Full Stack Development with MERN

1.Introduction :

1.1 Project Title:

BookNest: Where Stories Nestle

1.2 Team Members:

- Leelaprasanna Putha
- Ekshitha Chinnaramappagari
- Akhilandeswari Vajrala
- Gowthami Kummara

1.3 Project Overview:

BookNestle is a comprehensive online book store application built using the MERN (MongoDB, Express.js, React, Node.js) stack. The platform serves as a digital marketplace where book enthusiasts can explore, discover, and purchase books seamlessly. It caters to three distinct user roles - regular users, sellers, and administrators - each with tailored functionalities.

1.4 Objectives:

The main objectives of this project are:

User-Friendly Interface: Create an intuitive and responsive user interface for seamless book browsing and purchasing across all devices.

Secure Authentication: Implement robust authentication and authorization using JWT tokens with role-based access control.

Role-Based Access: Develop separate dashboards and functionalities for three user types:

Users: Browse, purchase, and manage orders

Sellers: Add, edit, and manage book listings

Admins: Oversee platform, manage users and sellers

1. **Efficient Database Design**: Design optimized MongoDB schemas for books, users, and orders ensuring fast query performance.
2. **Complete E-commerce Flow**: Implement shopping cart functionality, order placement, and order tracking features.
3. **Search and Filter**: Enable users to search and filter books by title, author, genre, and price range.
4. **Wishlist Management**: Allow users to save favorite books for future purchase.
5. **Order History**: Provide users with complete history of past and current orders with status tracking.

2.PROJECT OVERVIEW:

2.1 Purpose of the Project:

The purpose of BookNestle is to create a modern, efficient, and user-friendly online platform that connects book lovers with a vast collection of books while providing sellers with tools to manage their inventory and administrators with oversight capabilities.

Core Purpose:

BookNestle aims to revolutionize the online book buying experience through digital transformation, transitioning traditional book shopping to a seamless digital experience accessible 24/7 from anywhere in the world. The platform democratizes book selling by empowering individual sellers and small bookstores to reach a wider audience without needing their own e-commerce infrastructure.

2.2 Key Features and Functionalities

User Features:

User registration and login provides secure sign-up and login with JWT authentication, protecting user accounts and personal information. Browse books functionality allows exploring books with pagination and lazy loading, ensuring fast performance even with thousands of books. Search books enables finding specific books quickly by title, author, or keywords. Filter books helps users narrow down choices based on genre, price range, and availability preferences.

Seller Features:

Seller dashboard provides an overview of sales, inventory, and orders, giving at-a-glance business insights. Add books allows listing new books with details such as title, author, price, and stock, making it easy to expand product catalogs. Edit books enables updating existing book information, keeping listings accurate. Delete books allows removing books no longer available, maintaining a clean inventory. View orders shows sellers the orders for their books, helping them track sales and fulfillment. Inventory management monitors stock levels to prevent overselling.

Admin Features:

Admin dashboard displays platform-wide statistics and analytics for monitoring overall platform health. User management allows viewing, editing, and managing all users to maintain user quality. Seller management enables approving, verifying, and monitoring sellers to ensure seller reliability. Book oversight lets administrators review and manage all book listings to maintain content quality. Order monitoring tracks all orders across the platform to ensure smooth operations. Platform statistics show users count, books count, orders count, and revenue for data-driven decision making.

Technical Features:

JWT authentication provides secure token-based authentication for all users. Password encryption using bcryptjs ensures secure password storage. RESTful API design creates well-structured API endpoints for all functionalities. Responsive design ensures the platform works on all devices from mobile to desktop. Comprehensive error handling provides appropriate user feedback for all operations. Loading states show visual indicators during asynchronous operations. Form validation is implemented on both client and server sides.

3 . ARCHITECTURE:

3.1 SYSTEM ARCHITECTURE OVERVIEW

BookNestle follows a modern three-tier client-server architecture based on the MERN stack. The architecture separates the presentation layer, application layer, and data layer into distinct components that communicate through RESTful APIs.

The client layer consists of the React frontend running in the user's browser. This layer handles all user interface rendering and user interactions. When users perform actions like clicking buttons or submitting forms, the frontend makes HTTP requests to the backend API.

3.2 FRONTEND ARCHITECTURE (REACT)

The frontend of BookNestle is built using React, a powerful JavaScript library for building user interfaces. The architecture follows component-based design principles, ensuring reusability, maintainability, and scalability.

3.2.1 Frontend Technology Stack

The frontend uses React version 18.2.0 as the core library for building user interfaces. React Router DOM version 6.15.0 handles client-side navigation, allowing users to move between pages without full page reloads. Axios version 1.5.0 serves as the HTTP client for making API calls to the backend. Bootstrap version 5.3.1 provides the CSS framework for responsive styling, while React-Bootstrap version 2.8.0 offers pre-built Bootstrap components optimized for React. React Icons version 4.11.0 provides a comprehensive icon library, and React

3.2.2 Frontend Folder Structure

The frontend code is organized into a logical folder structure that promotes separation of concerns and code reusability. The public folder contains static files, primarily the index.html file that serves as the entry point for the React application. The src folder contains all source code for the application.

The utils directory contains utility functions and constants used throughout the application. The App.js file serves as the root component that sets up routing and providers. The index.js file is the entry point that renders the App component into the DOM.

3.2.3 Component Architecture

The frontend follows a hierarchical component structure that promotes reusability and maintainability.

The App component serves as the root container that wraps the entire application. It imports BrowserRouter from React Router DOM to enable client-side routing. It wraps the application with AuthProvider and CartProvider to make authentication and cart state available to all child components. It defines all routes using the Routes and Route components, mapping URL paths to page components. It renders the Navbar component consistently across all pages and the Footer component at the bottom of every page.

3.2.4 State Management

The application uses React Context API for global state management, avoiding prop drilling and making state accessible to any component that needs it.

3.2.5 Data Flow in Frontend

When a user interacts with the frontend, data flows through a predictable cycle. User interactions such as clicking buttons or submitting forms trigger event handlers defined in components. These event handlers call functions from the service layer, which make HTTP requests to the backend API. The service layer returns promises that resolve with response data or reject with errors.

3.3 BACKEND ARCHITECTURE (NODE.JS AND EXPRESS.JS)

The backend of BookNestle is built using Node.js with Express.js framework, following the MVC pattern adapted for RESTful API development.

3.3.1 Backend Technology Stack

The backend runs on Node.js version 18 or higher, providing a fast, event-driven JavaScript runtime. Express.js version 4.18.2 serves as the web application framework, providing robust routing and middleware capabilities. Mongoose version 7.5.0 acts as the Object Data Modeling library for MongoDB, providing schema validation and query building. JSON Web Tokens version 9.0.2 handle secure authentication by

3.3.2 Backend Folder Structure

The backend code is organized into a modular structure that separates concerns and promotes maintainability.

The config folder contains configuration files, with db.js handling the MongoDB connection setup. The models folder defines Mongoose schemas for each collection, including user.js, book.js, and order.js, each exporting a corresponding Mongoose model.

3.3.3 Server Initialization Process

When the server starts, it follows a systematic initialization process. First, it loads environment variables from the .env file using dotenv, making configuration values available throughout the application. It then establishes a connection to MongoDB by calling the connectDB function from the config module, which uses Mongoose to connect to the database specified in the MONGODB_URI environment variable.

Finally, the server starts listening on the port specified in the environment variables, defaulting to port 5000. A message is logged to confirm successful startup.

3.3.4 Request-Response Cycle

When a client makes a request to the backend, it follows a well-defined path through the application. The request first passes through configured middleware functions. CORS middleware adds appropriate headers to the response. Express.json parses any JSON request body and attaches it to req.body.

3.3.5 Middleware Architecture

Middleware functions in Express have access to the request and response objects and can modify them or end the request-response cycle. BookNestle uses several custom middleware functions.

The CORS middleware configures Cross-Origin Resource Sharing by setting appropriate headers. It allows requests from the frontend origin while blocking unauthorized origins, enhancing security.

3.4 DATABASE ARCHITECTURE (MONGODB)

BookNestle uses MongoDB, a NoSQL document database, for data persistence. MongoDB's flexible schema design is ideal for e-commerce applications where data structures may evolve over time.

3.4.1 Database Technology

MongoDB is a document-oriented database that stores data in flexible, JSON-like documents called BSON. Unlike traditional relational databases that require predefined schemas and tables, MongoDB allows documents in the same collection to have different fields, providing flexibility as application requirements change.

3.4.2 Database Schema Design

The database consists of three main collections: users, books, and orders. Each collection stores documents with flexible schemas while maintaining consistency through Mongoose models.

Users Collection:

The users collection stores all user information including authentication data and personal details. Each user document contains fields for name, email, password, role, and timestamps. The email field is indexed and must be unique to prevent duplicate registrations. The password field is hashed using bcrypt before storage and is excluded from query results by default to enhance security.

Books Collection:

The books collection stores all book listings on the platform. Each book document contains fields for title, author, genre, description, price, stock quantity, and image URL. The title and author fields are required and indexed for efficient search operations.

Orders Collection:

The orders collection stores all customer orders placed through the platform. Each order document references the user who placed the order, creating a relationship between orders and users.

The orderItems field contains an array of embedded documents, each representing a book in the order. Each item document includes a reference to the book, the title at the time of order, the price at the time of order, and the quantity purchased. Storing the title and price at order time ensures order history remains accurate even if book details change later.

3.4.3 Database Relationships

MongoDB handles relationships differently than traditional relational databases. BookNestle implements relationships using references and embedded documents based on the nature of the relationship.

The user to order relationship is one-to-many, implemented by storing the user's ObjectId in each order document. When a user places multiple orders, each order document contains a reference to the same user. This allows the system to efficiently retrieve all orders for a specific user by querying orders where the user field matches the user's ID.

3.4.4 Data Access Patterns

Different parts of the application require different data access patterns, and the database design optimizes for these patterns.

For user authentication, when a user logs in, the system queries the users collection by email. The query selects the password field for comparison with the provided password. After successful authentication, the user data without the password is returned along with a JWT token.

3.4.5 Indexing Strategy

Indexes are crucial for database performance, especially as collections grow. BookNestle implements strategic indexes on frequently queried fields.

A unique index on the email field in the users collection prevents duplicate account registrations and ensures fast login queries. When a user attempts to register, MongoDB quickly checks if the email already exists using this index.

4. SETUP INSTRUCTIONS

4.1 PREREQUISITES

Before installing and running the BookNestle application, ensure your system meets the following requirements and has the necessary software installed.

4.1.1 Required Software

Node.js and npm
Node.js is a JavaScript runtime that allows running JavaScript on the server side. npm is the Node Package Manager that comes bundled with Node.js and is used to install project dependencies. The application requires Node.js version 18.x or higher. Node.js can be downloaded from the official website at https://nodejs.org. During installation, ensure that the option to add Node.js to the system PATH is selected, which allows running node and npm commands from any terminal.

MongoDB

MongoDB is the NoSQL database used to store all application data including user information, book listings, and order records. The application requires MongoDB version 7.0 or higher. MongoDB Community Edition can be downloaded from https://www.mongodb.com/try/download/community. During installation on Windows, ensure that the option to install MongoDB as a Service is selected, which automatically starts MongoDB when the system boots. For development, MongoDB can also be run manually using the mongod command.

Git

Git is a version control system used to clone the project repository. Git can be downloaded from https://git-scm.com/downloads. During installation, accept the default options, which include adding Git to the system PATH. Git allows cloning the project repository and managing code versions.

Code Editor (Optional but Recommended)

While any text editor can be used for viewing and modifying code, Visual Studio Code is highly recommended. VS Code provides excellent support for JavaScript, React, and Node.js development with features like syntax highlighting, IntelliSense, and integrated terminal. VS Code can be downloaded from https://code.visualstudio.com.

4.1.2 System Requirements

Minimum Hardware Requirements

The development environment requires a processor of Intel Core i3 or equivalent, with at least 4 GB of RAM. Storage space of at least 10 GB free is needed for the application code, dependencies, and database files. A stable internet connection is required for downloading dependencies and running the application.

Recommended Hardware Requirements

For optimal development experience, an Intel Core i5 or better processor is recommended, along with 8 GB or more of RAM. Solid State Drive storage improves build times and database performance. A broadband internet connection ensures fast downloading of packages.

4.1.3 Knowledge Prerequisites

Basic familiarity with the command line or terminal is helpful for running the installation commands. Understanding of JavaScript and web development concepts is beneficial for modifying and extending the application. However, the setup instructions are designed to be followed by anyone, regardless of technical background.

4.2 INSTALLATION STEPS

Follow these step-by-step instructions to set up the BookNestle application on your local machine for development and testing.

4.2.1 Step 1: Clone the Repository

Open a terminal or command prompt and navigate to the directory where you want to install the project. Use the git clone command to download the project repository.

4.2.2 Step 2: Backend Setup

Navigate to the server directory and install all backend dependencies.The npm install command reads the package.json file in the server directory and downloads all required dependencies into a node_modules folder. This process may take a few minutes depending on your internet connection speed. The dependencies installed include express for the web server, mongoose for database connectivity, jsonwebtoken for authentication, bcryptjs for password hashing, cors for cross-origin resource sharing, and dotenv for environment variable management.

4.2.3 Step 3: Frontend Setup

Open a new terminal window and navigate to the client directory from the project root.The npm install command reads the package.json file in the client directory and installs all frontend dependencies. These include react and react-dom for the core React library, react-router-dom for client-side routing, axios for HTTP requests, bootstrap and react-bootstrap for styling, react-icons for icons, and react-toastify for notifications. This installation process may also take a few minutes.

The frontend does not require a separate .env file as it uses Vite's built-in environment variable handling. API endpoints are configured in the vite.config.js file to proxy requests to the backend during development.

4.2.4 Step 4: MongoDB Setup

MongoDB must be running before the backend server can connect to the database. The setup process depends on your operating system and how MongoDB was installed.

4.2.5 Step 5: Database Initialization (Optional)

The application will automatically create the required collections when data is first inserted. However, you may want to add some sample books to test the application. The backend includes a seed route that populates the database with sample data if it is empty.

4.2.6 Step 6: Verify Installation

Before running the application, verify that all components are correctly installed. Check that Node.js and npm are available:

5.FOLDER STRUCTURE

5.1 CLIENT FOLDER STRUCTURE (REACT FRONTEND)

The client folder contains the entire React frontend application. It is structured to promote reusability, maintainability, and scalability of the user interface code.

5.2.1 Root Level Files

package.json
The package.json file is the heart of the frontend application, listing all dependencies, scripts, and metadata. It includes dependencies for React, React Router, Axios, Bootstrap, and other frontend libraries. The scripts section defines commands for development, building, and previewing the application.

vite.config.js
This file configures Vite, the build tool used for the frontend. It defines the development server port, proxy settings for API requests during development, and plugins used by Vite. The proxy configuration forwards API requests to the backend server, avoiding CORS issues during development.

index.html
Located in the public folder, index.html is the main HTML file that serves as the entry point for the React application. It contains a div element with id root where the React application is mounted, and includes links to fonts and other external resources.

5.2.2 Source Code Structure (src/)

components/ Directory

The components directory houses reusable UI components that are used across multiple pages. Each component is typically defined in its own file with a descriptive name.

.

pages/ Directory

The pages directory contains components that correspond to specific routes in the application. Each page component combines multiple smaller components to create complete, functional pages.

Home.js is the landing page component that displays when users visit the root URL. It shows a welcome banner, featured books or bestsellers, and navigation cards linking to other sections of the application. The home page provides an overview of the platform and encourages users to explore further.

services/ Directory

The services directory abstracts API communication, providing clean interfaces for frontend components to interact with the backend.

api.js configures Axios with the base URL for API requests and sets up interceptors to automatically attach authentication tokens to outgoing requests. It handles common error scenarios and provides a consistent interface for making HTTP requests.

context/ Directory

The context directory implements React Context for global state management, avoiding prop drilling and making state accessible to any component.

AuthContext.js manages authentication state throughout the application. It provides the current user object, authentication status, and functions for login, registration, and logout. The context persists user data in localStorage, maintaining login state across browser sessions.

utils/ Directory

The utils directory contains helper functions and constants used throughout the application.

App.js

App.js is the root component of the React application. It imports BrowserRouter from React Router DOM to enable client-side routing. It wraps the application with AuthProvider and CartProvider to make authentication and cart state available throughout the component tree. It defines all routes using the Routes and Route components, mapping URL paths to page components. The App component also renders the Navbar component consistently across all pages.

index.js

index.js is the entry point of the React application. It imports React, ReactDOM, the App component, and global CSS styles. It uses ReactDOM.createRoot to render the App component into the DOM element with id root. The file may also include StrictMode for highlighting potential problems in development.

index.css

index.css contains global styles that apply to the entire application. It includes CSS resets, font definitions, and utility classes used across multiple components. Bootstrap is also imported here or in the main component.

5.2 SERVER FOLDER STRUCTURE (NODE.JS BACKEND)

The server folder contains the complete Node.js backend application built with Express.js and MongoDB. The structure follows the MVC pattern adapted for RESTful API development.

5.3.1 Root Level Files

package.json

The package.json file in the server directory lists all backend dependencies including express, mongoose, jsonwebtoken, bcryptjs, cors, and dotenv. It defines scripts for starting the server in development and production modes. The main field specifies the entry point file, typically index.js.

.env

The .env file stores environment variables that configure the application. It contains sensitive information such as database connection strings and JWT secrets, and should never be committed to version control. The file includes variables for PORT, MONGODB_URI, JWT_SECRET, and NODE_ENV.

index.js

index.js is the main entry point for the backend server. It imports required modules, configures Express middleware, connects to the database, registers routes, and starts the server listening on the configured port.

5.3.2 config/ Directory

The config directory contains configuration files that set up various aspects of the application.

db.js

This file handles MongoDB connection configuration. It exports an asynchronous function that uses mongoose.connect to establish a connection to the database using the URI from environment variables. The function includes error handling and logs connection status. It is imported and called in index.js during server startup.

5.3.3 models/ Directory

The models directory defines Mongoose schemas and models for MongoDB collections. Each model corresponds to a collection in the database and defines the structure of documents within that collection.

user.js
The user model defines the schema for user accounts. It includes fields for name, email, password, role, profilePicture, phoneNumber, addresses, wishlist, and timestamps. The schema includes validation rules ensuring required fields are present and email addresses are unique. It also defines middleware to hash passwords before saving and methods to compare passwords during authentication.

book.js
The book model defines the schema for book listings. It includes fields for title, author, genre, description, price, stock, imageUrl, seller reference, and timestamps. The schema includes validation for required fields and data types. Text indexes are defined on title and author to enable efficient search queries.

order.js
The order model defines the schema for customer orders. It includes fields for user reference, order items array, shipping address, total price, status, and timestamps. The order items array contains embedded documents with book reference, title, price, and quantity. The status field is restricted to specific enum values.

5.3.4 routes/ Directory

The routes directory defines API endpoints and maps them to controller functions. Each route file handles a specific resource or functional area.

authRoutes.js
This file defines authentication-related routes. It includes POST /register for user registration, POST /login for user login, and GET /profile for retrieving the authenticated user's profile. Each route is associated with a corresponding controller function.

bookRoutes.js
The book routes handle all book-related operations. It defines GET / for retrieving all books with optional filtering, GET /:id for getting a single book, POST / for creating new books, PUT /:id for updating books, and DELETE /:id for removing books. Protected routes use the authentication middleware to verify user identity.

orderRoutes.js
Order routes manage customer orders. It includes POST / for creating new orders, GET /myorders for retrieving the current user's orders, and GET /:id for getting specific order details. All order routes are protected and require authentication.

adminRoutes.js
Admin routes provide administrative functionality. It includes GET /stats for dashboard statistics, GET /users for listing all users, GET /sellers for listing all sellers, PUT /users/:id for updating user roles, and DELETE /users/:id for removing users. These routes use both authentication and authorization middleware to ensure only administrators can access them.

5.3.5 controllers/ Directory

The controllers directory contains the business logic for handling API requests. Each controller file exports functions that correspond to route handlers.

authController.js
The authentication controller handles user registration, login, and profile retrieval. It contains functions that validate input, interact with the user model, generate JWT tokens, and send appropriate responses. Error handling ensures proper status codes and messages are returned.

bookController.js
The book controller manages all book-related operations. It includes functions for fetching books with filtering and pagination, retrieving single books, creating new books, updating existing books, and deleting books. Controller functions validate input, check permissions, and interact with the book model.

orderController.js
The order controller handles order processing. It includes functions for creating new orders, retrieving user orders, and fetching specific order details. Order creation includes validation of items, calculation of totals, and updating book stock.

adminController.js
The admin controller provides administrative functionality. It includes functions for gathering dashboard statistics, managing users and sellers, and performing administrative operations. These functions typically aggregate data from multiple collections and return comprehensive results.

5.3.6 middleware/ Directory

The middleware directory contains custom Express middleware functions that process requests before they reach route handlers.

authMiddleware.js
This file exports authentication and authorization middleware functions. The protect function verifies JWT tokens from the Authorization header,

attaches the decoded user to the request object, and rejects invalid or missing tokens. The authorize function checks if the authenticated user has the required role, allowing or denying access to protected routes.

6 . RUNNING THE APPLICATION

6.1 PREREQUISITES CHECK

Before running the application, ensure Node.js and MongoDB are installed. Verify by running:

```
node --version
```

```
npm --version
```

Both commands should display version numbers without errors.

6.2 STARTING MONGODB DATABASE

Open a terminal and start MongoDB:

Windows:

bash

```
"C:\Program Files\MongoDB\Server\8.2\bin\mongod.exe" --dbpath="C:\data\db"
```

6.3 STARTING THE BACKEND SERVER

Open a new terminal and navigate to the server directory:

```
cd booknestle/server
```

Start the backend server in development mode

```
npm run dev
```

You should see:

text

```
Server running on port 5000
MongoDB Connected: localhost:27017
```

Keep this terminal open. Verify the backend is working by visiting http://localhost:5000/api/test in your browser. You should see {"message":"Backend is working!"}

6.4 STARTING THE FRONTEND SERVER

Open a new terminal and navigate to the client directory:

```
cd booknestle/client
```

Start the frontend development server:

```
npm run dev
```

You should see:

text

```
VITE v4.4.5  ready in 200 ms
➜  Local:   http://localhost:5173/
```

Keep this terminal open.

6.5 ACCESSING THE APPLICATION

Open your web browser and navigate to:

text

```
http://localhost:5173
```

The BookNestle home page should load, confirming that all components are running correctly.

7 . API DOCUMENTATION

7.1 BASE URL

All API endpoints are prefixed with /api. In development, the base URL is http://localhost:5000/api.

7.2 AUTHENTICATION ENDPOINTS

POST /api/auth/register
This endpoint allows new users to create an account by providing their name, email address, and password. The system validates that the email is not already registered, securely hashes the password, and creates a new user record. Upon success, it returns the user's information along with a JWT token for authenticated sessions.

POST /api/auth/login
This endpoint authenticates existing users by verifying their email and password credentials. If valid, it returns the user's information along with a JWT token that must be included in subsequent requests to protected endpoints. The token expires after 30 days.

GET /api/auth/profile
This protected endpoint returns the profile information of the currently authenticated user. It requires a valid JWT token in the Authorization header. The response includes user ID, name, email, role, and account creation date.

7.3 BOOK ENDPOINTS

GET /api/books
This endpoint retrieves a paginated list of all books. It supports optional query parameters for filtering by genre, searching by title or author, and

pagination controls. The response includes the book list, current page, total pages, and total count.

GET /api/books/:id
This endpoint retrieves detailed information about a specific book using its unique identifier. The response includes title, author, genre, description, price, stock quantity, image URL, and seller information.

POST /api/books
This protected endpoint allows sellers and administrators to add new books. It requires a valid JWT token with seller or admin privileges. The request includes all book details such as title, author, genre, description, price, stock, and image URL.

PUT /api/books/:id
This protected endpoint allows sellers and administrators to update existing book information. Sellers can only update their own books, while administrators can update any book.

DELETE /api/books/:id
This protected endpoint allows sellers and administrators to remove books from the platform. Sellers can only delete their own books, while administrators can delete any book.

7.4 ORDER ENDPOINTS

POST /api/orders
This protected endpoint allows authenticated users to place new orders. The request includes ordered items with book IDs and quantities, along with shipping address details. The system validates stock availability, calculates the total price, creates an order record, and updates inventory quantities.

GET /api/orders/myorders
This protected endpoint retrieves all orders placed by the currently authenticated user. The response includes a list of orders with items, total price, status, and creation dates, sorted with the most recent first.

GET /api/orders/:id
This protected endpoint retrieves detailed information about a specific order. Users can only access their own orders unless they have administrator privileges.

7.5 ADMIN ENDPOINTS

All admin endpoints require a valid JWT token with administrator role privileges.

GET /api/admin/stats
This endpoint returns platform-wide statistics including total users, total sellers, total books, total orders, and total revenue generated.

GET /api/admin/users
This endpoint retrieves a list of all registered users with their details including ID, name, email, role, and account creation date.

GET /api/admin/sellers
This endpoint retrieves a list of all users with seller role, including their seller-specific information.

PUT /api/admin/users/:id
This endpoint allows administrators to change a user's role to user, seller, or admin.

DELETE /api/admin/users/:id
This endpoint allows administrators to permanently remove users from the platform.

8 . AUTHENTICATION AND AUTHORIZATION

8.1 OVERVIEW

BookNestle implements a secure, token-based authentication system using JSON Web Tokens (JWT). The system ensures that only authenticated users can access protected resources and that users can only perform actions appropriate for their assigned role. Authentication verifies the identity of users, while authorization determines what actions they are permitted to perform.

8.2 AUTHENTICATION FLOW

User Registration

When a new user registers, the system accepts their name, email address, and password. The email address is checked against the database to ensure it is not already in use. The password is never stored in plain text. Instead, it is hashed using the bcrypt algorithm with a salt factor of 10, which adds a layer of security even if the database is compromised. A new user record is created with the default role of "user". A JWT token is generated and returned to the client along with the user information, allowing the user to be automatically logged in after registration.

User Login

When an existing user attempts to log in, the system retrieves their record from the database using the provided email address. The stored hashed password is compared with the hash of the provided password using bcrypt's comparison function. If the passwords match, authentication is successful. The system generates a new JWT token containing the user's ID and role, signed with a secret key stored in the server's environment variables. This token is returned to the client along with the user's profile information.

Token Generation
JWTs are generated using the jsonwebtoken library. Each token contains a payload with the user's ID and is signed with the JWT_SECRET from the environment variables. Tokens are configured to expire after 30 days, after which the user must log in again to obtain a new token. This expiration period balances security with user convenience.

8.3 TOKEN STORAGE AND TRANSMISSION

Client-Side Storage
Upon successful authentication, the client receives the JWT token and stores it in the browser's localStorage. This approach allows the token to persist across browser sessions, so users remain logged in even after closing and reopening the browser. The token is never stored in cookies to avoid cross-site request forgery vulnerabilities.

Token Transmission
For every request to protected API endpoints, the client includes the token in the Authorization header using the Bearer scheme. The header format is "Authorization: Bearer <token>". This method ensures that tokens are not exposed in URLs or request bodies, reducing the risk of accidental exposure in logs or browser history.

8.4 TOKEN VERIFICATION

When the backend receives a request to a protected route, it first extracts the token from the Authorization header. The token is verified using the same JWT_SECRET that was used to sign it. The verification process checks the token's signature to ensure it has not been tampered with and validates that the token has not expired. If verification succeeds, the decoded payload containing the user's ID is attached to the request object for use in subsequent middleware and route handlers. If verification fails, the server responds with a 401 Unauthorized status.

8.5 AUTHORIZATION SYSTEM

Role-Based Access Control

BookNestle implements role-based access control with three distinct user roles: user, seller, and administrator. Each role has a specific set of permissions that determine which actions the user can perform. The user's role is stored in the database and included in the JWT payload during authentication.

User Role Permissions

Regular users can browse books, search the catalog, add items to their shopping cart, place orders, view their own order history, and manage their personal wishlist. They can also update their own profile information. Users cannot access seller or administrative functions.

Seller Role Permissions

Sellers have all the permissions of regular users. Additionally, they can access a seller dashboard, add new books to the platform, edit their own book listings, delete their own books, and view orders placed for their books. Sellers cannot modify books listed by other sellers or access administrative functions.

Administrator Role Permissions

Administrators have full access to all platform features. They can view and manage all users, change user roles, delete user accounts, view and manage all sellers, access all book listings, view all orders across the platform, and access platform-wide statistics through the admin dashboard. Administrators can perform any operation on any resource.

8.6 AUTHORIZATION MIDDLEWARE

The backend implements authorization middleware that runs after authentication middleware. This middleware checks the role of the authenticated user against the required role for the requested route. If the user has the required role, the request proceeds to the route handler. If not, the server responds with a 403 Forbidden status. This ensures that even if a user obtains a valid token, they cannot access resources or perform actions beyond their authorized scope.

8.7 SECURITY MEASURES

Password Security

All passwords are hashed using bcrypt before storage. Bcrypt is a deliberately slow hash function that makes brute-force attacks impractical. The salt factor of 10 adds computational cost to each hash attempt, further slowing down attackers.

Token Security

JWT tokens are signed with a secret key stored only on the server. This secret should be a long, random string and is never exposed to clients. In production, it is stored as an environment variable and never committed to version control.

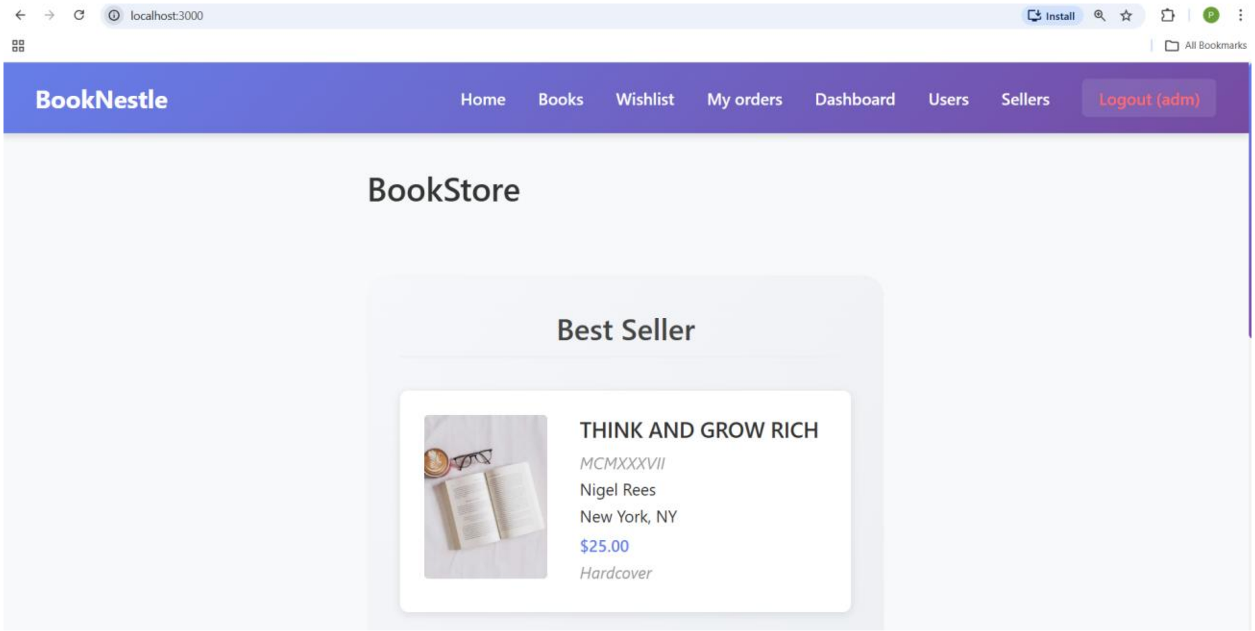
HTTPS

In production, all communication between client and server should occur over HTTPS to prevent token interception during transmission.

Token Expiration

JWTs expire after 30 days, limiting the window of opportunity if a token is compromised. Users must re-authenticate after expiration to obtain a new token.

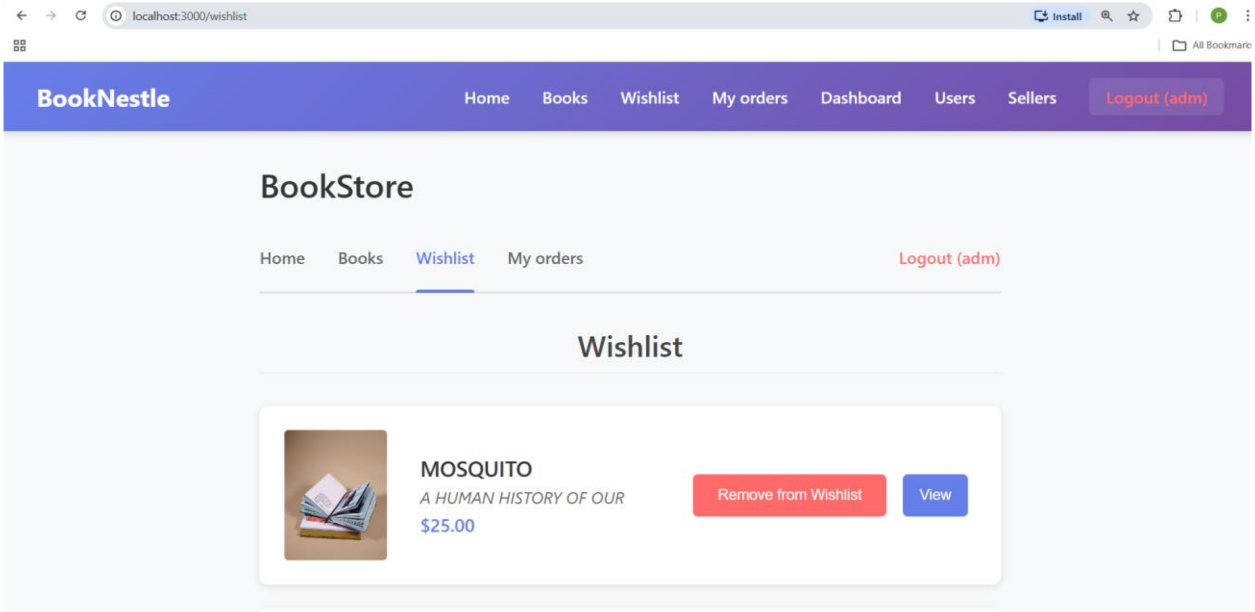
9. User Interface



BookStore

Best Seller

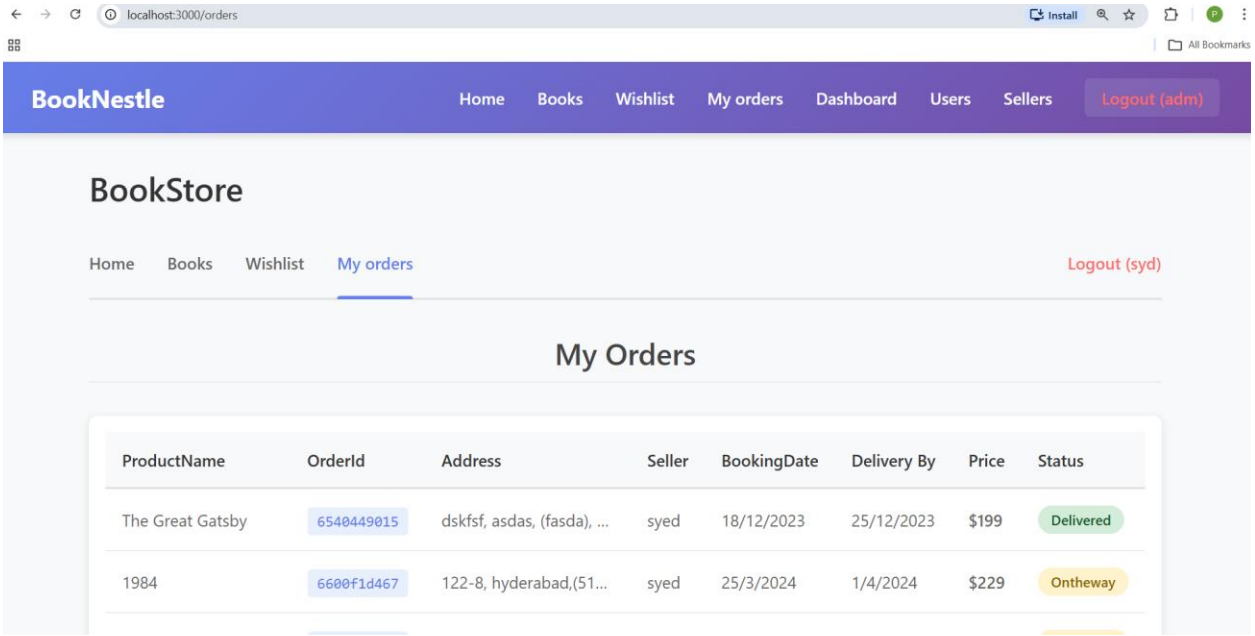
THINK AND GROW RICH
MCMXXXVII
Nigel Rees
New York, NY
$25.00
Hardcover



BookStore

Wishlist

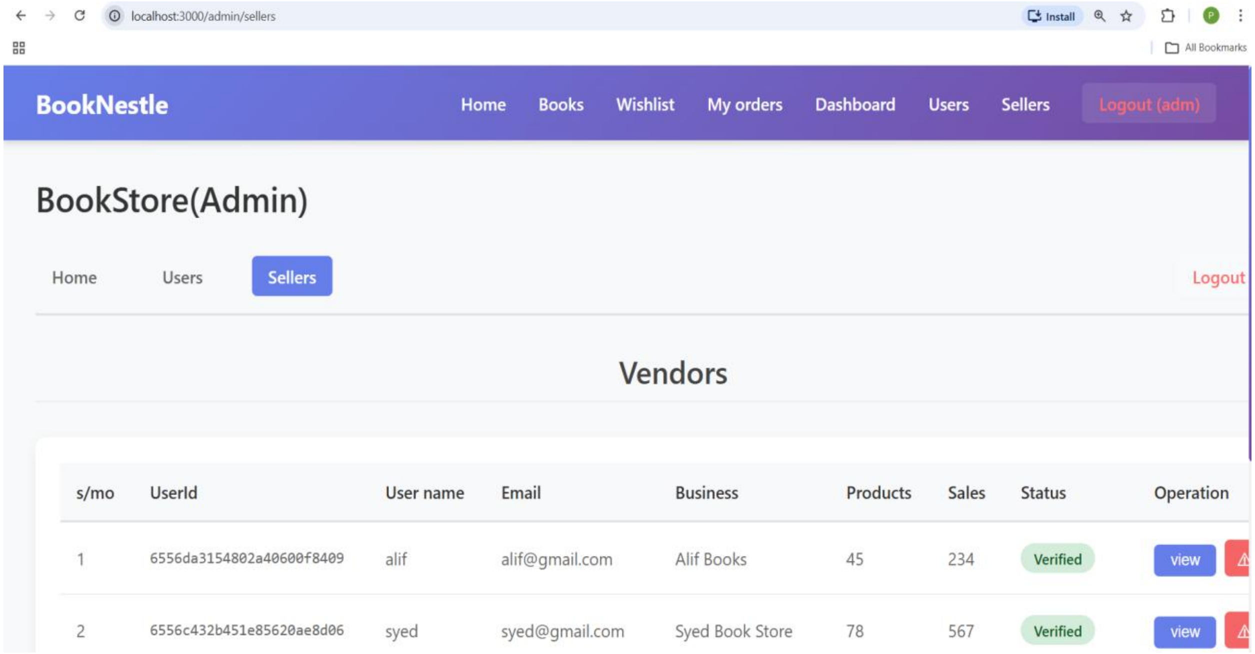
MOSQUITO
A HUMAN HISTORY OF OUR
$25.00



BookStore

My Orders

ProductName	OrderId	Address	Seller	BookingDate	Delivery By	Price	Status
The Great Gatsby	6540449015	dskfsf, asdas, (fasda), ...	syed	18/12/2023	25/12/2023	$199	Delivered
1984	6600f1d467	122-8, hyderabad,(51...	syed	25/3/2024	1/4/2024	$229	Ontheway



BookStore(Admin)

Vendors

s/mo	UserId	User name	Email	Business	Products	Sales	Status	Operation
1	6556da3154802a40600f8409	alif	alif@gmail.com	Alif Books	45	234	Verified	view
2	6556c432b451e85620ae8d06	syed	syed@gmail.com	Syed Book Store	78	567	Verified	view

10 .TESTING:

10.1 TESTING STRATEGY

The testing strategy for BookNestle encompasses multiple levels of testing to ensure the reliability, functionality, and user experience of the application. Unit testing is performed on individual components and functions to verify that each part works correctly in isolation. Integration testing focuses on the interactions between different modules, particularly the communication between the frontend and backend through API endpoints. End-to-end testing validates complete user flows such as registration, book browsing, adding items to cart, and order placement to ensure the entire system functions as expected. Manual testing is also conducted to evaluate the user interface, responsiveness, and overall user experience across different devices and browsers.

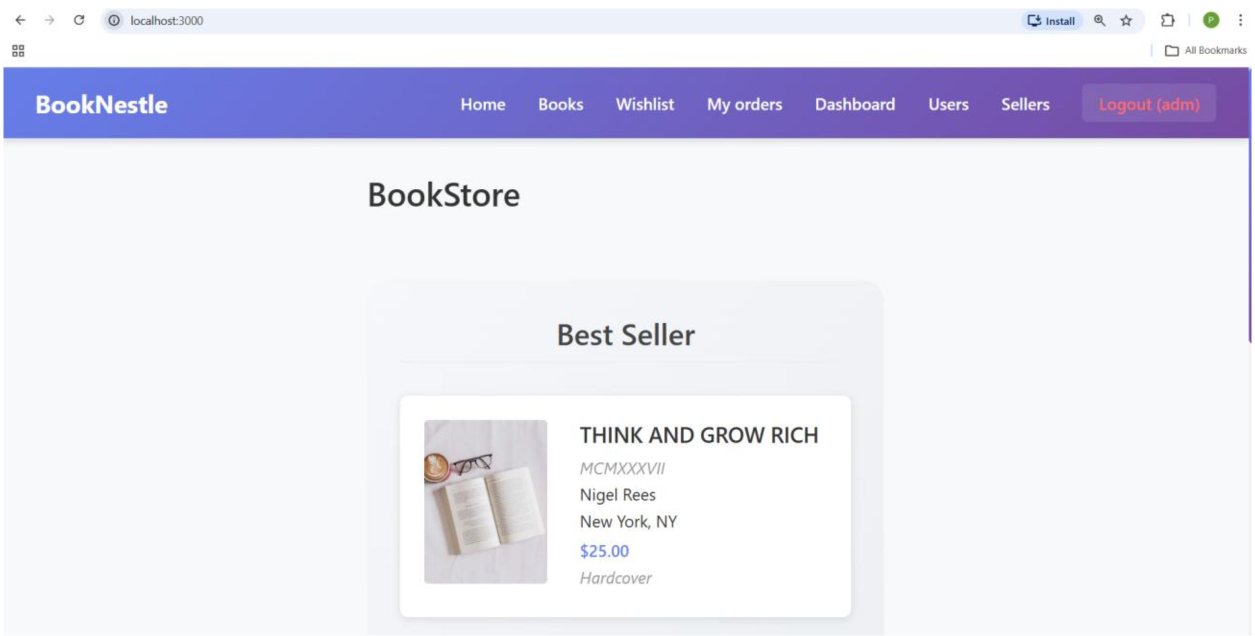
10.2 TESTING TOOLS

Several tools are employed to facilitate comprehensive testing of the application. Postman is used extensively for API testing, allowing the team to send requests to backend endpoints, verify responses, and automate test collections. The browser's developer tools are utilized for frontend debugging, network request monitoring, and responsive design testing across different screen sizes. For unit and integration testing, Jest serves as the primary testing framework along with React Testing Library for component testing. These tools enable the team to write and execute tests efficiently, ensuring that code changes do not introduce regressions. MongoDB Compass is used to verify database operations by directly inspecting collections and documents during testing.

10.3 TEST COVERAGE AND RESULTS

The testing process covers all critical functionality of the application including user authentication, book management, order processing, and administrative features. API endpoints are tested for proper request handling, error responses, and authentication requirements. The frontend components are tested for correct rendering, user interaction handling, and proper state management. Cross-browser testing ensures consistent behavior across Chrome, Firefox, and Edge browsers. Mobile responsiveness is verified using browser developer tools and actual mobile devices. Through this comprehensive testing approach, the application achieves a high level of stability and reliability, with most critical bugs identified and resolved before deployment. Any known issues are documented for future improvement.

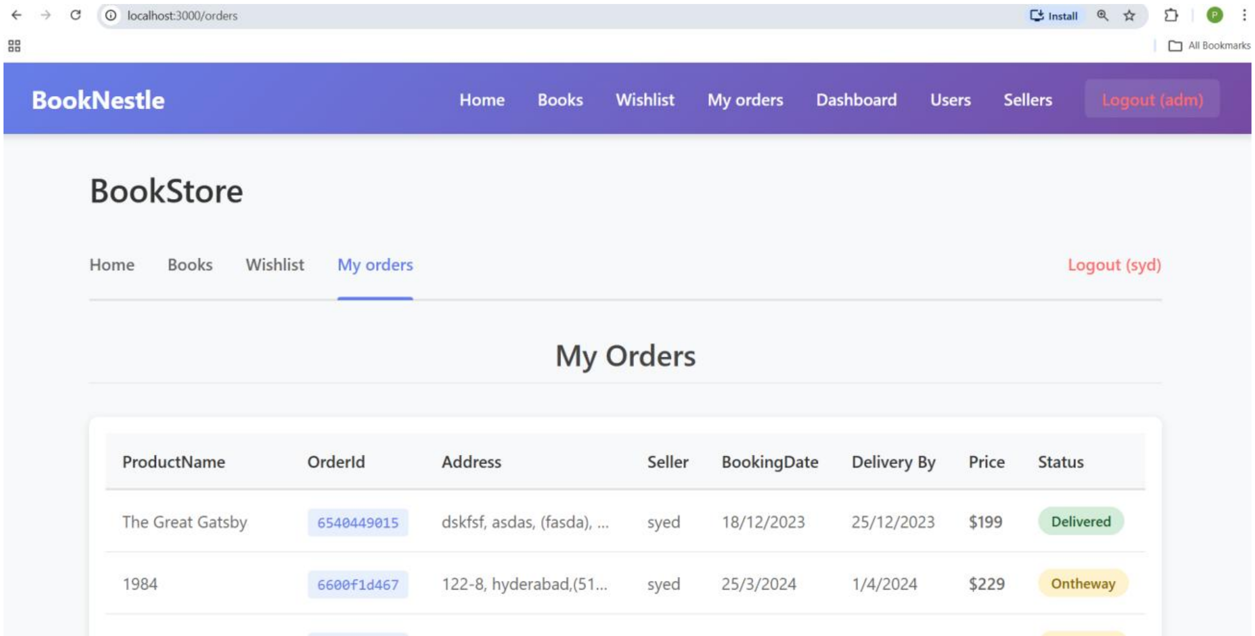
11. Screenshots or Demo



BookStore

Best Seller

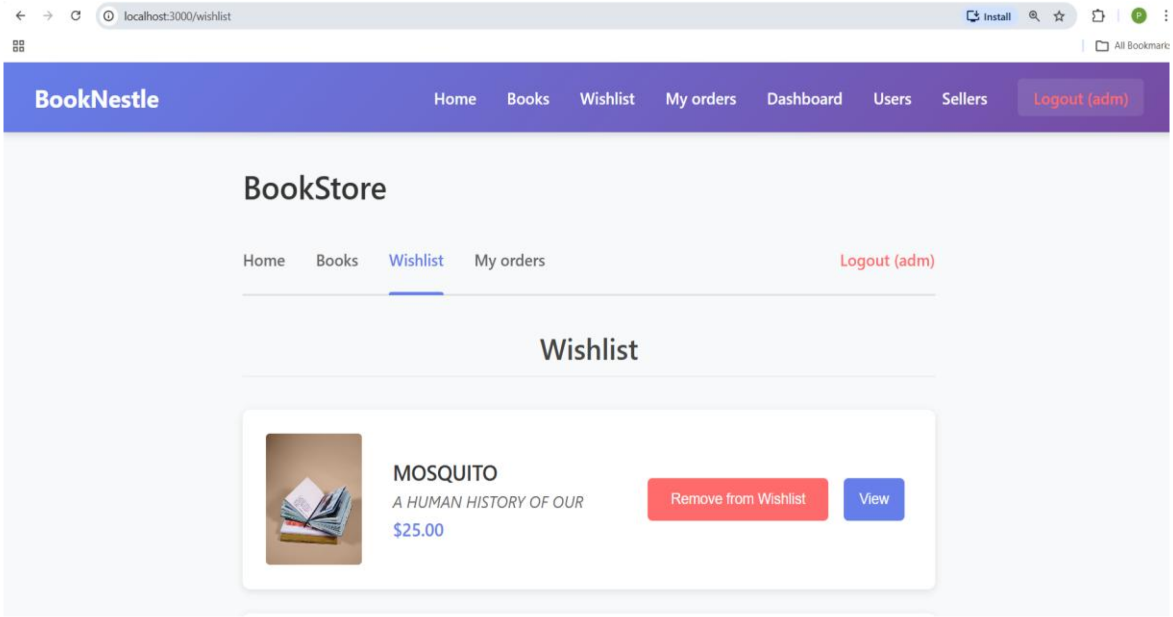
THINK AND GROW RICH
MCMXXXVII
Nigel Rees
New York, NY
$25.00
Hardcover



BookStore

My Orders

ProductName	OrderId	Address	Seller	BookingDate	Delivery By	Price	Status
The Great Gatsby	6540449015	dskfsf, asdas, (fasda), ...	syed	18/12/2023	25/12/2023	$199	Delivered
1984	6600f1d467	122-8, hyderabad,(51...	syed	25/3/2024	1/4/2024	$229	Ontheway



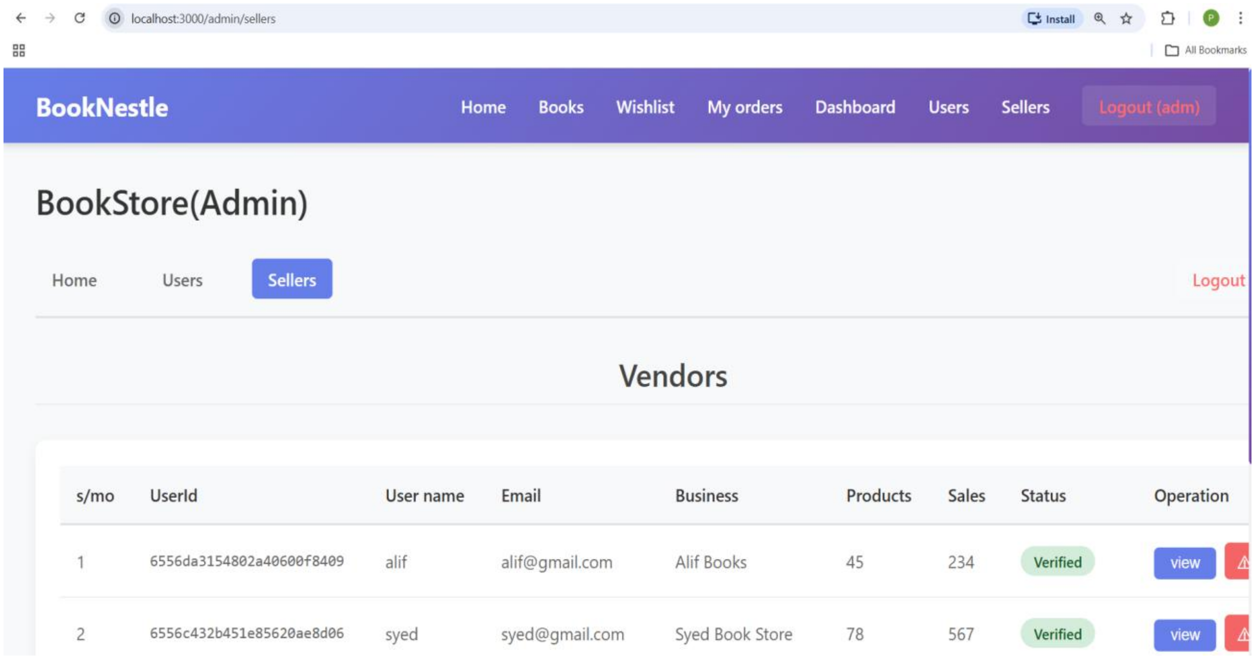
BookStore

Wishlist

MOSQUITO

A HUMAN HISTORY OF OUR

$25.00



BookStore(Admin)

Vendors

s/mo	UserId	User name	Email	Business	Products	Sales	Status	Operation
1	6556da3154802a40600f8409	alif	alif@gmail.com	Alif Books	45	234	Verified	view ⚠
2	6556c432b451e85620ae8d06	syed	syed@gmail.com	Syed Book Store	78	567	Verified	view ⚠

12 . KNOWN ISSUES

The BookNestle application has a few known issues that users and developers should be aware of. Cart persistence across browser sessions may occasionally fail to restore saved items due to localStorage limitations. Image upload for book covers currently only accepts URL links rather than direct file uploads, which may be inconvenient for some sellers. The search functionality may not return expected results when special characters are included in the query. On smaller mobile screens, the navigation menu may occasionally overlap with page content requiring a page refresh to correct. Session timeout handling is not yet implemented, so users remain logged in indefinitely until they manually log out. Payment processing is currently simulated and not integrated with a real payment gateway, so no actual financial transactions occur. These issues are documented for transparency and are planned to be addressed in future updates to enhance the overall user experience.

13. FUTURE ENHANCEMENTS

Several enhancements are planned for future versions of the BookNestle application to expand its functionality and improve user experience. The highest priority is integrating a real payment gateway such as Stripe or PayPal to enable actual financial transactions rather than simulated payments. Email notification functionality will be added to send order confirmations, shipping updates, and password reset links to users. A book review and rating system will allow customers to share their opinions and help others make informed purchasing decisions.

Advanced search filters including price range, publication date, and customer ratings will be implemented to help users find books more precisely. A recommendation engine based on user browsing history and purchase patterns will suggest relevant books to individual users. Mobile applications for iOS and Android platforms will be developed using React Native to provide native mobile experiences. Social media integration will enable users to share their favorite books and wishlists with friends. Multi-language support will make the platform accessible to a broader audience, and dark mode functionality will be added for user preference and reduced eye strain.