

# Full Stack Development with MERN

## 1. Introduction

**PROJECT TITLE:**Book Store App

### **TEAM MEMBERS:**

**R. Bala Praveen - Developer:** Responsible for both frontend and backend development, ensuring integration between the components, and implementing key features.

**B. Archana - Developer:** Backend developer responsible for creating and managing Express routes, server-side logic, and API integration.

**S. Barakath Safeena - Developer:** Focused on frontend development using React, including UI design and functionality for user interactions.

**D. Dharshini - Developer:** Database management with MongoDB, including schema design, data storage, and optimization for queries.

**A. Dhanush - Developer:** Responsible for testing, debugging, and ensuring reliability, as well as managing the deployment process.

## 2.Project Overview

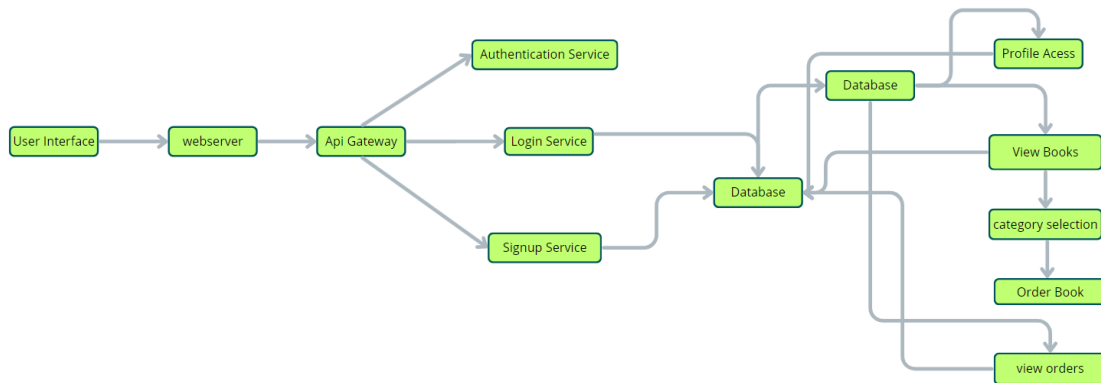
**Purpose:** The purpose of the Book Store app is to provide users with a seamless and convenient platform to browse, select, and purchase a wide variety of books. BookStore aims to enhance the book-shopping experience by offering an intuitive interface, efficient search and filtering options, and easy access to personalized

profiles and order histories. The primary goal is to make book discovery and purchasing simple, enjoyable, and accessible for users.

**Features:**

1. **User Profiles:** Users can create and access personal profiles, where they can manage personal details, preferences, and view order history.
2. **Book Catalog Browsing:** The app offers an extensive collection of books with options to view detailed information (title, author, genre, price, etc.).
3. **Book Search and Genre Filter:** Users can search for books by title or author and filter them by genre to find specific types of books quickly.
4. **Add to Cart and Quantity Selection:** Users can add books to their cart, specify quantities, and manage their selections before proceeding to purchase.
5. **Purchase Options:** The app allows users to select options like e-book format or special editions when available.
6. **Order Tracking:** Users can view current and past orders, including order status, payment details, and history.
7. **Order Confirmation:** After placing an order, users receive an order confirmation, ensuring clarity and transparency in the purchase process.

### 3. Architecture



*Bookstore architecture*

#### Frontend (User Interface)

The frontend is designed using **React**, providing a seamless and interactive user experience for accessing the various features of the book store. Key components include:

- **Login and Signup Components:** These interact with the backend's API Gateway to authenticate users.
- **Book Viewing and Ordering Components:** Users can browse, select categories, and order books.
- **Profile and Order Viewing Components:** Allows users to access their profile information and review their previous orders.

The frontend communicates with the backend via HTTP requests, typically through RESTful API endpoints.

#### Backend (Node.js and Express.js)

The backend is structured with **Node.js** and **Express.js** as follows:

- **API Gateway:** The central hub for routing client requests. It interacts with various services (authentication, login, signup) and manages the flow of data between the frontend and backend services.
- **Authentication Service:** Handles token-based authentication, issuing JWTs upon successful login. This service ensures secure access to authorized endpoints.
- **Login and Signup Services:** Handle user credentials, creating and verifying accounts against the MongoDB database.
- **Database Interaction:** Interfaces with MongoDB for data persistence, including user profiles, book listings, categories, and orders.

## Database (MongoDB)

The **MongoDB database** serves as the primary data storage, managing:

- **User Collection:** Stores user information, including profile details and authentication data.
- **Book Collection:** Contains details of books, organized by categories.
- **Order Collection:** Stores order history and user-specific order information.
- **Schema and Interactions:** Each service interacts with MongoDB using Mongoose models to perform CRUD operations for storing and retrieving data. The database schema is designed to support relations between users, books, and orders, enabling efficient querying and updates.

## 4.Setup Instructions

### Prerequisites

Before setting up the Book-Store project, ensure the following software dependencies are installed:

1. **Node.js** - For running the server and managing dependencies.
2. **MongoDB** - For the database to store user, seller, and book information.
3. **Vite** - For managing the frontend development and bundling.
4. **Git** - For cloning the repository.

### Installation

#### 1. Clone the Repository

```
git clone <repository-url>
```

```
cd Book-Store
```

#### 2. Backend Setup

1. **Navigate to the backend folder:**

```
cd Backend
```

2. **Install backend dependencies:**

```
npm install
```

3. **Set up environment variables:**

- Create a .env file in the Backend directory with the following:

```
PORT=4000
```

```
MONGO_URI=<your-mongodb-uri>
```

```
JWT_SECRET=<your-jwt-secret>
```

#### **4. Start the backend server:**

```
node server.js
```

### **3. Frontend Setup**

#### **1.Navigate to the frontend folder:**

```
cd ../frontend
```

#### **2.Install frontend dependencies:**

```
npm install
```

#### **3.Run the frontend server:**

```
npm run dev
```

#### **4. Access the Application**

- Open your browser and navigate to `http://localhost:<port>` (replace `<port>` with the frontend server port, typically 3000).

## **5. Folder Structure**

### **Client (React Frontend)**

Located in `frontend/src`, the React frontend is organized as follows:

#### **1. Admin, Seller, User:**

- Each folder represents a user type in the application, containing components and pages specific to that role.
- Example: Admin may have components for managing books, users, and orders; Seller may include components for managing their own

inventory; and User might have components for browsing, adding books to the cart, and placing orders.

## **2. Components:**

- Contains reusable components that are shared across different pages, such as navigation bars, buttons, forms, and modals.
- Helps keep code modular and avoids redundancy by reusing common elements.

## **3. App.jsx:**

- The main application file where routes are defined, linking different components to specific URLs.
- This serves as the entry point for the app, rendering components based on user actions and navigation.

## **4. main.jsx:**

- The root file where the React app is rendered into the DOM.
- Links the app to the HTML file and imports global settings or context providers.

## **5. App.css & index.css:**

- CSS files for styling. App.css contains app-specific styles, while index.css may hold global styles that apply across all components.

## **Server (Node.js Backend)**

Located in the Backend directory, the Node.js backend is organized as follows:

### **1. db:**

- Contains subfolders for Seller and Users to manage database schemas and models.
- Also includes config.js for database configuration, where the MongoDB connection is established.

## **2. uploads:**

- A folder for storing any files uploaded by users, such as book images or user profile pictures.

## **3. server.js:**

- The main server file where the Express server is initialized and configured.
- Defines routes, middleware, and other essential server-level settings.

## **4. package.json & package-lock.json:**

- Lists the dependencies and scripts required for the backend.
- Contains npm configurations to manage the backend environment.

# **6. Running the Application**

## **Running the Application Locally**

To start both the frontend and backend servers, follow these steps:

1.Navigate to the frontend directory:

```
cd book-store/frontend
```

2.Start the frontend server:

```
npm start
```



This command will run the frontend in development mode, typically on <http://localhost:3000>.

## **Backend**

1. Open a new terminal window and navigate to the backend directory:

```
cd book-store\backend
```

2. Start the backend server:

```
npm start
```

- This will start the backend server on the port defined in your .env file (e.g., <http://localhost:4000>).

With both servers running, the frontend will communicate with the backend, allowing full functionality for the Book-Store app.

## **7. API Documentation**

### **General Information**

- **Base URL:** <http://localhost:4000>

### **Admin Endpoints**

#### **1. Admin Login**

- **Endpoint:** /login
- **Method:** POST
- **Description:** Allows an admin to log in.
- **Request Body**

```
{  
  
  "email": "admin@example.com",  
  
  "password": "password123"  
}
```

- **Response:**
- Success: { "Status": "Success", "user": { "id": "user\_id", "name": "Admin Name", "email": "admin@example.com" } }
- Failure: "login fail" or "no user"

## 2. Admin Registration

- **Endpoint:** /signup
- **Method:** POST
- **Description:** Registers a new admin.
- **Request Body**

```
{  
  
  "name": "Admin Name",  
  
  "email": "admin@example.com",  
  
  "password": "password123"  
}
```

- **Response:** "Account Created" or "Already have an account"

## User Endpoints

### 3. Get All Users

- **Endpoint:** /users

- **Method:** GET
- **Description:** Retrieves all users.
- **Response:** Array of user objects.

#### 4. Delete User

- **Endpoint:** /userdelete/:id
- **Method:** DELETE
- **Description:** Deletes a user by ID.
- **Response:** 200 OK on success, or an error message.

#### 5. User Login

- **Endpoint:** /login
- **Method:** POST
- **Description:** Logs in a user.
- **Request Body:**

```
{  
  "email": "user@example.com",  
  "password": "password123"  
}
```

- **Response:** Success with user info, or failure messages.

#### 6. User Registration

- **Endpoint:** /signup
- **Method:** POST
- **Description:** Registers a new user.

- **Request Body**

```
{  
  
  "name": "User Name",  
  
  "email": "user@example.com",  
  
  "password": "password123"  
}
```

- **Response:** "Account Created" or "Already have an account"

## **Seller Endpoints**

### **7. Seller Login**

- **Endpoint:** /slogin
- **Method:** POST
- **Description:** Logs in a seller.
- **Request Body**

```
{  
  
  "email": "seller@example.com",  
  
  "password": "password123"  
}
```

- **Response:** Success with seller info, or failure messages.

### **8. Seller Registration**

- **Endpoint:** /ssignup

- **Method:** POST
- **Description:** Registers a new seller.
- **Request Body**

```
{
  "name": "Seller Name",
  "email": "seller@example.com",
  "password": "password123"
}
```

- **Response:** "Account Created" or "Already have an account"

## 9. Add Book

- **Endpoint:** /items
- **Method:** POST
- **Description:** Adds a new book item.
- **Request Body:** Form-data with book details and itemImage file.
- **Response:** Created item object or error message.

## 10. Get Seller's Items

- **Endpoint:** /getitem/:userId
- **Method:** GET
- **Description:** Retrieves items uploaded by a seller.
- **Response:** Array of item objects.

## Order Endpoints

### 11. Add Order

- **Endpoint:** /userorder
- **Method:** POST
- **Description:** Creates a new order.
- **Request Body**

```
{  
  
  "flatno": "123",  
  
  "city": "City",  
  
  "state": "State",  
  
  "pincode": "123456",  
  
  "totalamount": 100,  
  
  "seller": "Seller Name",  
  
  "sellerId": "seller_id",  
  
  "BookingDate": "2024-11-14",  
  
  "description": "Order description",  
  
  "Delivery": "Delivery details",  
  
  "userId": "user_id",  
  
  "userName": "User Name",  
  
  "booktitle": "Book Title",  
  
  "bookauthor": "Author",
```

```
"bookgenre": "Genre",  
"ItemImage": "path/to/image.jpg"  
}
```

- **Response:** Created order object or error message.

## 12. Get User Orders

- **Endpoint:** /getorders/:userId
- **Method:** GET
- **Description:** Retrieves orders placed by a user.
- **Response:** Array of order objects.

## Wishlist Endpoints

### 13. Get All Wishlist Items

- **Endpoint:** /wishlist
- **Method:** GET
- **Description:** Retrieves all wishlist items.
- **Response:** Array of wishlist item objects.

### 14. Add Item to Wishlist

- **Endpoint:** /wishlist/add
- **Method:** POST
- **Description:** Adds an item to the wishlist.

### **Request Body:**

json

Copy code

```
{  
  
  "itemId": "item_id",  
  
  "title": "Item Title",  
  
  "itemImage": "path/to/image.jpg",  
  
  "userId": "user_id",  
  
  "userName": "User Name"  
}
```

- **Response:** Created wishlist item or error message.

### **15. Remove Item from Wishlist**

- **Endpoint:** /wishlist/remove
- **Method:** POST
- **Description:** Removes an item from the wishlist.
- **Request Body**

```
{  
  
  "itemId": "item_id"  
}
```

- **Response:** Success message or error message.



## 8. Authentication

### 1.Authentication:

- When a user logs in (for both /login and /slogin endpoints for users and sellers respectively), their credentials (email and password) are validated against the stored data in the database.
- Upon successful login, the server could issue a **JSON Web Token (JWT)** to the user. This token would be signed using a secret key and contain encoded user information, such as user ID and role, which could be used for further authorization.
- The token is then sent back to the client (frontend) and stored locally, usually in cookies or local storage, to be included in subsequent requests.

### 2.Authorization:

- For secure endpoints (such as creating orders or modifying items), the client should send the JWT in the Authorization header as a Bearer token.
- Middleware on the server validates the JWT by verifying its signature and checking for its expiry.
- Once validated, the middleware extracts user details from the token payload, allowing the server to determine if the user has the required permissions to access the endpoint.
- For example, only users with an "admin" role could access specific admin routes, while sellers could access their product management routes.

### 3.Token Expiration and Renewal:

- JWTs typically have an expiration time for security purposes. Upon expiration, the user might need to log in again, or a refresh token mechanism can be implemented to issue new tokens without re-authentication.

## 9. User Interface

BookStore

User Seller Admin

### Login to user account

Email address

Password

[Log in](#)

Don't have an account? [Create](#) [Signup](#)

### Signup

Name

Email address

Password

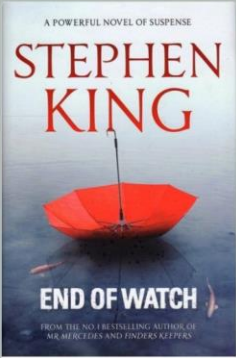
[Signup](#)

Already have an account [Login](#)

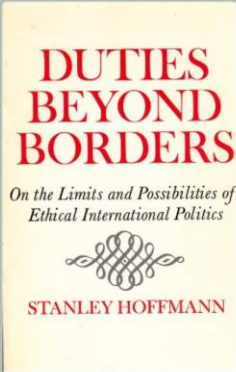
BookStore

[Home](#)
[Books](#)
[Wishlist](#)
[My orders](#)
[Logout\(arch \)](#)

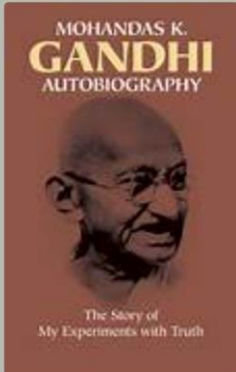
Best Seller



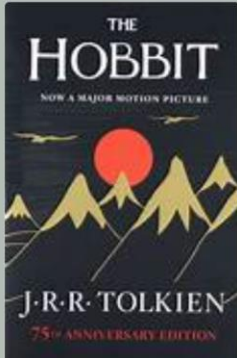
STEPHEN KING



DUTIES BEYOND BORDERS



MOHANDAS GANDHI AUTOBIOGRAPHY




THE HOBBIT

BookStore

[Home](#)
[Books](#)
[Wishlist](#)
[My orders](#)
[Logout\(arch \)](#)

Wishlist



python Book

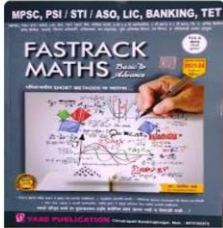
Author:

Genre:

Price: \$

Remove from Wishlist

View



maths book

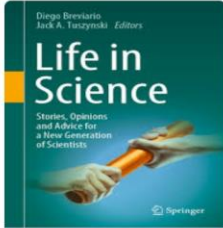
Author:

Genre:

Price: \$

Remove from Wishlist

View



science book

Author:


Genre:


Price: \$


Remove from Wishlist

View

My Orders

	ProductName:	Orderid:	Address:	Seller	BookingDate	Delivery By	Price	Status
	-4c2b	6734c2b88d	, ,(), .	Archana.B	13/11/2024	11/20/2024	\$1099	ontheway

	ProductName:	Orderid:	Address:	Seller	BookingDate	Delivery By	Price	Status
	-7603	6737603fa7	6, Thiruninravur,(39), tamilnadu.	Ramu	15/11/2024	11/22/2024	\$1099	ontheway

	ProductName:	Orderid:	Address:	Seller	BookingDate	Delivery By	Price	Status
	-7608	67376086a7	8, chennai,(24), tamilnadu.	Ramu	15/11/2024	11/22/2024	\$1099	ontheway

## 10. Testing

### Testing Strategy:

1. **Unit Testing:** Focused on testing individual components and functions in isolation, ensuring that each part of the code behaves as expected. This was particularly important for smaller, reusable parts of the codebase, such as utility functions and API service functions.
2. **Integration Testing:** Conducted to verify that different modules and services interact correctly. This included tests for interactions between the frontend and backend, as well as interactions between different backend services.
3. **End-to-End (E2E) Testing:** Covered the entire workflow of the application, simulating a user's journey through the app to catch any issues that may arise in real scenarios. This involved testing core flows, like user signup/login, viewing books, ordering, and payment processes.

### Tools Used:

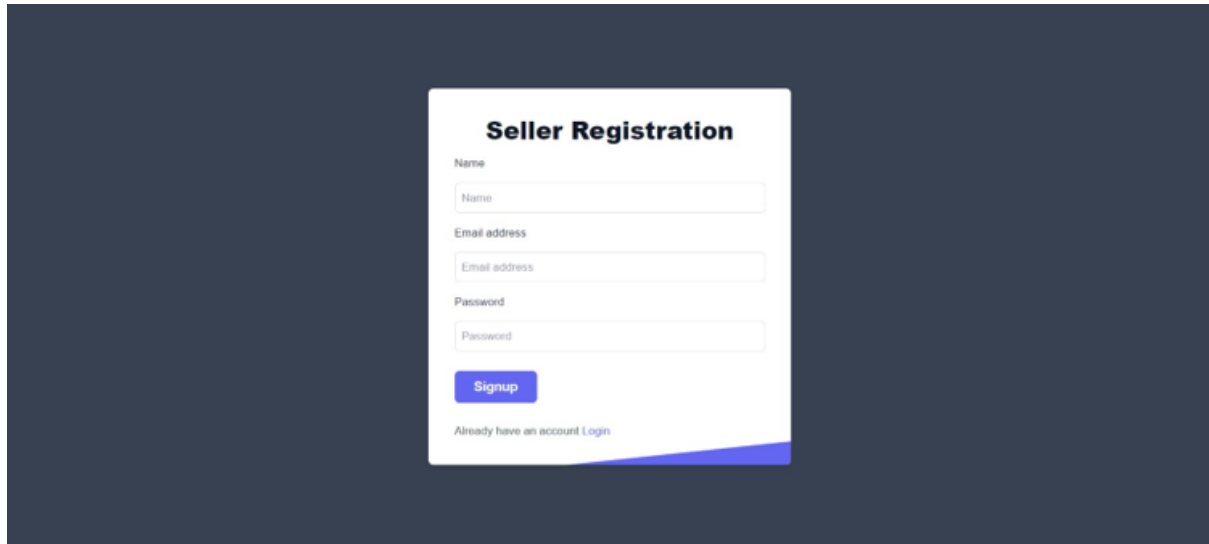
- **Jest:** For unit testing, primarily on the backend, due to its robust testing capabilities and support for mocking dependencies.
- **Mocha and Chai:** Used for both unit and integration testing, especially for the API layer, to ensure that REST endpoints are functioning correctly.
- **Selenium:** For end-to-end testing, as it allowed automated UI testing by simulating user actions on the frontend.
- **Postman:** Utilized for API testing, verifying endpoint functionality and ensuring the API responds as expected to different request scenarios.

### Continuous Integration (CI):

- **GitHub Actions:** Configured to run tests automatically on each push and pull request, ensuring code quality and preventing regressions.

## 11. Screenshots or Demo

**Demo link:** [https://drive.google.com/file/d/1-4S\\_EmINfj0TpAeeAZrMdyHye0AyELsz/view?usp=drivesdk](https://drive.google.com/file/d/1-4S_EmINfj0TpAeeAZrMdyHye0AyELsz/view?usp=drivesdk)



A screenshot of a web application showing a "Seller Registration" form. The form is centered on a dark blue background. It contains three input fields: "Name", "Email address", and "Password". Below the "Password" field is a blue "Signup" button. At the bottom of the form, there is a link that says "Already have an account Login".

**Seller Registration**

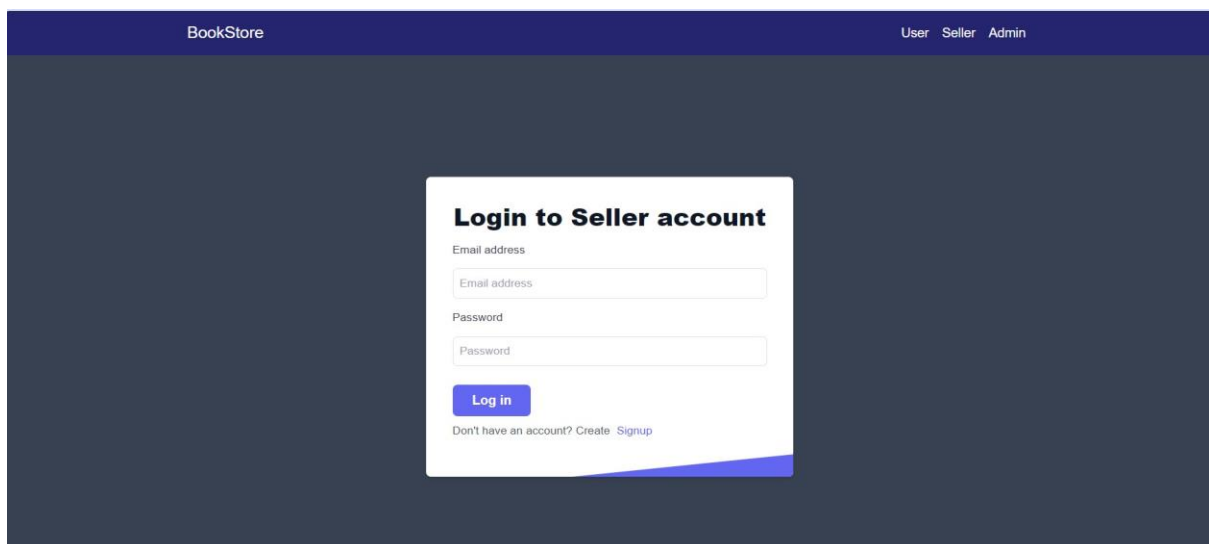
Name

Email address

Password

[Signup](#)

[Already have an account Login](#)



A screenshot of a web application showing a "Login to Seller account" form. The form is centered on a dark blue background. It contains two input fields: "Email address" and "Password". Below the "Password" field is a blue "Log in" button. At the bottom of the form, there is a link that says "Don't have an account? Create Signup". The top of the page has a dark blue header with the text "BookStore" on the left and "User Seller Admin" on the right.

BookStore User Seller Admin

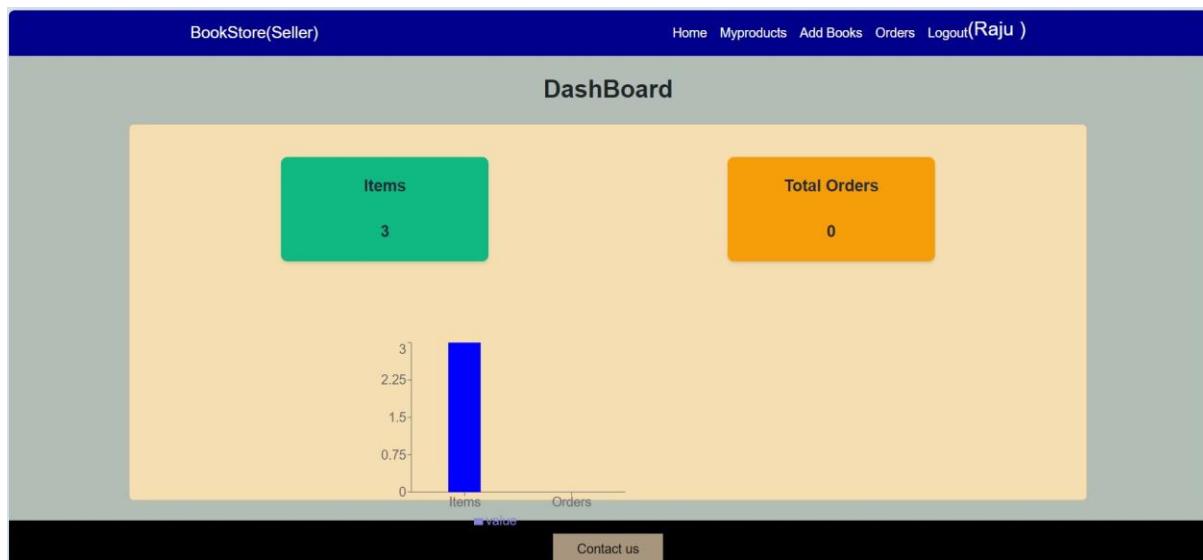
**Login to Seller account**

Email address

Password


[Log in](#)

[Don't have an account? Create Signup](#)

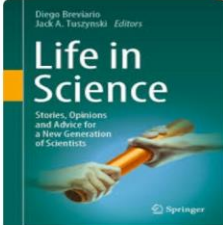


BookStore(Seller) Home Myproducts Add Books Orders Logout(Raju )

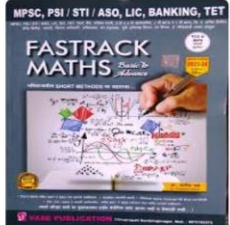
### Books List



**python Book**  
Author: Guido  
Genre: education  
Price: \$800  
Description:programming language  
...



**science book**  
Author: Ravikrishnan  
Genre: education  
Price: \$700  
Description:general science ...



**maths book**  
Author: Kannan  
Genre: education  
Price: \$800  
Description:general maths ...

BookStore(Seller)HomeMyproductsAdd BooksOrdersLogout(Raju )

### Add Book

Item Image

Choose File

No file chosen

Submit

## Admin Registration

Name

Email address

Password

Signup

Already have an account Login



BookStore

UserSellerAdmin

Loginto Admin account

Email address

Email address

Password

Password

Log in

Don't have an account? Create Signup



BookStore(Admin)

HomeUsersSellersLogout (Hemalatha )

Vendors

sl/no	UserId	User name	Email	Operation
1	6735f316cbb17ca4c2ef3436	hema	ac@123gmail.com	<div><div></div><div>view</div></div>
2	67375467a7565b2b20a0b242	Raju	Raju@123gmail.com	<div><div></div><div>view</div></div>
3	673754e0a7565b2b20a0b24b	Ramu	Ramu@gmail.com	<div><div></div><div>view</div></div>

## 12. Known Issues

### Data Persistence Delay in MongoDB

- **Description:** Occasionally, there is a delay in updating or retrieving data in MongoDB, which may result in users seeing outdated information or facing slower response times.
- **Workaround:** Refresh the page or retry the action. Investigate database performance and query optimization for a long-term solution.

### Intermittent Authentication Timeout

- **Description:** Sometimes, users may experience timeouts during authentication, particularly when the server load is high.
- **Impact:** This affects the login and signup processes, preventing users from accessing the application.
- **Workaround:** Increase the timeout threshold in the authentication service. Consider optimizing the backend to handle higher loads.

### API Gateway Routing Conflicts

- **Description:** Certain routes in the API gateway may conflict, leading to incorrect redirections, especially between the login and signup services.
- **Impact:** This can cause users to encounter unexpected errors when navigating between login and signup pages.
- **Workaround:** Review API gateway configurations and use more specific route paths.

### Session Expiration Not Consistently Handled

- **Description:** Session expiration may not prompt the user to re-login in certain cases, leading to unauthorized access errors.
- **Impact:** Users might experience interruptions when their session expires unexpectedly.
- **Workaround:** Implement a session timeout warning to prompt users to reauthenticate before the session expires.

## **13. Future Enhancements**

### **Enhanced Search and Filter Options**

- **Description:** Add more comprehensive search filters for users to find books by various criteria, such as genre, author, publication date, and ratings.
- **Impact:** Improves user experience by making it easier to navigate large collections of books and find specific items.

### **Recommendation System**

- **Description:** Implement a recommendation engine that suggests books based on users' past interactions, browsing history, and preferences.
- **Impact:** Provides a personalized experience, encouraging users to discover and engage with more content on the platform.

### **Rating and Review System**

- **Description:** Enable users to rate and review books, allowing for community engagement and helping other users make informed decisions.
- **Impact:** Builds a community aspect and provides valuable feedback for book curation.

### **Social Sharing Options**

- **Description:** Add social sharing options for users to share their favorite books or reviews on platforms like Facebook, Twitter, and Instagram.
- **Impact:** Enhances brand visibility and promotes user engagement by allowing users to share their experiences.