BOOK STORE APPLICATION USING MERN STACK

A PROJECT REPORT

SUBMITTED BY

PAAUL ANTONY. P – 310821104066

SANJAIKUMARAN. P – 310821104083

RAGHUL. R – 310821104073

C X Arul Mc Millan – 310821104013

IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

JEPPIAAR ENGINEERING COLLEGE

ANNA UNIVERSITY

CHENNAI – 600 025

NOVEMBER 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"BOOK STORE APPLICATION USING MERN STACK"** is the Bonafide work of **"SANJAIKUMARAN P (310821104083)"** who carried out the project work for Naan Mudhalvan during the academic year 2021-2025.

MENTOR **HEAD OF THE DEPARTMENT**

Date: _____ **Internal:**

External:

S.NO	Contents	Page No
1.	Introduction	**04**
2.	Project Overview	**04**
3.	Architecture	**05**
4.	Set Up Instructions	**09**
5.	Folder Structure	**13**
6.	Running the Application	**14**
7.	API Documentation.	**14**
8.	Authentication	**19**
9.	User Interface	**22**
10.	Testing	**26**
11.	Screenshots	**27**
12.	Known Issues	**30**
13.	Future Enhancements	**30**

1. INTRODUCTION

The **'Book Store'** project is a comprehensive web application developed using the **MERN stack.** MERN stands for MongoDB, Express.js, React.js, and Node.js, which together provide an efficient and modern framework for developing full-stack applications. This project aims to offer a user-friendly platform for browsing, purchasing, and managing books online.

The report details the design, implementation, and features of the Book Store application. It highlights the development process, from setting up the architecture to deploying the application. The project was collaboratively built by a team of developers, whose efforts ensured the creation of a scalable and responsive solution.

2. Purpose

The **Book Store** project is a web application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js). Its primary goal is to create a seamless online platform where users can browse, search, and purchase books. Designed for scalability and security, this application simplifies the book-shopping experience, making it accessible to a wide audience, from individual readers to libraries and bookstores.

Key Objectives:

- Provide a digital platform for book discovery and purchase.
- Ensure a user-friendly and secure experience for all users.
- Offer real-time updates on book availability and transaction status.

Features

1. **User Authentication**:
    - Secure login and registration using JSON Web Tokens (JWT).
    - Role-based access control for users and administrators.
2. **Book Catalog**:
    - Display books with details like title, author, price, genre, and availability.
    - Search and filter functionality for better user experience.
3. **Cart and Checkout**:

- Add selected books to a shopping cart.
- Proceed to a secure checkout process.

4. **Responsive Design**:
    - Optimized for both desktop and mobile devices.
5. **Order Management**:
    - Allow users to view past orders and track order status.

Target Users

- **General Readers**: Individuals looking for a convenient way to purchase books.
- **Libraries and Institutions**: Bulk ordering and catalog management.
- **Authors and Publishers**: Showcase and sell their books.

3. Architecture

The Book Store application is designed using the **MERN stack** to leverage its end-to-end JavaScript capabilities. This architecture is divided into three main layers: the **Frontend**, **Backend**, and **Database**, with clear communication flows between them. This modular approach ensures flexibility, scalability, and maintainability.

1. Frontend (Client)

The **frontend** is the user-facing layer, built using **React.js** to provide an interactive and dynamic interface for users. It follows a **Component-Based Architecture**, where the application is divided into small, reusable components.

Key Features and Responsibilities:

1. **Rendering Dynamic Content**:
    - React's virtual DOM ensures faster rendering of changes, improving the performance of the application.
    - Dynamic pages such as the home page, book catalog, and user profile are rendered in real-time based on API responses.
2. **Component Breakdown**:
    - **Header and Footer Components**: Used on all pages for consistent navigation and branding.
    - **BookCard Component**: Displays information about individual books.

- **Cart Component**: Shows selected books, allowing users to update quantities or remove items.
- **Authentication Components**: Includes login, registration, and profile management forms.
- **Search and Filter Components**: Provides a responsive interface for users to search by book title, author, genre, or price range.

3. **State Management**:
    - **Redux Toolkit**: Manages global states such as the user's session, shopping cart, and search filters, ensuring seamless state updates across components.
    - Example: When a user adds a book to the cart, the cart updates across the application without reloading the page.
4. **API Integration**:
    - Communicates with the backend via RESTful APIs using **Axios** or the Fetch API.
    - Handles loading states, error messages, and data updates in the UI.
5. **Styling and Responsiveness**:
    - Uses **CSS-in-JS** libraries (e.g., Styled Components) for component-specific styling.
    - Responsive layouts ensure compatibility with desktops, tablets, and mobile devices using media queries and frameworks like **Bootstrap** or **Tailwind CSS**.
6. **Routing**:
    - **React Router** is used for client-side routing, enabling Single-Page Application (SPA) behaviour.
    - Key routes:
        - /: Home page
        - /books: Book catalog
        - /cart: Shopping cart
        - /profile: User profile and order history

2. Backend (Server)

The **backend** is implemented using **Node.js** and **Express.js**. It acts as the application's brain, managing the business logic, authentication, and data processing.

Responsibilities and Key Features:

1. **API Layer**:

- Exposes RESTful API endpoints to handle CRUD operations for users, books, and orders.

- **Example Endpoints:**
    - **GET /api/books:** Fetch all books with optional filters (e.g., genre, author).
    - **POST /api/auth/login:** Authenticate user credentials and generate a JWT.
    - **POST /api/orders:** Save order details to the database.

2. **Authentication and Authorization**:
    - **JWT (JSON Web Tokens)**:
        - Secures routes by issuing tokens upon user login.
        - Tokens are validated in middleware to restrict access to protected endpoints.
    - **Role-based access control (RBAC):**
        - Admins can access routes for adding/editing books.
        - Regular users have restricted permissions.
3. **Business Logic**:
    - Implements the core functionality, such as calculating totals for orders, validating stock availability, and updating order statuses.
4. **Middleware**:
    - **Custom Middleware**:
        - Error handling middleware standardizes API error responses.
        - Logging middleware tracks API requests for debugging.
    - **Third-Party Middleware**:
        - **CORS** for cross-origin requests.
        - **Helmet** for securing HTTP headers.
5. **Scalability**:
    - Non-blocking I/O in Node.js handles multiple simultaneous requests efficiently.
    - The backend is designed with a modular folder structure:
        - **routes/:** Defines API endpoints.

- **controllers/:** Contains business logic for routes.
- **models/:** Defines MongoDB schemas.

3. Database

The **database** layer is powered by **MongoDB**, a NoSQL database well-suited for hierarchical and relational data storage.

Design and Features:

1. **Database Schema**:
    - **Users Collection**:
        - Fields: username, email, passwordHash, role (admin/user), createdAt.
    - **Books Collection**:
        - Fields: title, author, price, genre, description, stock.
    - **Orders Collection**:
        - Fields: userId, bookIds, totalPrice, orderStatus, createdAt.
2. **Scalability**:
    - MongoDB's sharding and replication features ensure the database can handle large volumes of data and requests.
3. **Querying**:
    - **Mongoose** (an ORM for MongoDB) simplifies database operations:
        - Example: Book.find({ genre: 'Fiction' }) fetches all fiction books.
4. **Indexing**:
    - Key fields like title and author are indexed for faster search performance.

4. Communication Flow

The MERN stack ensures seamless communication between layers:

1. **Frontend to Backend**:
    - The frontend makes HTTP requests (e.g., via Axios) to the backend's API endpoints.
    - Example: When the user adds a book to the cart, the frontend sends a POST request to the /api/cart endpoint.
2. **Backend to Database**:

- The backend uses Mongoose to query MongoDB.
- Example: When a user requests the book catalog, the backend fetches the list of books from MongoDB and sends it back to the frontend.

3. **Backend Response**:
    - The backend sends JSON responses to the frontend.
    - Example: After successful login, the backend responds with a JWT token for session management.

5. Security Considerations

- **HTTPS**: Ensures secure communication between the client and server.
- **Environment Variables**: Secrets (like JWT keys) are stored securely in .env files.
- **Input Validation**: Validates user input to prevent SQL injection or NoSQL injection attacks.
- **Password Hashing**: bcryptjs has been used to encrypt the password in the database from being attacked.

4. Setup Instructions for the Book Store Application

1. Prerequisites

Before setting up the project, ensure you have the following software installed on your system:

1. **Node.js and npm**:
    - **Node.js** is required to run JavaScript on the server-side, while **npm** (Node Package Manager) allows you to install dependencies.
    - Download the latest stable version of Node.js from the official website of which will also install npm.
    - Verify the installation by running:

```
node -v   # Check Node.js version
npm -v    # Check npm version
```

2. **MongoDB**:
    - **MongoDB** is the database used for storing user data, book information, and order details.
    - You can either install **MongoDB locally** or use **MongoDB Atlas** (a cloud-based service).

3. **Text Editor**:
    - Install a code editor like **Visual Studio Code (VS Code)**, **Sublime Text**, or **Atom** to write and manage the code.

4. **Browser**:

- Any modern web browser (like Google Chrome or Firefox) to test the application.

2. Setting Up the Project

Step 1: Set up the Backend (Server)

The backend will handle API requests, manage authentication, and interact with the MongoDB database.

1. **Create the Server Folder**:
   - In your terminal, navigate to the directory where you want to create the project and create the backend folder:

     mkdir server
     cd server
2. **Initialize the Project**:
   - Run npm init to initialize a new Node.js project. This will create a package.json file:

     npm init -y

3. **Install Backend Dependencies**:
   - Install required dependencies:

     npm install express mongoose dotenv bcryptjs jsonwebtoken cors

   Here's what each package does:
   - **express:** The web framework for Node.js.
   - **mongoose:** A MongoDB object modeling tool for Node.js.
   - **dotenv:** Loads environment variables from a .env file.
   - **bcryptjs:** Used for hashing passwords.
   - **jsonwebtoken:** Used for generating and verifying JWT tokens for authentication.
   - **cors:** Middleware to enable cross-origin requests.

4. **Set up the Express Server**:
   - Create an index.js file inside the server folder:

     nodemon index.js

- In index.js, set up the basic server with Express:

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// MongoDB Connection
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('MongoDB connected'))
.catch((err) => console.log('MongoDB connection error: ', err));

// Basic Route
app.get('/', (req, res) => {
  res.send('Book Store API is running');
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

5. **Create Environment Variables**:
   - Create a .env file in the server folder:
   - Add the MongoDB URI and a secret key for JWT:

     MONGO_URI=mongodb://localhost:27017/bookstore
     JWT_SECRET=your_secret_key

Step 2: Set up the Frontend (React)

The frontend will be built with **React** to create a user-friendly and responsive interface.

1. **Create the Frontend Folder**:
   - Go back to the root directory and create the client folder:

     npx create-react-app client

2. This will create a React application in the client folder.

3. **Install Frontend Dependencies**:

- Navigate to the client folder:

  cd client

- Install additional dependencies:

  npm install axios react-router-dom

  - **axios:** For making API requests to the backend.
  - **react-router-dom:** For handling routing in the React application.

4. **Set Up React Components**:
   - Create the necessary components for the frontend, such as:
     - **Home.js:** Displays the book catalog.
     - **Login.js:** Handles user authentication.
     - **Cart.js:** Displays the shopping cart.
     - **Signup:** Handles new signup for a new user.

   - You can set up routing in App.js:

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Home from './components/Home';
import Login from './components/Login';
import Cart from './components/Cart';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/cart" element={<Cart />} />
      </Routes>
    </Router>
  );
}

export default App;
```

5. **Connect Frontend with Backend**:
   - Use **Axios** in the frontend to make requests to the backend.
   - Example of fetching books from the backend:

```
import axios from 'axios';

const fetchBooks = async () => {
  const response = await axios.get('http://localhost:5000/api/books');
  console.log(response.data);
};
```

Step 3: Run the Application

1. **Start the Backend**:
   - In the server folder, run:

     node index.js

2. This will start the backend server on **http://localhost:5000.**

3. **Start the Frontend**:
    - In the client folder, run:

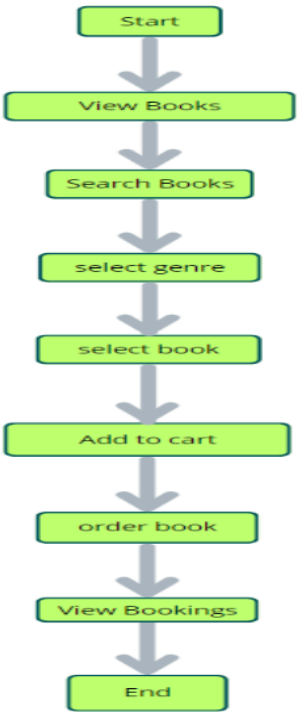
npm start

4. This will start the frontend React app on http://localhost:3000.
5. **Test the Application**:
    - Open a browser and visit http://localhost:3000 to check if the frontend is working.
    - Test the backend by navigating to http://localhost:5000/api/books (after setting up routes for books).

3. Troubleshooting and Notes

- **MongoDB Errors**: Ensure MongoDB is running locally. If using Atlas, double-check the URI in the .env file.
- **CORS Errors**: If the frontend can't make requests to the backend, ensure you have app.use(cors()); in your Express server.
- **Frontend Not Displaying**: Check the browser console for errors and ensure the backend server is running and accessible.

5.Folder Structure



6. Running the Application

To streamline development, you can run both the frontend and backend servers concurrently.

Option 1: Use Two Terminals

- Open two terminal windows or tabs:
    1. In the first terminal, navigate to **server/** and start the backend:

cd server

npm start

    2. In the second terminal, navigate to client/ and start the frontend:

cd client

npm run dev

7. API Documentation for the Book Store Application

This section outlines the API endpoints available in the backend of the **MERN Book Store Application**. The APIs are organized based on their functionality, and the details include HTTP methods, endpoints, request parameters, and example responses.

Base URL

The backend server is hosted at:
http://localhost:5000

All API endpoints will be prefixed with this base URL.

1. Authentication API

1.1 Register User

- **Endpoint**: /api/auth/register
- **Method**: POST
- **Description**: Registers a new user.
- **Request Body:**

```
{
  "username": "john_doe",
  "email": "john.doe@example.com",
  "password": "securePassword123"
}
```

Response:

```
{
  "message": "User registered successfully",
  "user": {
    "id": "64bfcf9021c5f5c72893b123",
    "username": "john_doe",
    "email": "john.doe@example.com"
  }
}
```

1.2 Login User

- **Endpoint**: /api/auth/login
- **Method**: POST
- **Description**: Authenticates a user and generates a JWT token.
- **Request Body:**

```
{
  "email": "john.doe@example.com",
  "password": "securePassword123"
}
```

- **Response:**

```
{
    "id": "6733547b7afb8351b9acb0a6",
    "role": "user",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJhdXRoQ2xhaW1zIjpbeyJuYW1lIjoiU3VyeWEifSx7InJvbGUiOiJ1c2VyIn1dLCJpYXQiO
        jE3MzE3MzExMDYsImV4cCI6MTczNDMyMzEwNn0.
        u6zdPRxGBWygWzGgglAU2axYPFyHHsPCkCAAitaWlUk"
}
```

2. Books API

2.1 Get All Books

- **Endpoint**: /api/books
- **Method**: GET
- **Description**: Retrieves a list of all available books.

- **Response**


```
_id: ObjectId('67338475f46bb69a1201c8f5')
url : "https://m.media-amazon.com/images/I/51Hfv2MfNGL._SY445_SX342_.jpg"
title : "Rich dad Poor dad"
author : "Robert T Kiyosaki"
price : 300
desc : "Rich Dad Poor Dad by Robert Kiyosaki is a personal finance classic tha…"
language : "English"
createdAt : 2024-11-12T16:38:13.975+00:00
updatedAt : 2024-11-12T17:02:31.492+00:00
__v : 0
```

```
_id: ObjectId('67338c6e0b330cc8dbe9c580')
url : "https://m.media-amazon.com/images/I/41qowEITwjL._SY445_SX342_.jpg"
title : "The Intelligent Investor"
author : "Benjamin Graham"
price : 300
desc : "The Intelligent Investor by Benjamin Graham is a seminal book on inves…"
language : "English"
createdAt : 2024-11-12T17:12:14.064+00:00
updatedAt : 2024-11-12T17:12:14.064+00:00
__v : 0
```


2.2 Get a Single Book

- **Endpoint**: /api/books/:id
- **Method**: GET
- **Description**: Retrieves details of a single book by its ID.
- **Parameters**:
    - id (string): The unique identifier of the book.

Response:


```
_id: ObjectId('67338475f46bb69a1201c8f5')
url : "https://m.media-amazon.com/images/I/51Hfv2MfNGL._SY445_SX342_.jpg"
title : "Rich dad Poor dad"
author : "Robert T Kiyosaki"
price : 300
desc : "Rich Dad Poor Dad by Robert Kiyosaki is a personal finance classic tha…"
language : "English"
createdAt : 2024-11-12T16:38:13.975+00:00
updatedAt : 2024-11-12T17:02:31.492+00:00
__v : 0
```

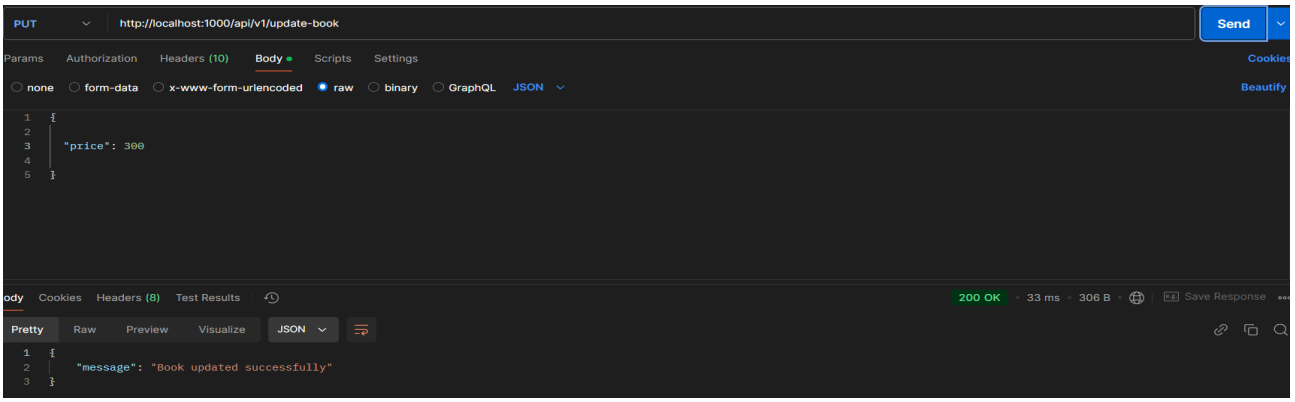

2.3 Add a New Book

- **Endpoint**: /api/books
- **Method**: POST
- **Description**: Adds a new book to the store (Admin-only functionality).

- **Request Body:**

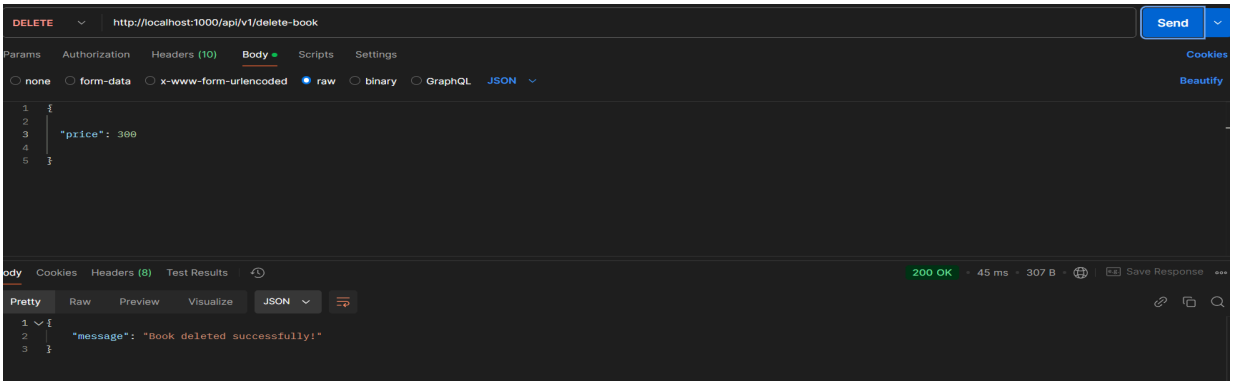
2.4 Update a Book

- **Endpoint**: /api/books/:id
- **Method**: PUT
- **Description**: Updates details of an existing book by its ID (Admin-only functionality).



2.5 Delete a Book

- **Endpoint**: /api/books/:id
- **Method**: DELETE
- **Description**: Deletes a book from the store by its ID (Admin-only functionality).



3. Orders API

3.1 Create an Order

- **Endpoint**: /api/orders
- **Method**: POST
- **Description**: Creates a new order for the user.
- **Request Body**:

```
{
  "books": [
    {
      "id": "64bfccae21c5f5c72893a567",
      "quantity": 2
    },
    {
      "id": "64bfccae21c5f5c72893a568",
      "quantity": 1
    }
  ]
}
```

- **Response:**

```
{
  "message": "Order placed successfully",
  "order": {
    "id": "64bfcf9021c5f5c72893a456",
    "books": [
      {
        "id": "64bfccae21c5f5c72893a567",
        "quantity": 2
      },
      {
        "id": "64bfccae21c5f5c72893a568",
        "quantity": 1
      }
    ],
    "total": 34.97,
    "status": "Pending"
  }
}
```


Get User Orders

- **Endpoint**: /api/orders
- **Method**: GET
- **Description**: Retrieves all orders placed by the logged-in user.
- **Response**:


```
[
  {
    "id": "64bfcf9021c5f5c72893a456",
    "books": [
      {
        "title": "The Great Gatsby",
        "quantity": 2,
        "price": 10.99
      },
      {
        "title": "1984",
        "quantity": 1,
        "price": 12.99
      }
    ],
    "total": 34.97,
    "status": "Pending"
  }
]
```


8. Authentication in the Book Store Application

Authentication is an essential part of the MERN Book Store application. It ensures secure access to protected resources, such as adding books, placing orders, and viewing user-specific data. The application uses **JWT (JSON Web Token)** for secure, stateless authentication.

Authentication Features

1. **User Registration**: Allows new users to sign up.
2. **User Login**: Verifies user credentials and issues a JWT token.
3. **Token Verification**: Validates the JWT token for accessing protected routes.
4. **Role-Based Access Control (RBAC)**: Restricts certain actions (like adding or deleting books) to admin users only.

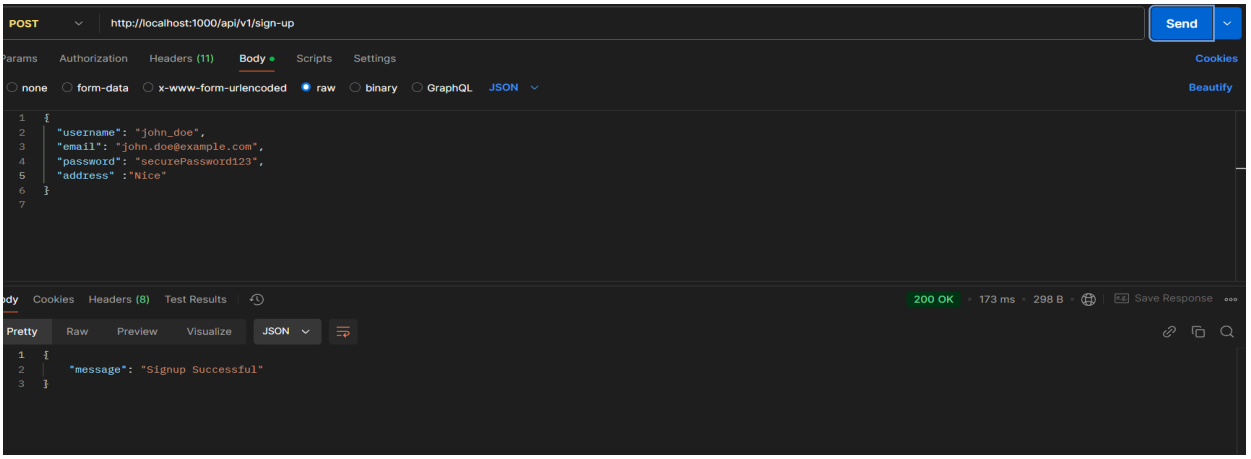
Workflow

1. **Registration**:
    - A new user provides a username, email, and password.
    - The server hashes the password using **bcrypt** and stores it securely in the database. After successful registration, the user can log in.
2. **Login**:
    - The user submits their email and password.
    - The server validates the credentials against the database.
    - If valid, the server generates a **JWT token** and returns it to the user.
    - The user includes this token in the Authorization header of subsequent requests to access protected routes.
3. **Token Validation**:
    - Each protected route checks for a valid JWT token.
    - If the token is missing, invalid, or expired, access is denied.

Implementation Details

1. Registration API

- **Endpoint**: /api/auth/register
- **Method**: POST
- **Request Body**:



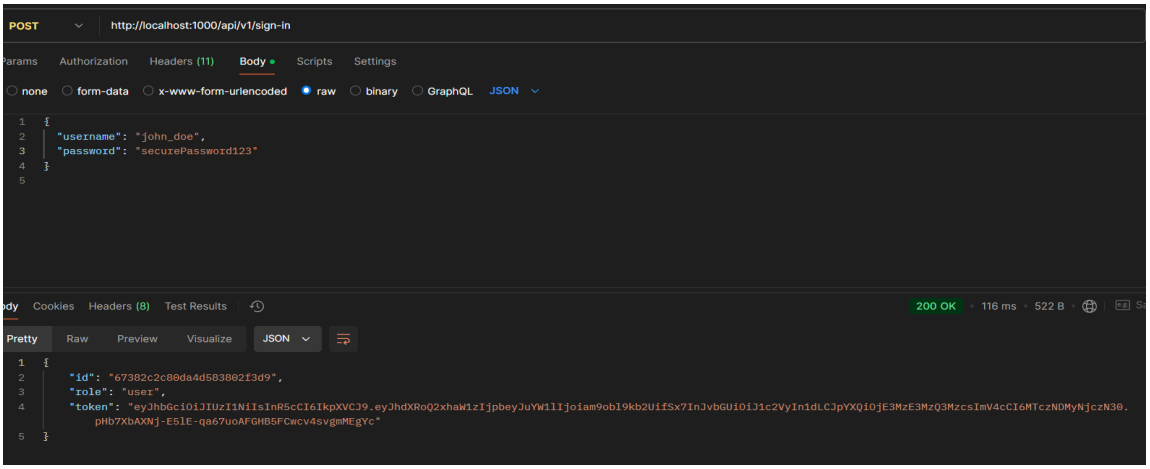
Workflow:

- The server validates the input data.
- The password is hashed using **bcrypt** before being saved to the database.
- The server returns a success message upon successful registration.


```
_id: ObjectId('67382c2c80da4d583802f3d9')
username : "john_doe"
email : "john.doe@example.com"
password : "$2a$10$1BOva.mQEbMj/ej6MR4s1uyc/d8iKoqbNNAKK23v.bHdb0bvLeRDC"
address : "Nice"
avatar : "https://cdn-icons-png.flaticon.com/128/1377/3177440.png"
role : "user"
favourites : Array (empty)
cart : Array (empty)
orders : Array (empty)
createdAt : 2024-11-16T05:22:52.655+00:00
updatedAt : 2024-11-16T05:22:52.655+00:00
__v : 0
```


2. Login API

- **Endpoint**: /api/auth/login
- **Method**: POST
- **Request Body**:



Workflow:

- The server retrieves the user by email.
- The submitted password is compared with the stored hashed password using **bcrypt**.
- If authentication succeeds, a **JWT token** is generated and sent back to the client.

3. Protecting Routes with Middleware

A middleware function checks for a valid JWT token in the Authorization header before allowing access to protected routes.

- **Middleware Code**:

```
const jwt = require('jsonwebtoken');

function authenticateToken(req, res, next) {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).json({ message: "Access denied. No token provided." });

  try {
    const verified = jwt.verify(token, process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(403).json({ message: "Invalid token." });
  }
}

module.exports = authenticateToken;
```

Role-Based Access Control (RBAC)

To restrict certain actions to admin users:

- **Middleware Code**:

```
function isAdmin(req, res, next) {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ message: "Access denied. Admins only." });
  }
  next();
}

module.exports = isAdmin;
```

JWT Token Structure

The **JWT token** consists of three parts:

1. **Header**: Contains metadata about the token, like the type (JWT) and hashing algorithm.
2. **Payload**: Contains user-specific data (e.g., id, role, email).
3. **Signature**: A hashed string that verifies the token's authenticity

Example Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiIxMjM0NTY3ODkwIiwicm9sZSI6InVzZXIiLCJpYXQiOjE1MTYyMzkwMjJ9.s5bMJvtiF8jFnOC_kNeQPji6mBhmFffNZoQxEHOEadY

9. User Interface

The user interface (UI) of the Book Store application is designed using React.js, providing a responsive, user-friendly, and intuitive experience. The application aims to deliver seamless navigation for book browsing, user authentication, and order management.

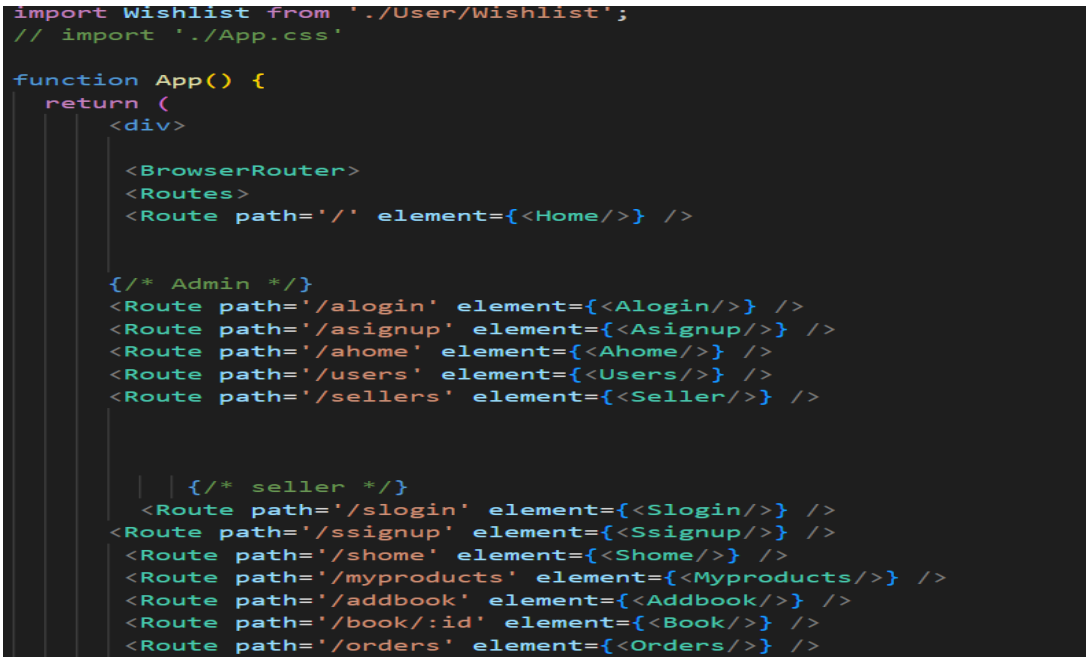
1. UI Components

The UI is built using modular React components, ensuring maintainability and reusability. Below is an overview of the primary components and their functionalities:

1.1 Navbar

- **Description:** Displays navigation options like Home, Books, Cart, and Profile.
- **Features:**
    - Dynamic links based on authentication status (e.g., Login/Register for unauthenticated users, Logout for authenticated users).
    - Shopping cart icon with a count of items.
- **Implementation:**
    - Styled using CSS/Bootstrap/Tailwind for responsive

```
import 'bootstrap/dist/css/bootstrap.min.css';
import Login from './User/Login'
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import Signup from './User/Signup';
import Unavbar from './User/Unavbar';
import Addbook from './Seller/Addbook';
import Item from './Seller/Book';
import Myproducts from './Seller/Myproducts';
import Slogin from './Seller/Slogin';
import Book from './Seller/Book';
import Orders from './Seller/Orders';
import Products from './User/Products';
import Uitem from './User/Uitem';
import Myorders from './User/Myorders';
import Uhome from './User/Uhome';
import OrderItem from './User/OrderItem';
import Shome from './Seller/Shome';
import Ssignup from './Seller/Ssignup';
import Home from './Componenets/Home';
import Alogin from './Admin/Alogin';
import Asignup from './Admin/Asignup';
import Ahome from './Admin/Ahome';
import Users from './Admin/Users';
import Vendors from './Admin/Seller';
import Seller from './Admin/Seller';
import Wishlist from './User/Wishlist';
// import './App.css'
```



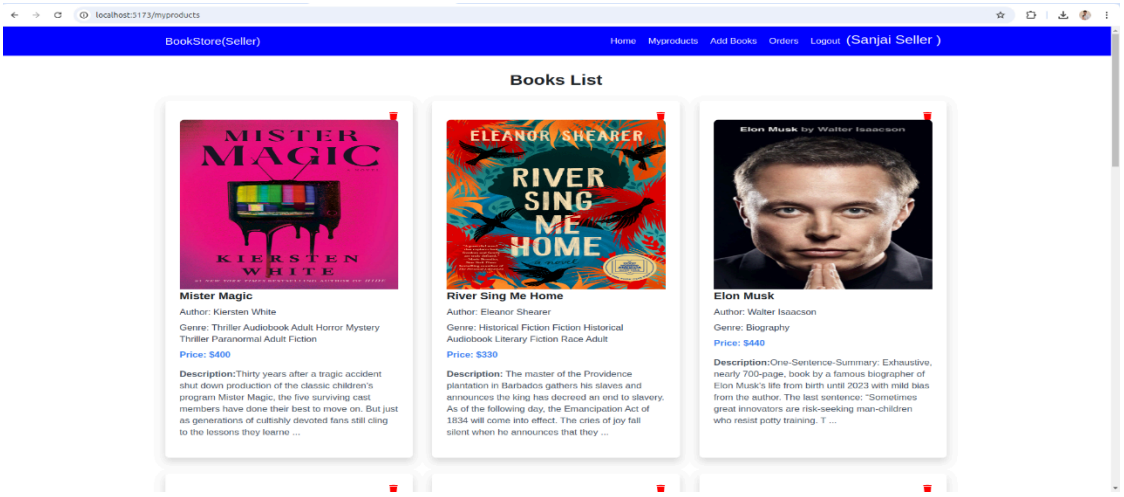
```
import Wishlist from './User/Wishlist';
// import './App.css'

function App() {
  return (
      <div>

       <BrowserRouter>
       <Routes>
       <Route path='/' element={<Home/>} />


      {/* Admin */}
      <Route path='/alogin' element={<Alogin/>} />
      <Route path='/asignup' element={<Asignup/>} />
      <Route path='/ahome' element={<Ahome/>} />
      <Route path='/users' element={<Users/>} />
      <Route path='/sellers' element={<Seller/>} />



        {/* seller */}
       <Route path='/slogin' element={<Slogin/>} />
      <Route path='/ssignup' element={<Ssignup/>} />
       <Route path='/shome' element={<Shome/>} />
       <Route path='/myproducts' element={<Myproducts/>} />
       <Route path='/addbook' element={<Addbook/>} />
       <Route path='/book/:id' element={<Book/>} />
       <Route path='/orders' element={<Orders/>} />
```

1.2 Home Page

- **Description**: The landing page of the application, providing an overview of the store.
- **Features**:
    - A welcome banner highlighting featured books or promotions.
    - Quick links to popular categories or newly added books.

1.3 Books Page

- **Description**: Displays a list of books available in the store.
- **Features**:
    - A **grid layout** showcasing book details like title, author, price, and a thumbnail image.
    - Search bar and filters for genre, price range, or author.
    - "Add to Cart" button for each book.



1.4 Book Details Page

- **Description**: Provides detailed information about a specific book.
- **Features**:
    - Full description, stock availability, and reviews.
    - "Add to Cart" and "Buy Now" buttons.
    - Related books section for suggestions.

1.6 Authentication Pages

- **Login Page**:
    - Simple form with fields for email and password.
    - Login button triggers API call to authenticate the user.
    - Link to "Register" page for new users.
- **Register Page**:
    - Form with fields for username, email, and password.
    - Register button triggers the user registration API.

1.7 User Profile Page

- **Description**: Displays user-specific information.
- **Features**:
    - List of orders placed by the user.
    - Option to update profile details or logout.

1.8 Admin Panel:

- **Description**: Accessible only to admin users.
- **Features**:
    - Add, update, or delete books.
    - View user orders and manage inventory.

2. Responsive Design

- **Framework**: The UI is styled using **CSS**, **Bootstrap**, or **Tailwind CSS** for responsiveness.
- **Mobile Optimization**:
    - Collapsible Navbar for small screens.
    - Grid layouts adjust dynamically for different screen sizes.

3. State Management

- **React Context API** or **Redux** is used to manage the application state efficiently.
- **Examples**:
    - User authentication status is stored globally to display dynamic navbar links.
    - Shopping cart data is stored and updated in real-time.

4. Error Handling

- **Features**:
    - Custom error pages (e.g., 404 for invalid routes).
    - User-friendly error messages for API failures (e.g., "Unable to fetch books").

10. Testing

Testing ensures the reliability and functionality of the Book Store application. The testing process includes both manual and automated tests across various levels of the application.

Types of Testing

1. Unit Testing

- **Purpose**: Test individual components and functions in isolation.
- **Tools**: Jest and React Testing Library.

- **Examples**:
    - Test the BookCard component to ensure it renders book details correctly.
    - Validate that the addToCart function updates the cart state as expected.

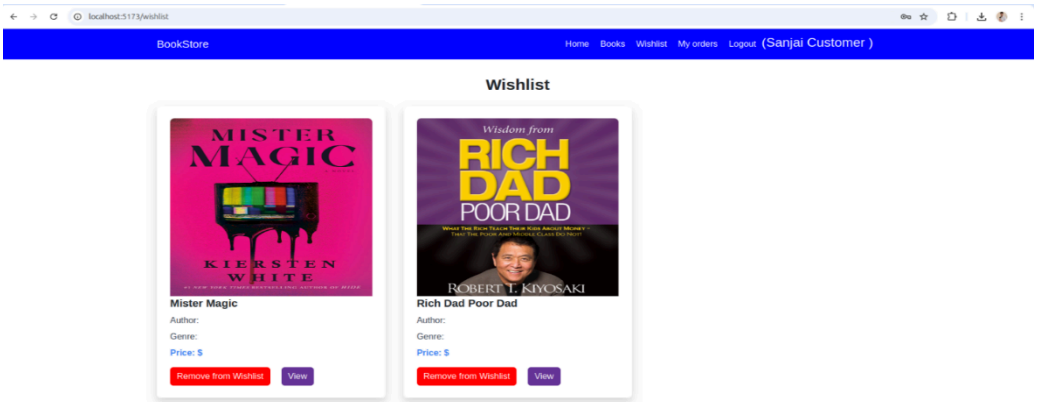
2. Integration Testing

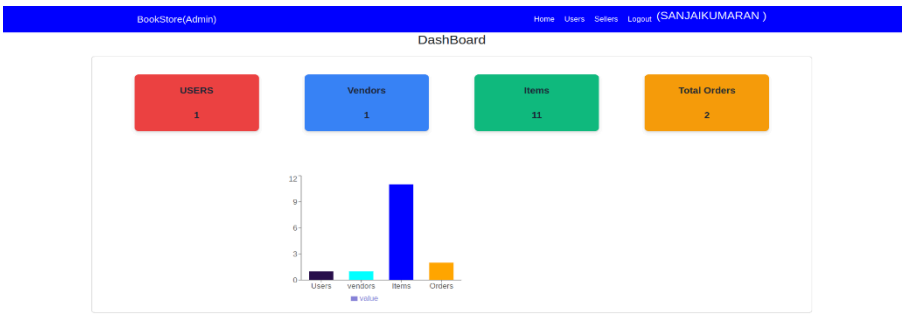
- **Purpose**: Ensure that different parts of the application work together.
- **Tools**: Jest and Supertest.
- **Examples**:
    - Test the API endpoints for adding books, ensuring they integrate correctly with the database.
    - Validate that the BooksPage fetches and displays data from the backend.

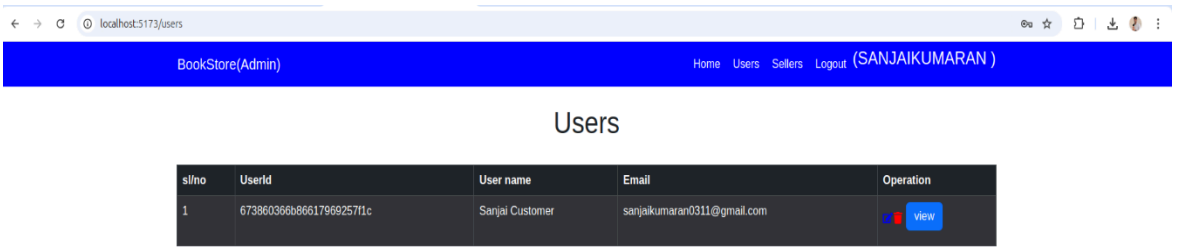
3. End-to-End (E2E) Testing

- **Purpose**: Simulate user interactions to test the entire application flow.
- **Tools**: Cypress or Playwright.
- **Examples**:
    - Verify that a user can log in, browse books, add items to the cart, and place an order.
    - Test error handling for invalid user input on the login page.

11. Screenshots of the Project

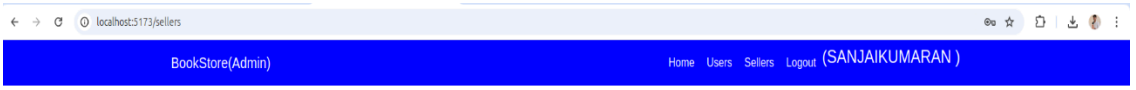






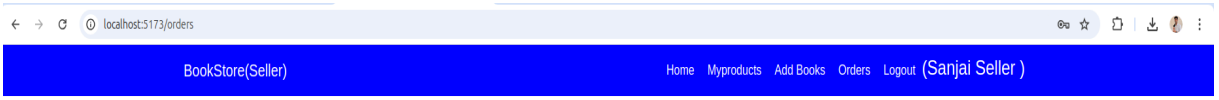
Users

sl/no	UserId	User name	Email	Operation
1	673860366b866179692571c	Sanjai Customer	sanjaikumaran0311@gmail.com	view



Vendors

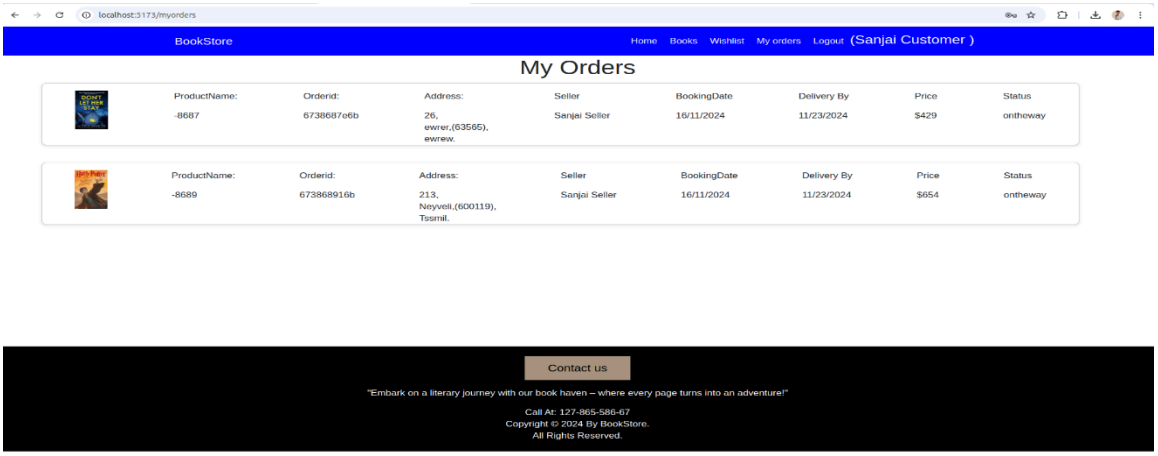
sl/no	UserId	User name	Email	Operation
1	6738512e6b866179692571f0a	Sanjai Seller	sanjaikumarancse@gmail.com	view

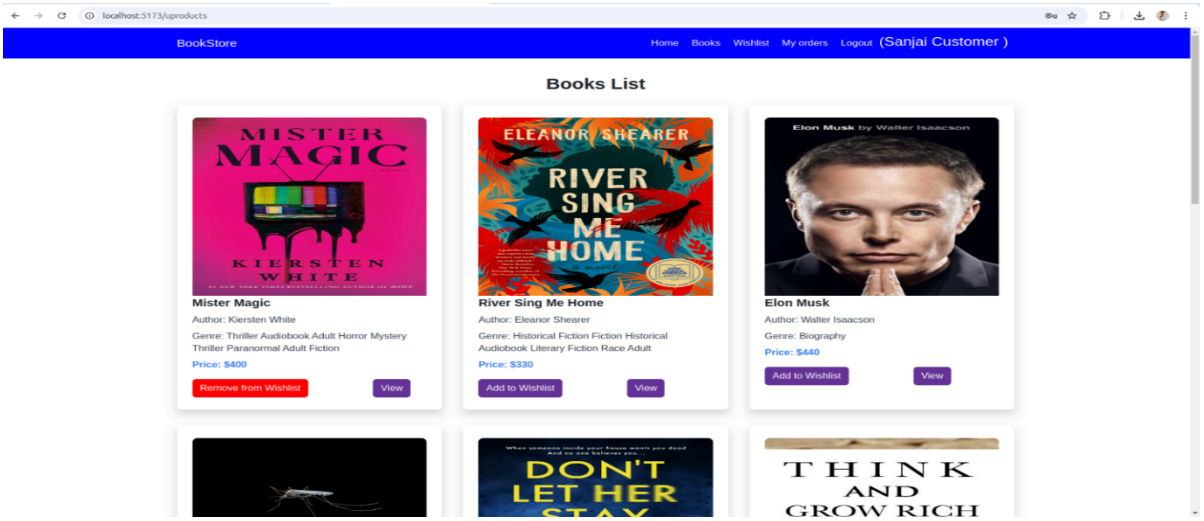


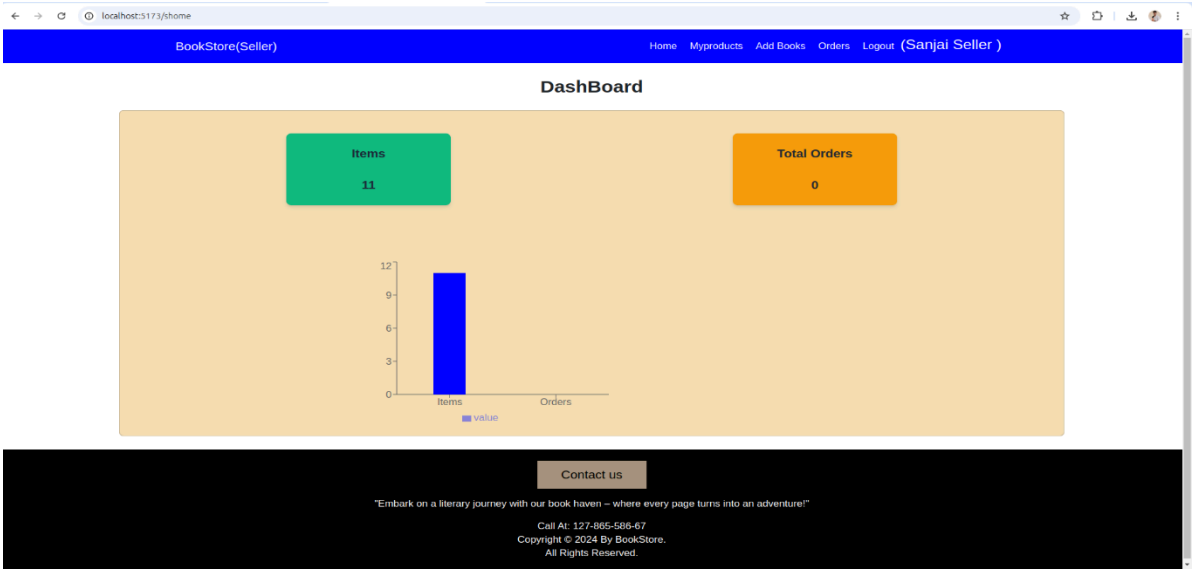
Orders

ProductName:	Orderid:	Customer Name	Address:	BookingDate	Delivery By	Warranty	Price	Status
	6738687e6b	Sanjai Customer	26, ewrer,(63565), ewrew.	16/11/2024	11/23/2024	1 year	429	ontheway

ProductName:	Orderid:	Customer Name	Address:	BookingDate	Delivery By	Warranty	Price	Status
	673868916b	Sanjai Customer	213, Neyveli,(600119), Tssmil.	16/11/2024	11/23/2024	1 year	654	ontheway


localhost:5173/myorders
BookStore
Home Books Wishlist My orders Logout (Sanjai Customer)
My Orders
ProductName: -8687
Orderid: 6738687e6b
Address: 26, ewrer,(63565), ewrew.
Seller Sanjai Seller
BookingDate 16/11/2024
Delivery By 11/23/2024
Price $429
Status ontheway
ProductName: -8689
Orderid: 673868916b
Address: 213, Neyveli,(600119), Tnsmil.
Seller Sanjai Seller
BookingDate 16/11/2024
Delivery By 11/23/2024
Price $654
Status ontheway
Contact us
"Embark on a literary journey with our book haven – where every page turns into an adventure!"
Call At: 127-865-586-67
Copyright © 2024 By BookStore.
All Rights Reserved.


localhost:5173/uproducts
BookStore
Home Books Wishlist My orders Logout (Sanjai Customer)
Books List
Mister Magic
Author: Kiersten White
Genre: Thriller Audiobook Adult Horror Mystery Thriller Paranormal Adult Fiction
Price: $400
Remove from Wishlist
View
River Sing Me Home
Author: Eleanor Shearer
Genre: Historical Fiction Fiction Historical Audiobook Literary Fiction Race Adult
Price: $330
Add to Wishlist
View
Elon Musk
Author: Walter Isaacson
Genre: Biography
Price: $440
Add to Wishlist
View


localhost:5173/shome
BookStore(Seller)
Home Myproducts Add Books Orders Logout (Sanjai Seller)
DashBoard
Items
11
Total Orders
0
12
9
6
3
0
Items
Orders
value
Contact us
"Embark on a literary journey with our book haven – where every page turns into an adventure!"
Call At: 127-865-586-67
Copyright © 2024 By BookStore.
All Rights Reserved.

12. Known Issues

- **Limited Payment Gateway Integration**: The Book Store application currently lacks a fully integrated payment processing system, limiting users' ability to make real-world purchases through credit cards, PayPal, or other gateways.

- **Search Functionality Limitations**: The Book Store application's search feature, despite offering basic search options like title, author, and genre, struggles with accuracy and refinement, necessitating improved filtering and typo tolerance.

- **Responsiveness on Older Devices**: The application, using modern CSS frameworks like Bootstrap and Tailwind, faces compatibility issues on older mobile devices and browsers, especially with smaller screen sizes and outdated versions.

13. Future Enhancements

- **Integration of Payment Gateway:** The application will integrate secure payment gateways like PayPal, Stripe, and Razorpay for seamless purchase experiences, supporting multiple payment methods like craedit/debit cards, digital wallets, and net banking.

- **Advanced Search and Recommendation Engine**
  The website will improve search functionality with natural language processing, advanced filtering, and a machine learning-powered recommendation engine, enhancing user engagement and personalizing the shopping experience.

- **Mobile Application Development**
  A mobile app for Android and iOS will enhance the Book Store's accessibility, offering offline browsing, push notifications, and a user-friendly interface.

- **User Reviews and Ratings**
  The Book Store will introduce a review and rating system, enabling users to provide feedback on their reading experiences, fostering an interactive community.

- **Admin Dashboard for Inventory and User Management**
  The admin dashboard will enhance operational efficiency by allowing administrators to manage inventory, view sales analytics, and oversee user activity, providing valuable insights into user behavior and sales trends.