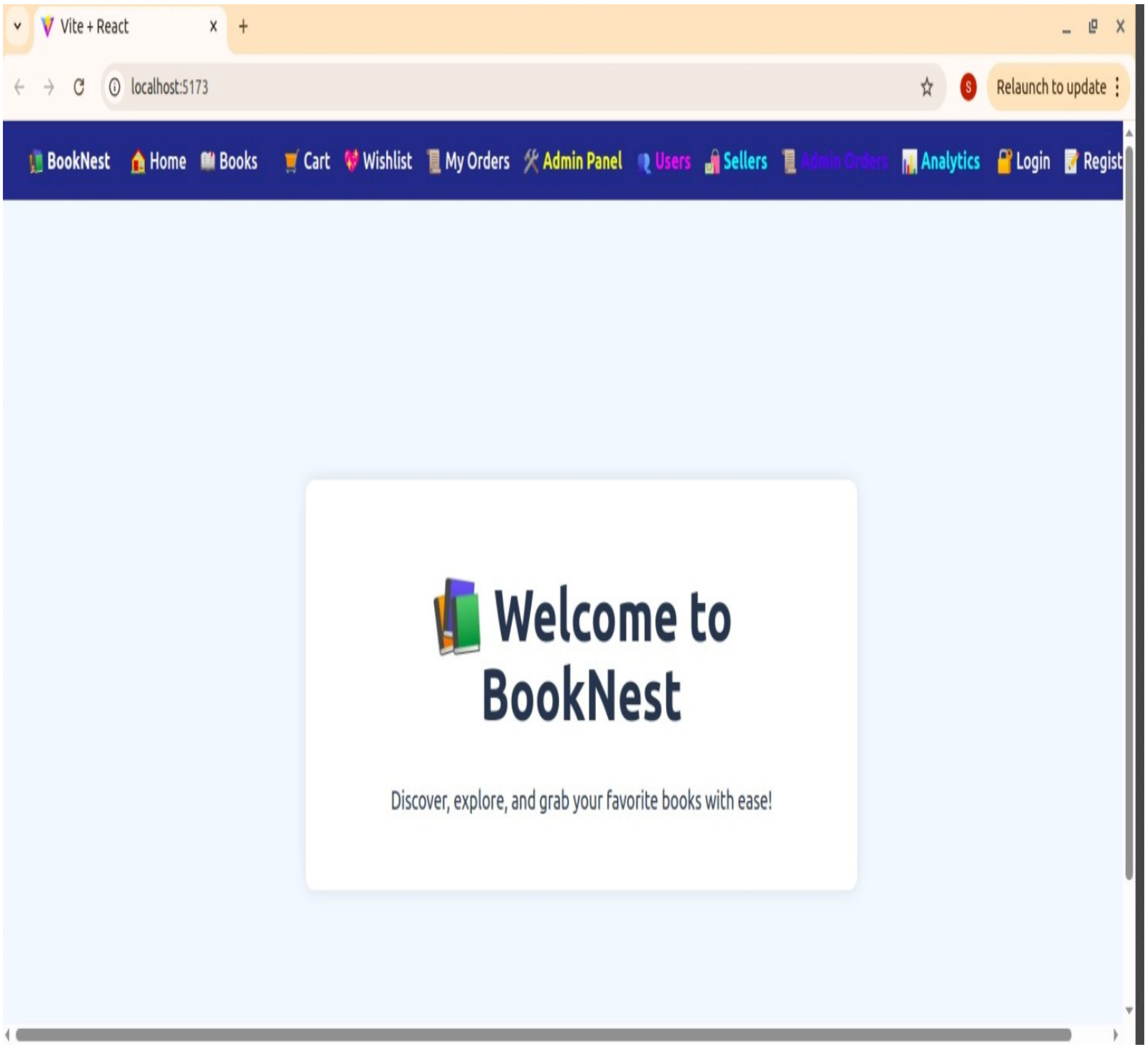
Running the entire application ensures that each part (frontend, backend, database) works cohesively.Testing various user flows (e.g., booking, cancelling, updating appointments) helps confirm that all processes are functioning as intended.

USER INTERFACE ELEMENTS:

- Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.
- Ensuring responsive design and usability across devices and screen sizes is also essential.

- FINAL DEPLOYMENT:
- Once testing is complete, the application can be deployed to a production server, making it accessible to end-users.

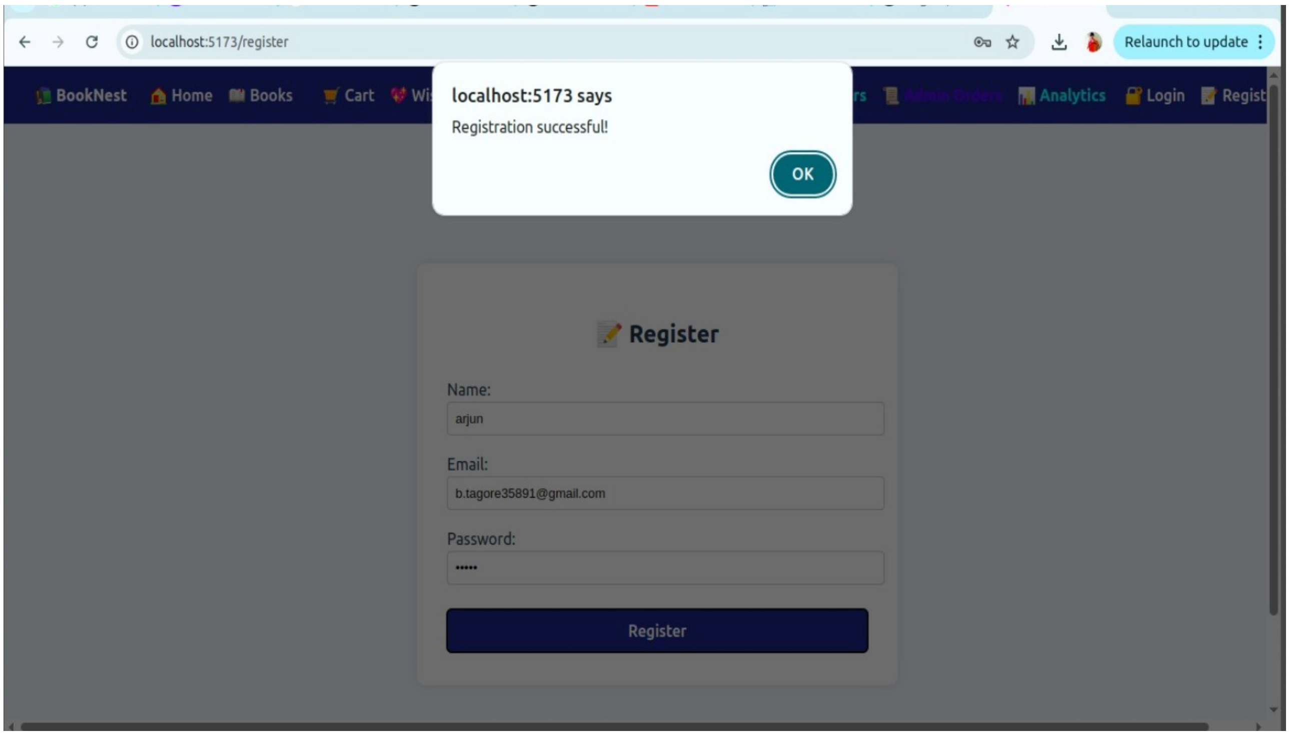
Home Page:



Welcome to BookNest

Discover, explore, and grab your favorite books with ease!

REGISTER PAGE:


localhost:5173/register

localhost:5173 says

Registration successful!

OK

📝 Register

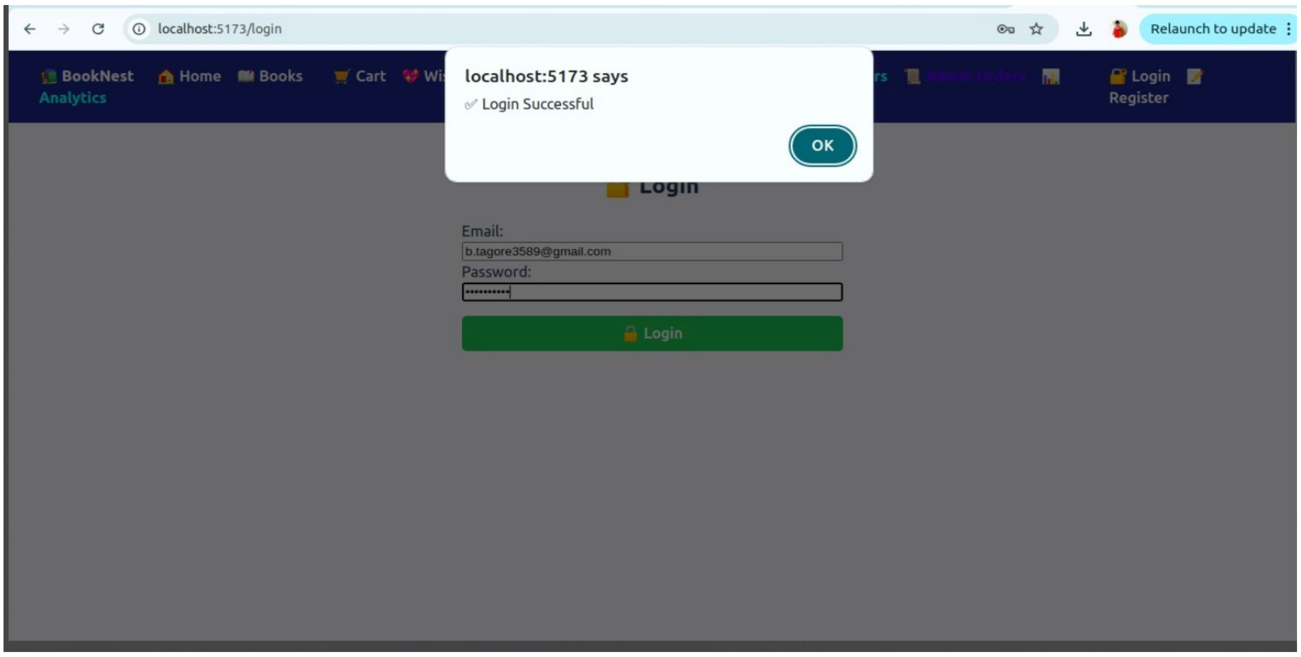
Name: arjun

Email: b.tagore35891@gmail.com

Password: •••••

Register

LOGIN PAGE:


localhost:5173/login

localhost:5173 says

✅ Login Successful

OK

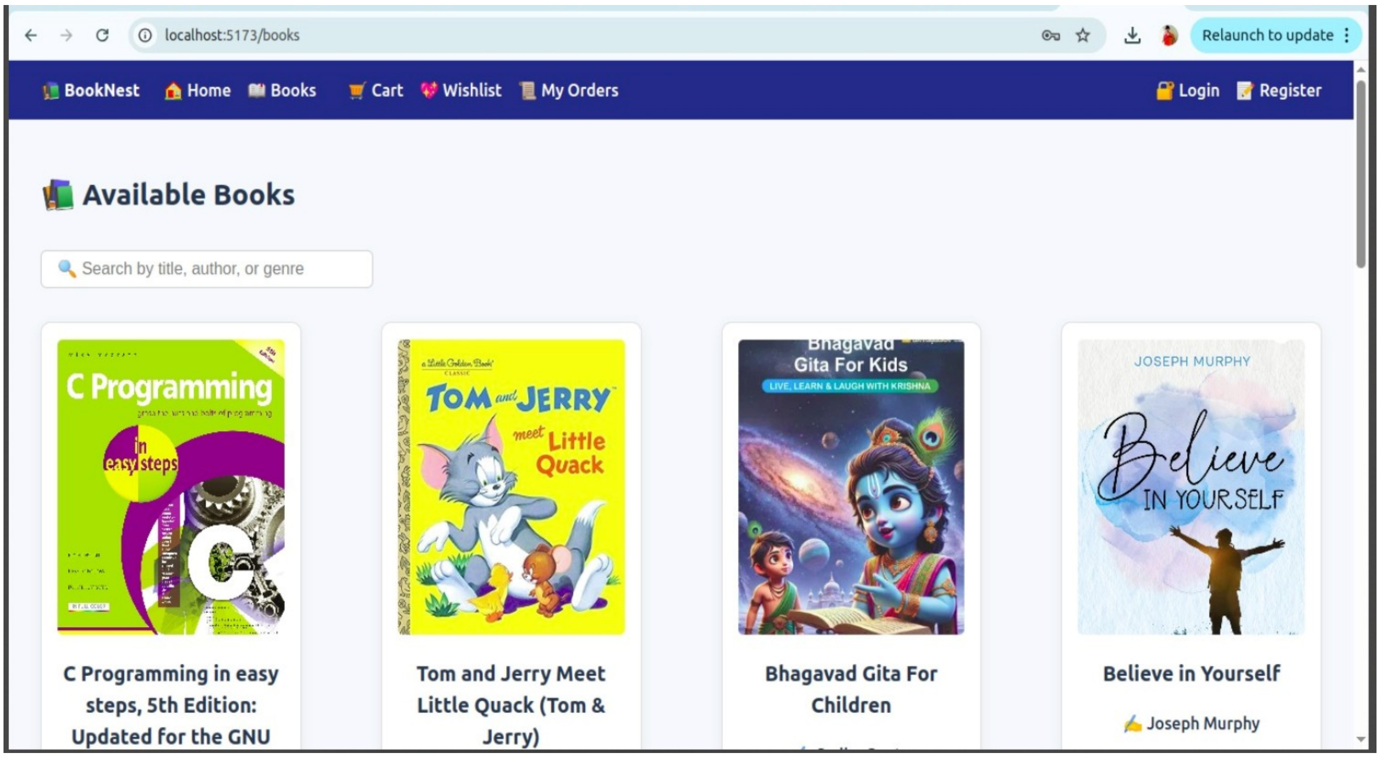
🔐 Login

Email: b.tagore3589@gmail.com

Password: ••••••••••

🔐 Login

BOOKS PAGE:


localhost:5173/books

📚 BookNest 🏠 Home 📖 Books 🛒 Cart 💖 Wishlist 📜 My Orders 🔐 Login 📝 Register

📚 Available Books

🔍 Search by title, author, or genre

C Programming in easy steps, 5th Edition: Updated for the GNU

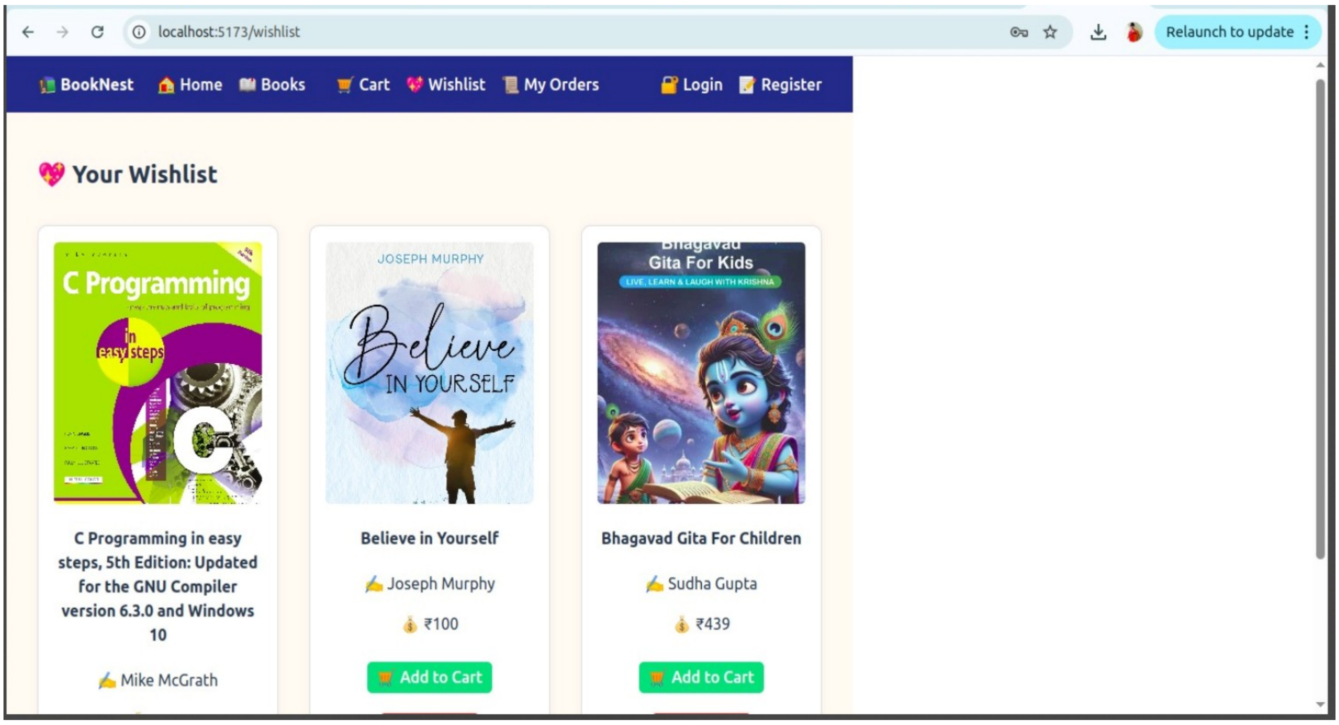
Tom and Jerry Meet Little Quack (Tom & Jerry)

Bhagavad Gita For Children

Believe in Yourself

✍ Joseph Murphy

WISHLIST PAGE:


localhost:5173/wishlist

📚 BookNest 🏠 Home 📖 Books 🛒 Cart 💖 Wishlist 📜 My Orders 🔐 Login 📝 Register

💖 Your Wishlist

C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10

✍ Mike McGrath

Believe in Yourself

✍ Joseph Murphy

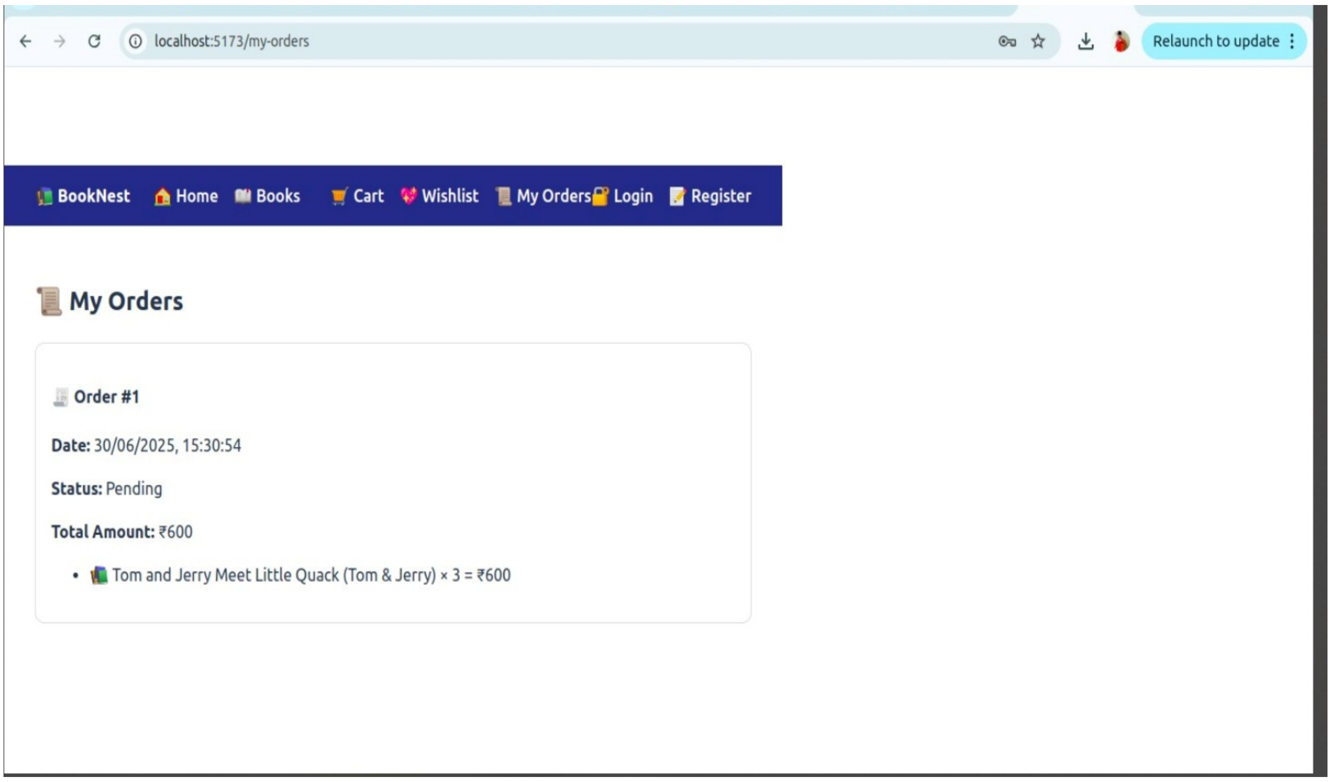
💰 ₹100

Bhagavad Gita For Children

✍ Sudha Gupta

💰 ₹439

MY ORDERS :



My Orders

Order #1

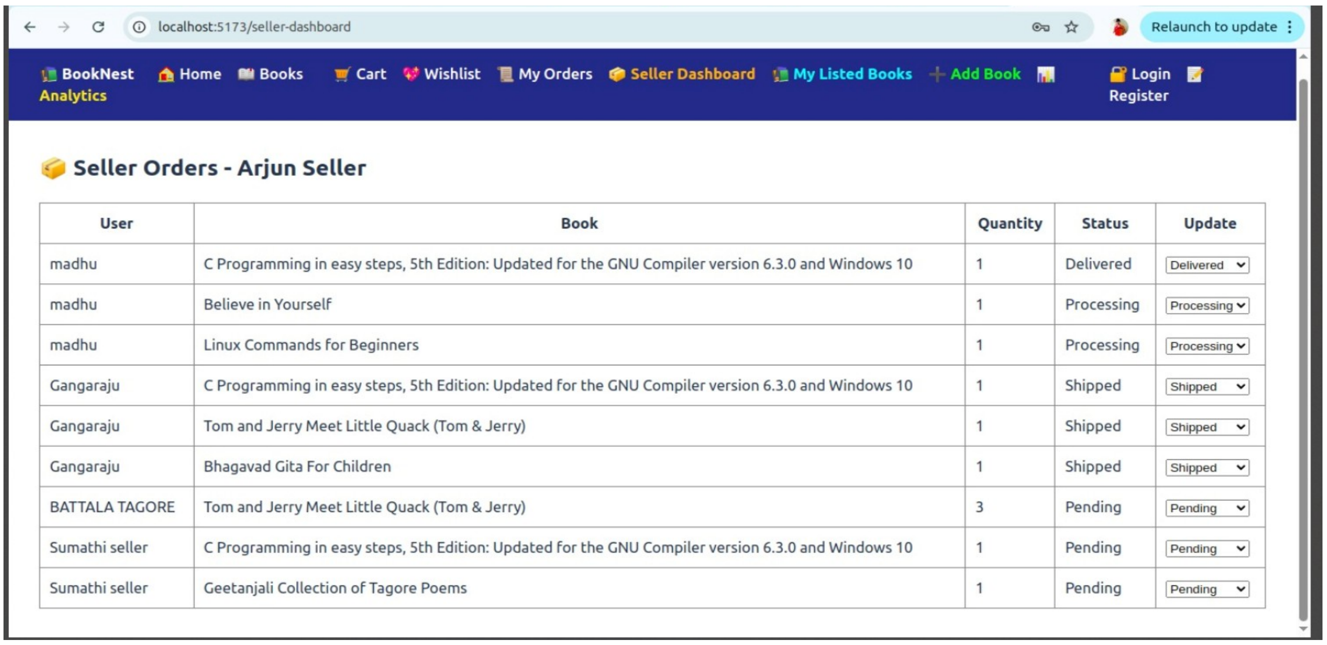
Date: 30/06/2025, 15:30:54

Status: Pending

Total Amount: ₹600

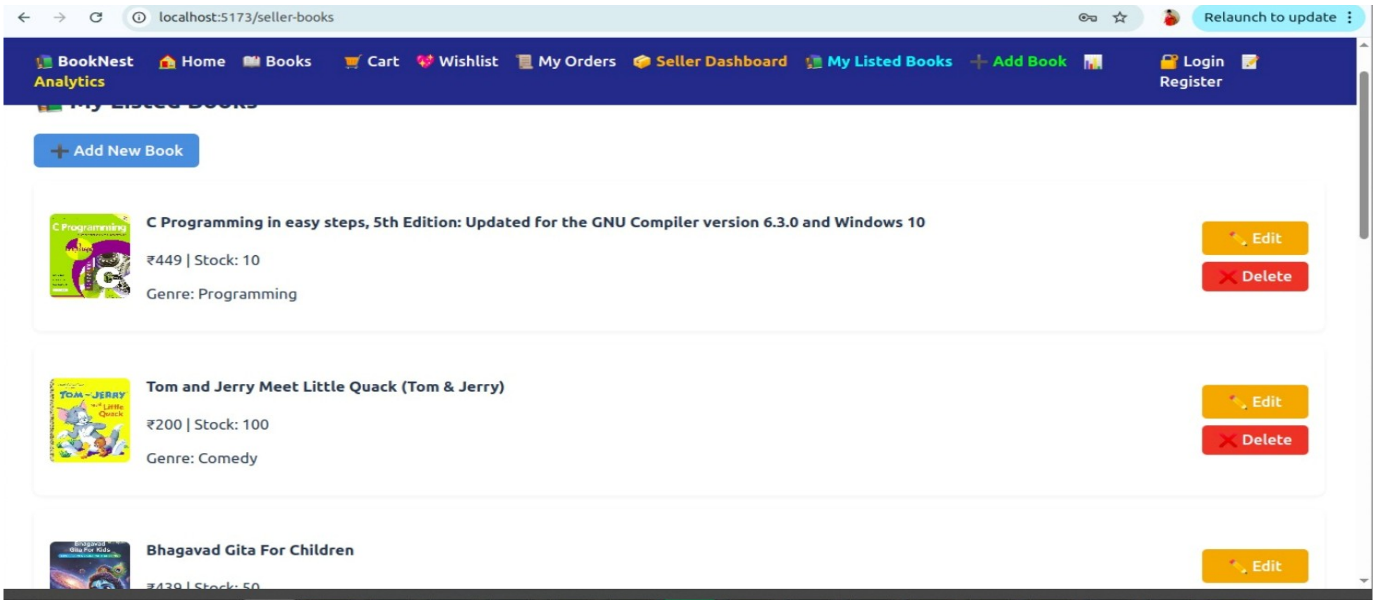
- Tom and Jerry Meet Little Quack (Tom & Jerry) × 3 = ₹600

SELLER DASHBOARD:



Seller Orders - Arjun Seller

User	Book	Quantity	Status	Update
madhu	C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10	1	Delivered	Delivered
madhu	Believe in Yourself	1	Processing	Processing
madhu	Linux Commands for Beginners	1	Processing	Processing
Gangaraju	C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10	1	Shipped	Shipped
Gangaraju	Tom and Jerry Meet Little Quack (Tom & Jerry)	1	Shipped	Shipped
Gangaraju	Bhagavad Gita For Children	1	Shipped	Shipped
BATTALA TAGORE	Tom and Jerry Meet Little Quack (Tom & Jerry)	3	Pending	Pending
Sumathi seller	C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10	1	Pending	Pending
Sumathi seller	Geetanjali Collection of Tagore Poems	1	Pending	Pending



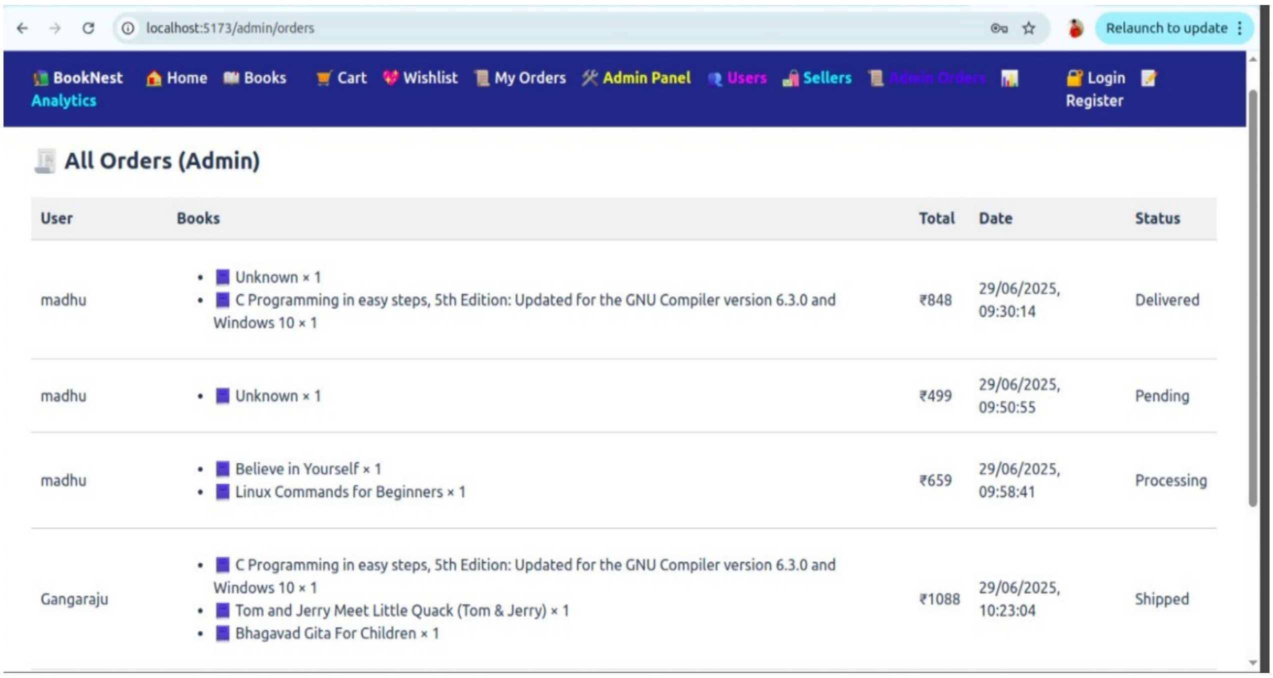
My Listed Books

C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10
₹449 | Stock: 10
Genre: Programming

Tom and Jerry Meet Little Quack (Tom & Jerry)
₹200 | Stock: 100
Genre: Comedy

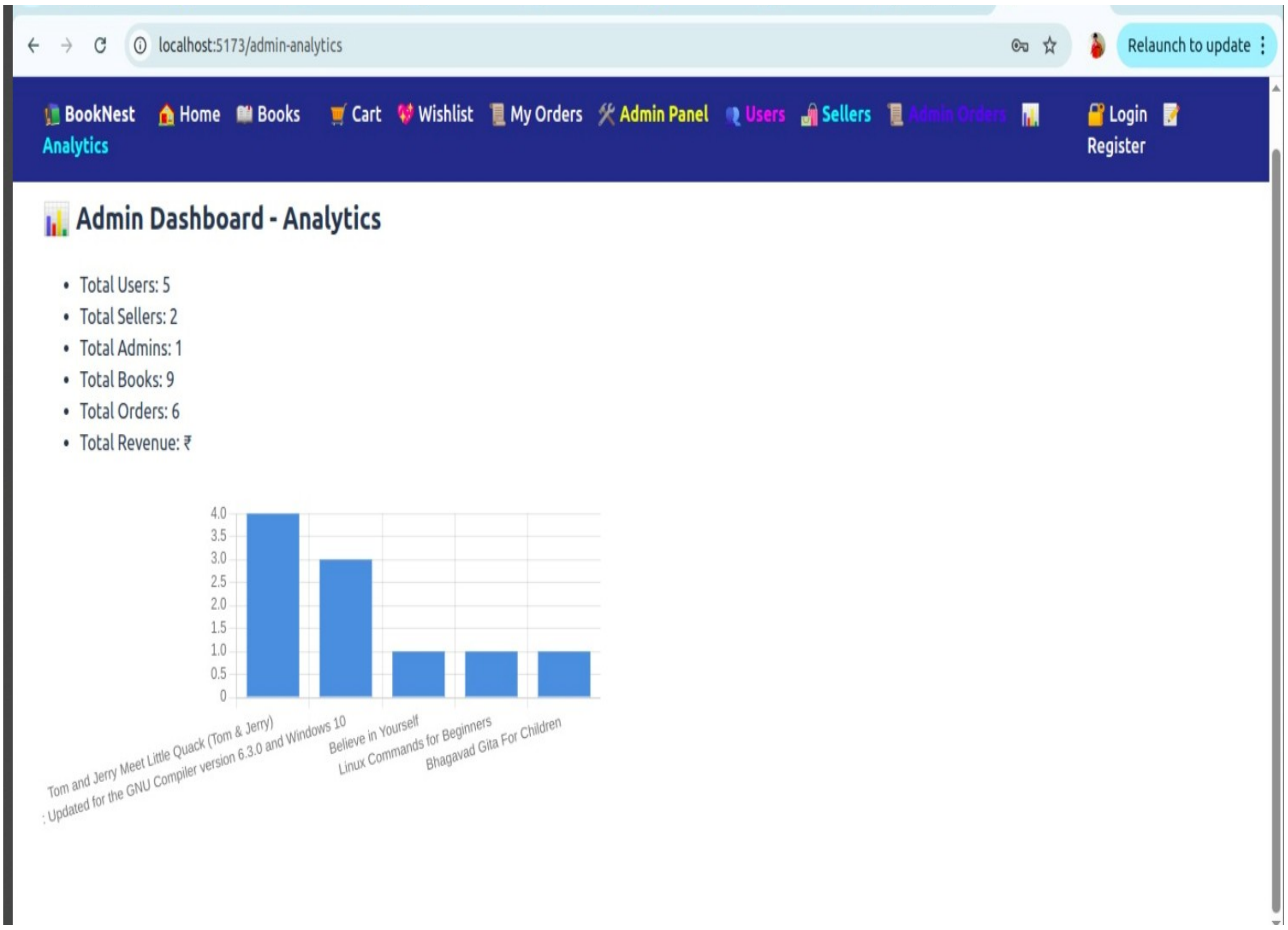
Bhagavad Gita For Children
₹439 | Stock: 50

ADMIN DASHBOARD:



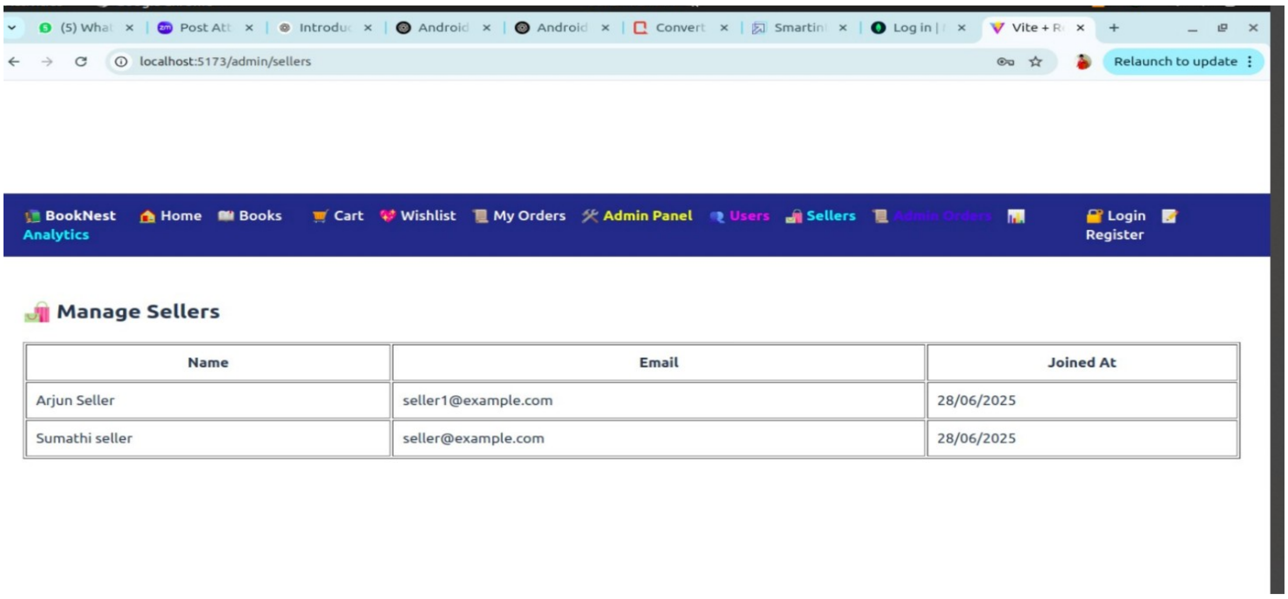
All Orders (Admin)

User	Books	Total	Date	Status
madhu	• Unknown × 1 • C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10 × 1	₹848	29/06/2025, 09:30:14	Delivered
madhu	• Unknown × 1	₹499	29/06/2025, 09:50:55	Pending
madhu	• Believe in Yourself × 1 • Linux Commands for Beginners × 1	₹659	29/06/2025, 09:58:41	Processing
Gangaraju	• C Programming in easy steps, 5th Edition: Updated for the GNU Compiler version 6.3.0 and Windows 10 × 1 • Tom and Jerry Meet Little Quack (Tom & Jerry) × 1 • Bhagavad Gita For Children × 1	₹1088	29/06/2025, 10:23:04	Shipped


localhost:5173/admin-analytics
Relaunch to update
BookNest
Home
Books
Cart
Wishlist
My Orders
Admin Panel
Users
Sellers
Admin Orders
Analytics
Login
Register
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
Tom and Jerry Meet Little Quack (Tom & Jerry)
: Updated for the GNU Compiler version 6.3.0 and Windows 10
Believe in Yourself
Linux Commands for Beginners
Bhagavad Gita For Children

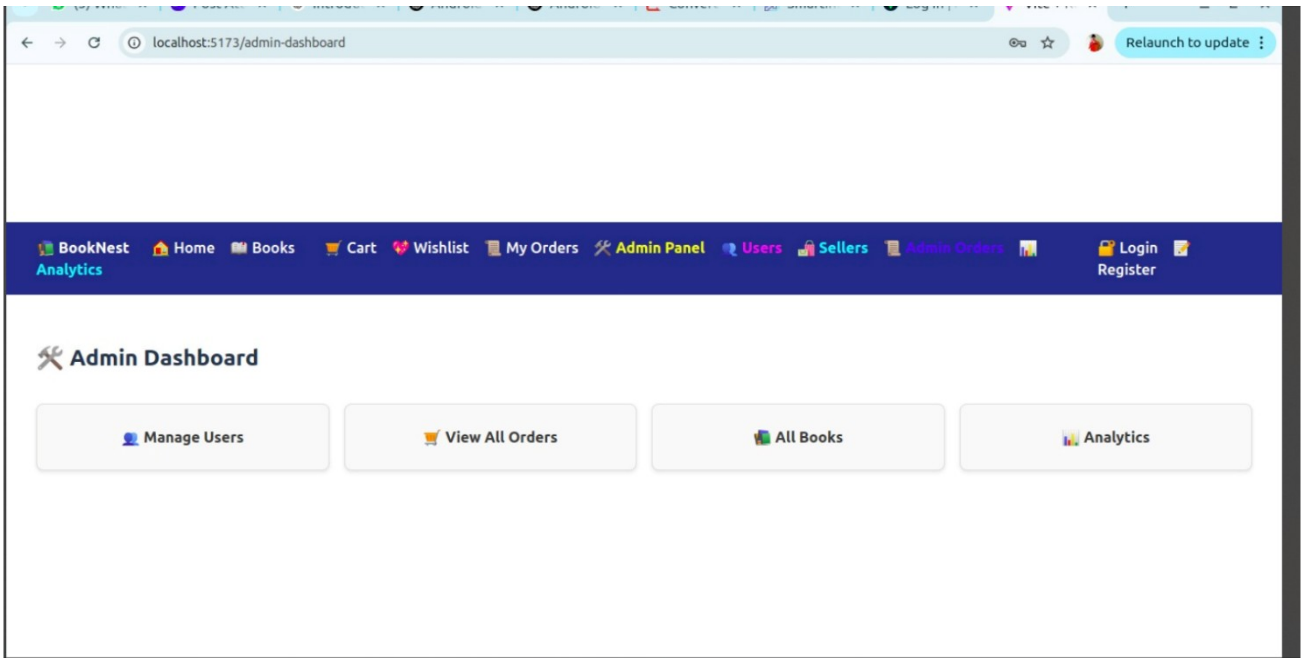
📊 Admin Dashboard - Analytics

- Total Users: 5
- Total Sellers: 2
- Total Admins: 1
- Total Books: 9
- Total Orders: 6
- Total Revenue: ₹

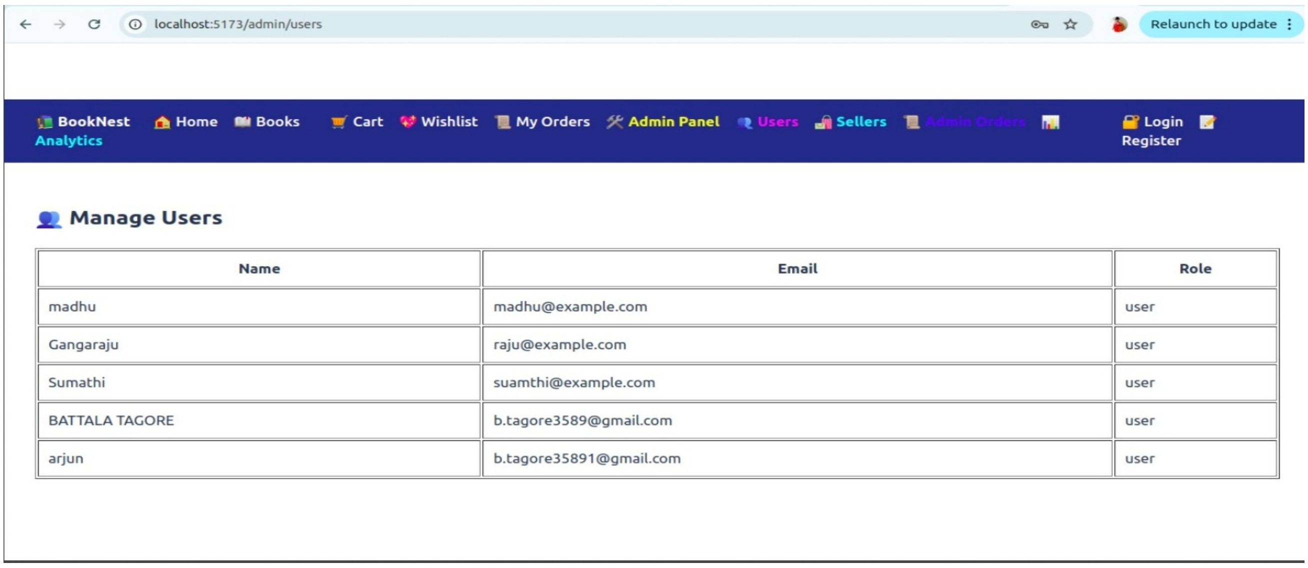


Manage Sellers

Name	Email	Joined At
Arjun Seller	seller1@example.com	28/06/2025
Sumathi seller	seller@example.com	28/06/2025



Admin Dashboard



Manage Users

Name	Email	Role
madhu	madhu@example.com	user
Gangaraju	raju@example.com	user
Sumathi	suamthi@example.com	user
BATTALA TAGORE	b.tagore3589@gmail.com	user
arjun	b.tagore35891@gmail.com	user

CONCLUSION :

The BookNest project is a full-stack web application built using the MERN stack that simulates a real-world online book store. It includes three main roles—User, Seller, and Admin—each with specific features and responsibilities.

The system supports user registration, book listings, orders, inventory management, and admin-level monitoring. With secure authentication, a responsive UI, and RESTful API integration, BookNest offers a smooth and reliable experience. The project is scalable and can be enhanced with features like payment gateways and real-time updates in the future.