

# **BOOK STORE WITH MERN**

**NAAN MUDHALVAN**

**A Project Report**

*Submitted by*

<b>AYISHWARYA C</b>	<b>210921205008</b>
<b>GIFTA GRACE V</b>	<b>210921205015</b>
<b>JOICE ABINAIYA A</b>	<b>210921205024</b>
<b>SRINITHYA T</b>	<b>210921205051</b>
<b>SUJI S</b>	<b>210921205052</b>

*In partial fulfillment for the award of the degree*

*Of*

**BACHELOR OF TECHNOLOGY**

**In**

**INFORMATION TECHNOLOGY**



**LOYOLA INSTITUTE OF TECHNOLOGY, CHENNAI**

**ANNA UNIVERSITY:CHENNAI 600 025**

## **1. Project Overview:**

### **Purpose:**

The purpose of the Book-Store Application is to redefine the reading experience by merging traditional book exploration with modern digital convenience. This project seeks to address the needs of avid readers by providing a seamless, user-friendly platform that transforms how literature is discovered, accessed, and purchased.

Leveraging the powerful **MERN** (MongoDB, Express.js, React, Node.js) stack, the application delivers an efficient, engaging, and highly interactive browsing and purchasing experience. The goal is to connect book enthusiasts with their preferred genres and authors in a fast-paced digital age where time and convenience are paramount.

By offering features such as intuitive browsing, category-based searches, secure purchasing, and robust order tracking, this project aims to create a comprehensive online bookstore. Users can easily discover new titles, revisit timeless favorites, and purchase books anytime, anywhere.

The application combines scalable database capabilities, responsive server architecture, and an engaging front-end interface to bring a personalized and enriching literary experience to readers, eliminating the barriers of traditional bookstores and providing a future-forward platform for book discovery.

### **Features:**

The Book-Store Application offers a suite of robust features designed to cater to the modern reader's every need. Key features include a comprehensive book listing with detailed information about each title, such as author, genre, description, price, and availability. The platform also allows for advanced search and filtering options, enabling users to explore their favorite genres, authors, and popular books. User registration and authentication ensure secure access, while personalized user accounts allow users to manage their browsing, wish lists, and purchase history.

To further enhance the user experience, the application offers a seamless purchasing process. Users can add books to their cart, specify quantities, and securely complete their transactions, with inventory updates reflected in real-time. Order management capabilities

provide users with order confirmations, shipment tracking, and detailed order histories. With a responsive design and smooth navigation, the application ensures a delightful experience on any device, empowering readers to connect with literature in a way that fits their lifestyle and preferences.

## **2. Architecture Description for the Book-Store Application**

### **Frontend: React Architecture**

The frontend of the Book-Store Application leverages React to deliver a dynamic and highly responsive user experience. Key components of the frontend architecture include:

- **Component-Based Structure:** The frontend is built using reusable React components, such as **BookList**, **SearchBar**, **BookDetails**, **Cart**, **OrderHistory**, etc. This modular approach simplifies development, testing, and maintenance.
- **State Management:** The application uses React's state and context or state management libraries to manage user data, cart states, and book listings, ensuring a seamless flow of information across components.
- **Responsive Design:** The interface is designed to be responsive, adapting gracefully across different devices (desktop, tablet, mobile) for a consistent user experience. This is achieved through the use of CSS frameworks, media queries, and custom styling.
- **Routing:** The frontend utilizes React Router for managing page navigation. Users can navigate to different sections such as book listings, cart, and order history without page reloads, offering a smooth browsing experience.

### **Backend: Node.js and Express.js Architecture**

The backend forms the core of the application's server-side logic, built using Node.js and Express.js:

- **Express Server:** Express.js serves as the web framework to build the server logic, handling incoming HTTP requests, serving static files, and defining APIs.

- **RESTful APIs:** The backend exposes RESTful endpoints to perform CRUD operations (Create, Read, Update, Delete) on resources like users, books, orders, and inventory. For example:
  - **POST** /api/register for user registration.
  - **GET** /api/books to fetch a list of available books.
  - **POST** /api/orders for completing purchases.
- **Middleware:** Express middleware functions handle cross-cutting concerns such as request parsing, user authentication, error handling, and CORS (Cross-Origin Resource Sharing) settings.
- **Authentication and Security:** The backend uses token-based authentication (like JWT - JSON Web Token) to securely manage user sessions and permissions, ensuring that only authorized users can perform sensitive operations.

## Database: MongoDB Schema and Interactions

The database layer is built using MongoDB, offering flexibility and scalability to store the application's data:

- **Schema Design:** MongoDB documents are structured to represent collections such as **Users, Books, Orders**
  - **Users Collection:** Stores user profile data (username, email, hashed passwords, order history).
  - **Books Collection:** Contains book details like title, author, genre, price, description, and availability status.
  - **Orders Collection:** Records user orders, including order ID, user ID, book details, total price, order date, and status.
- **Database Operations:** Interactions with MongoDB are handled using Mongoose, providing a robust schema and model-based approach for data validation and querying.

## 3. Setup Instructions:

### Prerequisites:

Before you begin, ensure you have the following software installed on your local machine:

- **Node.js:** [Download Node.js](#)

- Required for running JavaScript code on the server side.
- **MongoDB:** [Download MongoDB](#)
  - Serves as the database for storing book and user information.
- **Git (optional):** [Download Git](#)
  - Used to clone repositories from version control services like GitHub.

## Installation:

Follow these steps to set up the Book-Store Application locally:

### 1. Clone the Repository:

Open a terminal or command prompt.

Navigate to the directory where you want to clone the project.

Run the following command:

bash

Copy code

```
git clone <repository_url>
```

Replace `<repository_url>` with the URL of your project repository.

### 2. Navigate to the Project Directory:

Change your directory to the cloned project folder:

bash

Copy code

```
cd <project-folder-name>
```

### 3. Install Dependencies:

The project typically has two main parts: the frontend (React) and the backend (Node.js + Express).

Navigate to the `backend` folder and run the following commands:

bash

Copy code

```
cd backend
```

```
npm run
```

This will install all necessary dependencies for the backend.

Then, navigate to the `frontend` folder and install frontend dependencies:

bash

Copy code

```
cd ../frontend
```

```
npm run
```

#### 4. Set Up Environment Variables:

Create an `.env` file in the `backend` directory.

Add the following environment variables (these values are examples and should be customized for your setup):

makefile

Copy code

```
PORT=4000
```

```
MONGO_URI=mongodb://localhost:27017/bookstoredb
```

```
JWT_SECRET=your_jwt_secret_key
```

You may need to add additional variables depending on your project configuration (e.g., API keys, frontend-specific settings).

#### 5. Run MongoDB Server (if applicable):

Ensure that your MongoDB service is running locally.

If you're using a local MongoDB server, you can start it with the following command (this may vary based on your OS):

bash

Copy code

```
mongod
```

#### 6. Start the Backend Server:

In the `backend` directory, run:

bash

Copy code

`npm run dev`

This command starts the backend server with hot reloading using `nodemon`.

## 7. Start the Frontend Application:

---

In the `frontend` directory, run:

bash

Copy code

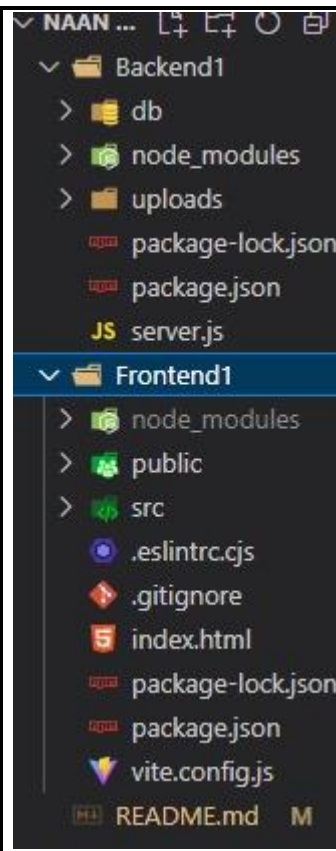
`npm start`

This will start the React development server.

## 8. Access the Application:

Open your browser and navigate to `http://localhost:3000` to view the application.

## 4.Folder Structure:



Here's an explanation of the folder structure based on your provided image, separating the client (frontend) and server (backend) structures:

### Client Structure (Frontend1):

- **Frontend1:** This folder contains the client-side code for your application, typically developed using frameworks like React. The key components are:
  - **node\_modules:** This folder stores all the dependencies and libraries installed for your React application. These dependencies are specified in `package.json`.
  - **public:** The `public` directory generally holds static assets like images, icons, and the main `index.html` file that serves as the entry point for the React app.
  - **src:** The `src` folder contains the source code of the React application, including components, pages, styles, hooks, context, and any other logic files. This is where the main application logic resides.
  - **.eslintrc.cjs:** This is an ESLint configuration file used for linting your JavaScript/React code to maintain consistent code quality and style.
  - **.gitignore:** This file specifies which files and directories should be ignored by Git version control, preventing them from being pushed to a repository.



- **index.html**: This is the main HTML file used by the React app, typically found in the **public** folder. It acts as the template to load the React code into the browser.
- **package.json**: This file contains metadata about the project, including dependencies, scripts, and other configuration settings for the React application.
- **package-lock.json**: This file locks the versions of dependencies and ensures a consistent installation across environments.
- **vite.config.js**: This is a configuration file for Vite, a build tool that serves as a faster alternative to Webpack for modern frontend projects. It specifies settings for building and serving the React application.

### Server Structure (Backend1):

- **Backend1**: This folder contains the server-side code for your application, usually developed using Node.js and Express.
  - **db**: This folder likely contains database-related configurations or scripts for connecting to the database (e.g., MongoDB).
  - **node\_modules**: This folder holds all the dependencies and libraries installed for the server-side application. These dependencies are defined in **package.json**.
  - **uploads**: This folder may store files uploaded by users through the server-side API (e.g., images, documents).
  - **package.json**: This file includes metadata about the server-side project, as well as dependencies, scripts, and configurations for the Node.js application.
  - **package-lock.json**: Similar to the frontend's version, this file locks dependency versions for consistent installation.
  - **server.js**: This file is the main entry point for the server-side application. It sets up and configures the Express.js server, handles routing, and connects to the database.

## 5. Running the Application:

To start the Book-Store Application locally, you will need to run both the frontend and backend servers. Follow the instructions below to get the application up and running:

### Step 1: Starting the Frontend Server

Navigate to the frontend directory: Open a terminal window and change the directory to the **Frontend1** folder:

```
bash
Copy code
cd Frontend1
```

1. Install dependencies (if not already installed): If you haven't installed the necessary dependencies, run the following command:

```
bash
Copy code
npm install
```

2. Start the frontend server: Use the following command to start the React development server:

```
bash
Copy code
npm start
```

This will typically run the React app on <http://localhost:3000> by default. You can open this URL in your browser to view the application.

## Step 2: Starting the Backend Server

Navigate to the backend directory: Open another terminal window (or a new tab) and change the directory to the **Backend1** folder:

```
bash
Copy code
cd Backend1
```

Install dependencies (if not already installed): Run the following command to install the required server-side dependencies:

```
bash
Copy code
npm install
```

Start the backend server: Use the following command to start the Node.js server:

```
bash
Copy code
npm start
```

By default, this will start the Express server, typically on <http://localhost:5000> (or another port if specified in your environment configuration).

## Additional Notes:

- Ensure that both servers are running simultaneously for full functionality.
- If there are any environment variables required, make sure to configure them appropriately (e.g., `.env` files).

## **6. API Documentation:**

The backend of the Book-Store Application exposes a set of RESTful endpoints that facilitate user interaction with the system. Below is a summary of the key endpoints, including request methods, parameters, and example responses.

### **1. User Authentication**

- **Endpoint:** `/api/users/register`
  - **Method:** `POST`
  - **Description:** Registers a new user.
  - **Parameters (JSON Body):**
    - `name` (string)
    - `email` (string)
    - `password` (string)

#### **Example Response:**

json

Copy code

```
{  
  
  "message": "User registered successfully",  
  
  "user": {  
  
    "id": "12345",  
  
    "name": "Sarah"  
  
  }  
  
}
```

- **Endpoint:** `/api/users/login`
  - **Method:** `POST`
  - **Description:** Logs in a user.
  - **Parameters** (JSON Body):
    - `email` (string)
    - `password` (string)

**Example Response:**

json

Copy code

```
{  
  
  "token": "jwt-token-here",  
  
  "user": {  
  
    "id": "12345",  
  
    "name": "Sarah"  
  
  }  
}
```

## 2. Books Management

- **Endpoint:** `/api/books`
  - **Method:** `GET`
  - **Description:** Fetches all available books.

**Example Response:**

json

Copy code

```
[  
  
  {  
  
    "id": "book123",  
  
    "title": "Book Title",
```

```
"author": "Author Name",  
  
"genre": "Fiction",  
  
"price": 19.99  
  
}  
  
]
```

- **Endpoint:** `/api/books/:id`
  - **Method:** `GET`
  - **Description:** Fetches details of a specific book by ID.
  - **Parameters:**
    - `id` (string, path parameter)

#### Example Response:

json

Copy code

```
{  
  
  "id": "book123",  
  
  "title": "Book Title",  
  
  "author": "Author Name",  
  
  "description": "A brief summary...",  
  
  "price": 19.99,  
  
  "availability": true  
  
}
```

### 3. Order Management

- **Endpoint:** `/api/orders`
  - **Method:** `POST`
  - **Description:** Places a new order.

- **Parameters** (JSON Body):
  - **userId** (string)
  - **items** (array of book IDs)

**Example Response:**

json

Copy code

```
{  
  
  "message": "Order placed successfully",  
  
  "order": {  
  
    "id": "order123",  
  
    "totalPrice": 49.99,  
  
    "status": "Pending"  
  }  
}
```

- **Endpoint:** `/api/orders/:userId`
  - **Method:** `GET`
  - **Description:** Retrieves order history for a specific user.
  - **Parameters:**
    - **userId** (string, path parameter)

**Example Response:**

json

Copy code

```
[  
  
  {  
  
    "orderId": "order123",  
  
    "items": ["book123", "book456"],  
  
    "totalPrice": 49.99,  
  
  }  
]
```

```
"status": "Completed"
```

```
}
```

```
]
```

## **7. Authentication:**

### **Authentication Mechanisms:**

Authentication ensures that the person or entity accessing the system is who they claim to be. Common mechanisms include:

1. **Username and Password:** This basic form of authentication requires users, sellers, and admins to log in using their credentials (e.g., email and password).
2. **Session Management:** After successful login, a secure session (e.g., session token or JWT - JSON Web Token) is created to maintain the state of the logged-in user.

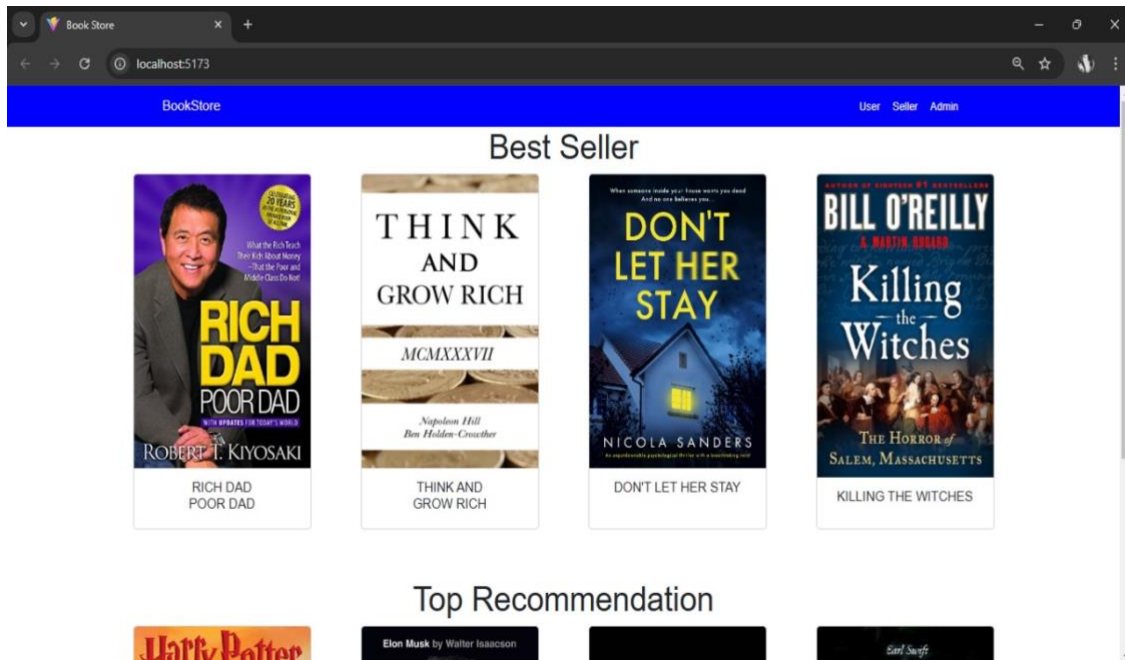
### **Authorization Mechanisms:**

Authorization defines what a user can do after being authenticated based on their assigned role (user, seller, or admin). Here's how it can work for BookEase:

1. **Role-Based Access Control (RBAC):**
  - **Users** have permissions related to browsing, purchasing books, managing their profiles, and providing feedback.
  - **Sellers** are authorized to list, update, and manage their books, along with order fulfillment functionalities.
  - **Admins** are authorized to manage users, sellers, books, and perform other system-level functions.
2. **Access Control Lists (ACLs):** This can specify fine-grained permissions for specific actions, such as allowing only admins to delete books or only sellers to update inventory.

## **8. User Interface:**

## 1. Landing page:



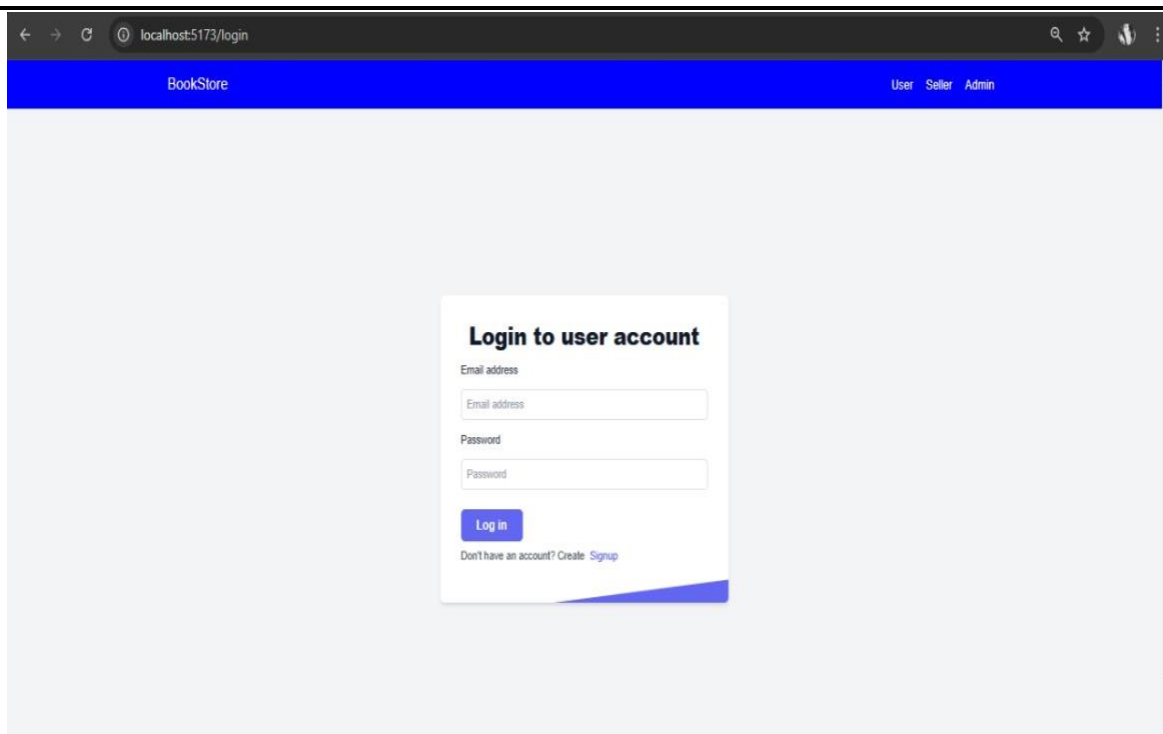
## 2. User [User interface]:

### User Signup page:

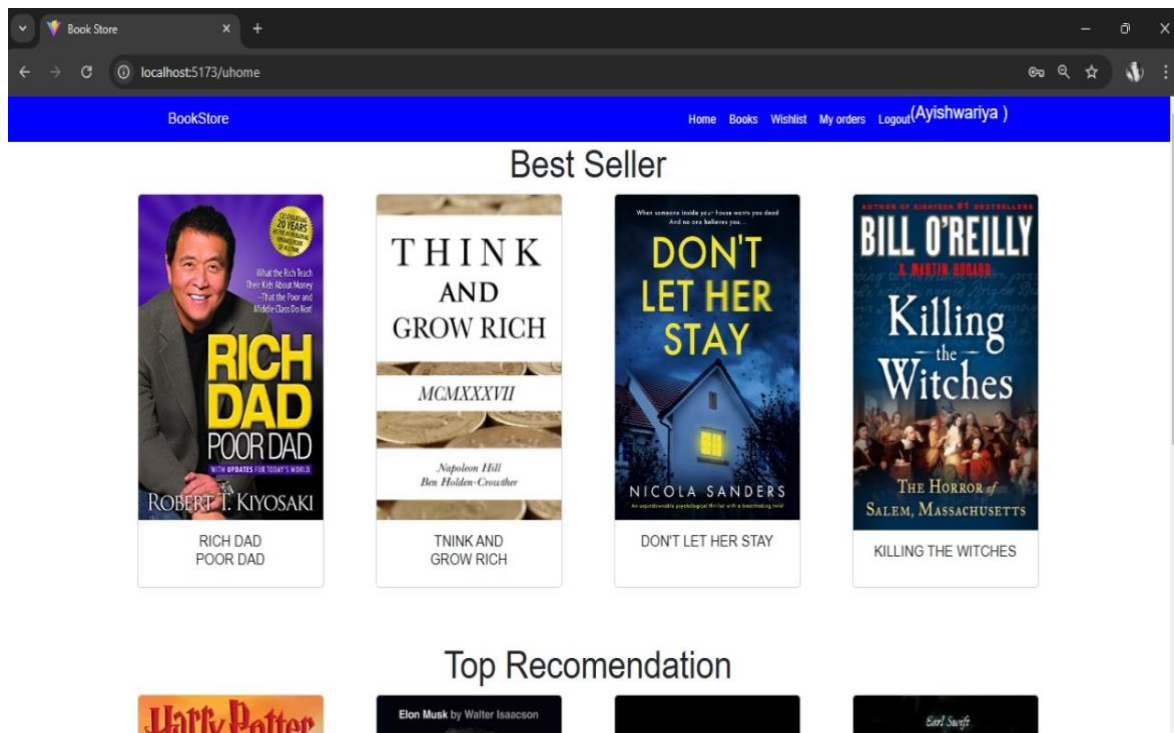
The screenshot shows a web browser window with the address bar displaying 'localhost:5173/signup'. The page has a light gray background. In the center, there is a white box with a blue header containing the text 'Signup'. Below the header, there are three input fields: 'Name', 'Email address', and 'Password'. Below the 'Password' field, there is a blue button with the text 'Signup'. At the bottom of the box, there is a link that says 'Already have an account Login'.

### User Login page:





## User Home Page:

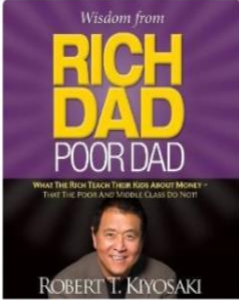


## User Book List:

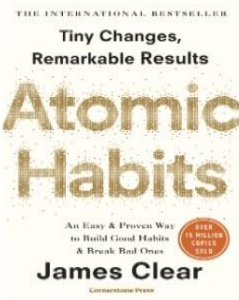
localhost:5173/uproducts

BookStoreHomeBooksWishlistMy ordersLogout(Ayishwariya)

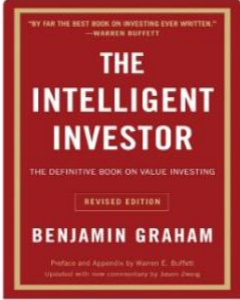
Books List




**Rich Dad Poor Dad.**  
Author: Robert T. Kiyosaki  
Genre: Financial  
Price: \$350  
[Add to Wishlist](#) [View](#)



**Atomic Habits**  
Author: James Clear  
Genre: Habits  
Price: \$850  
[Add to Wishlist](#) [View](#)



**The Intelligent Investor**  
Author: Benjamin Graham  
Genre: Financial  
Price: \$750  
[Add to Wishlist](#) [View](#)



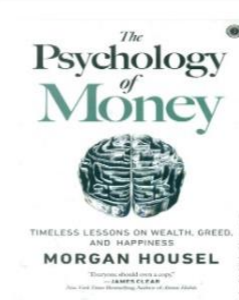
**The Psychology of Money**

## User WishList:

localhost:5173/wishlist

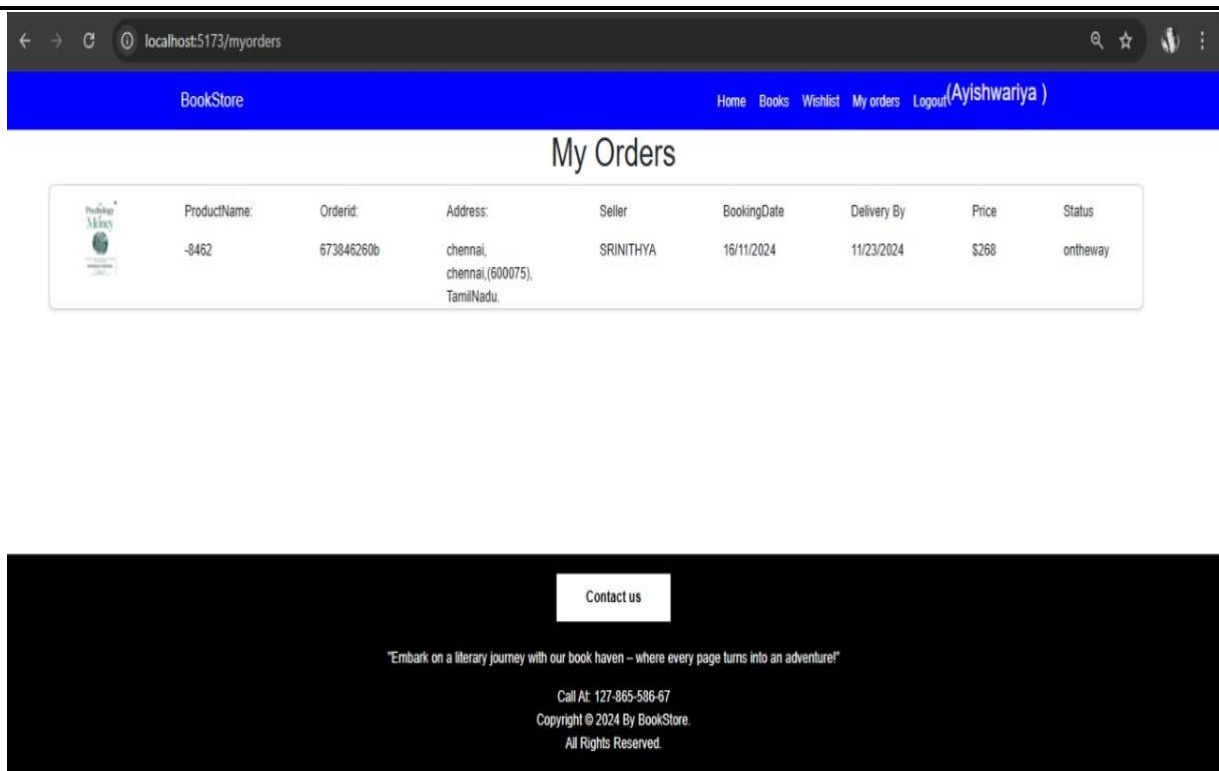
BookStoreHomeBooksWishlistMy ordersLogout(Ayishwariya)

Wishlist



**Psychology of Money**  
Author:  
Genre:  
Price: \$  
[Remove from Wishlist](#) [View](#)

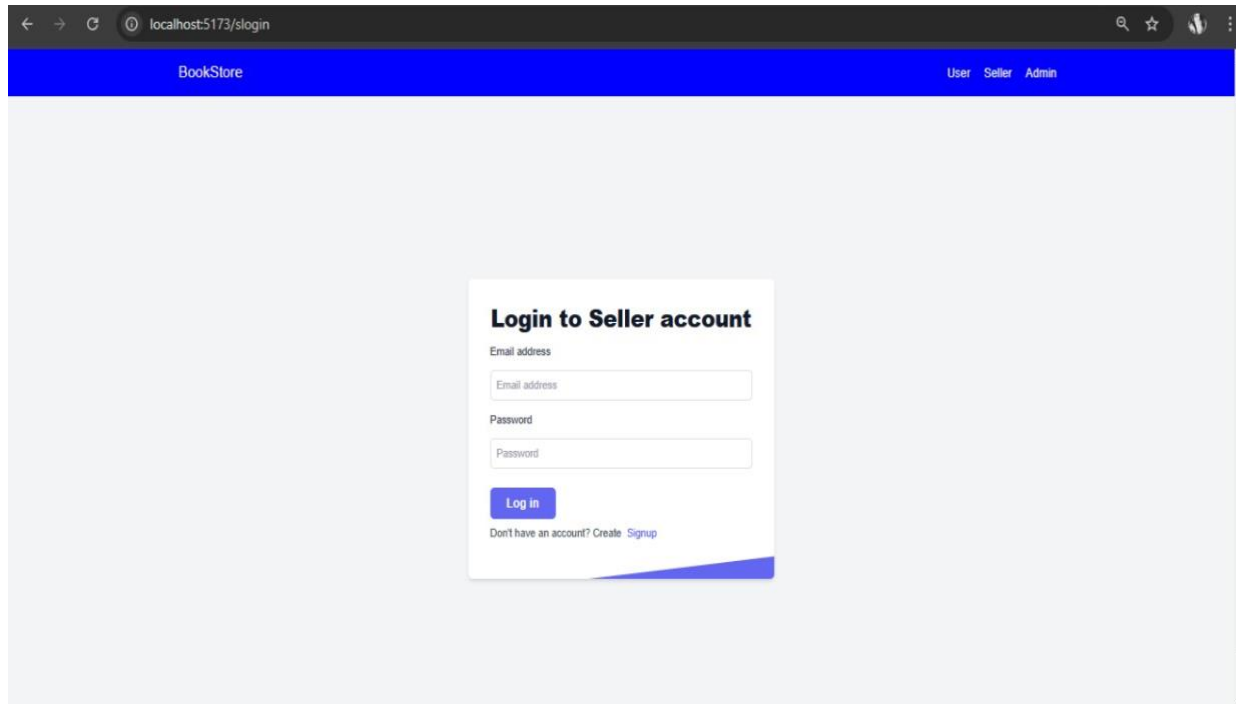
## User [My Orders]:



### 3. Seller[User Interface]:

#### Seller Signup page:

## Seller Login Page:



The screenshot shows a web browser at localhost:5173/login. The page has a blue header with 'BookStore' on the left and 'User Seller Admin' on the right. The main content area is light gray and contains a white login form titled 'Login to Seller account'. The form has two input fields: 'Email address' and 'Password'. Below the fields is a blue 'Log in' button. At the bottom of the form, it says 'Don't have an account? Create Signup'.

BookStore

User Seller Admin

### Login to Seller account

Email address

Password

Log in

Don't have an account? Create Signup

## Seller HomePage:



## Seller Products:

← → ↺

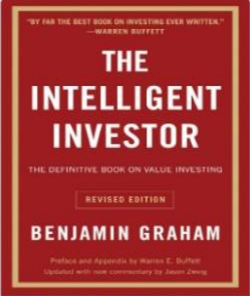
localhost:5173/myproducts

🔍 ☆ 👤 ⋮

BookStore(Seller)

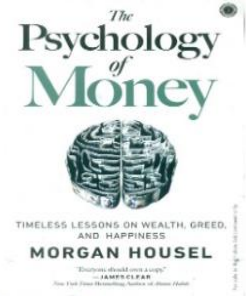
Home Myproducts Add Books Orders Logout(SRINITHYA)

Books List



**The Intelligent Investor**  
THE DEFINITIVE BOOK ON VALUE INVESTING  
REVISED EDITION  
**BENJAMIN GRAHAM**  
Preface and Appendix by Warren E. Buffett  
Updated with new commentary and fresh data

**The Intelligent Investor**  
Author: Benjamin Graham  
Genre: Financial  
Price: \$750  
Description: value investing ...



*The*  
**Psychology of Money**  
TIMELESS LESSONS ON WEALTH, GREED, AND HAPPINESS  
**MORGAN HOUSEL**  
"Everything should come a little bit."  
— SAMUEL ALTMAN  
New York Times Bestselling Author of The Psychology of Money

**Psychology of Money**  
Author: Morgan Housel  
Genre: Money  
Price: \$169  
Description: Collection of short stories exploring the strange ways people think about money. ...

## Seller [Add books]:

localhost:5173/addbook

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

### Add Furniture

title

author

genre

Price


Description

Item Image

Choose File No file chosen

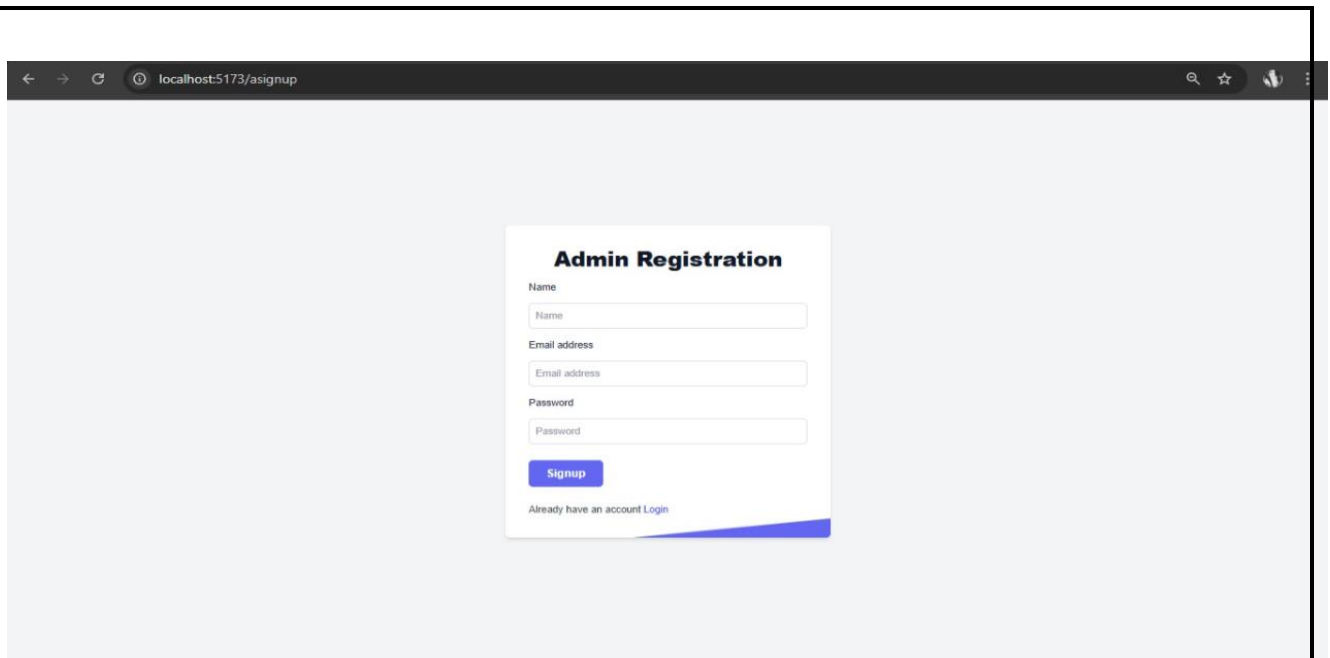
Submit

## Seller Orders:

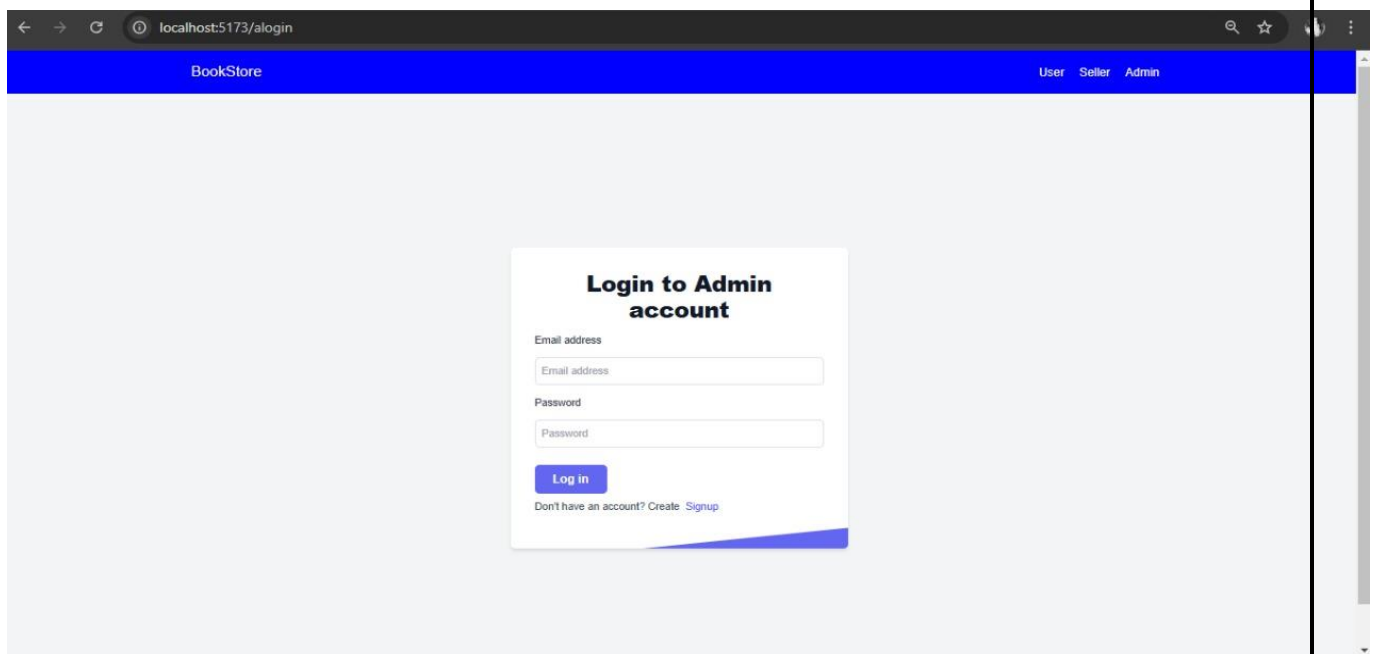
BookStore(Seller) Home Myproducts Add Books Orders Logout(SRINITHYA)									
Orders									
	ProductName:	Orderid:	Customer Name	Address:	BookingDate	Delivery By	Warranty	Price	Status
		673846260b	Ayishwariya	chennai, chennai,(600075), TamilNadu.	16/11/2024	11/23/2024	1 year	268	ontheway

## 4.Admin[User Interface]:

### Admin Signup page:

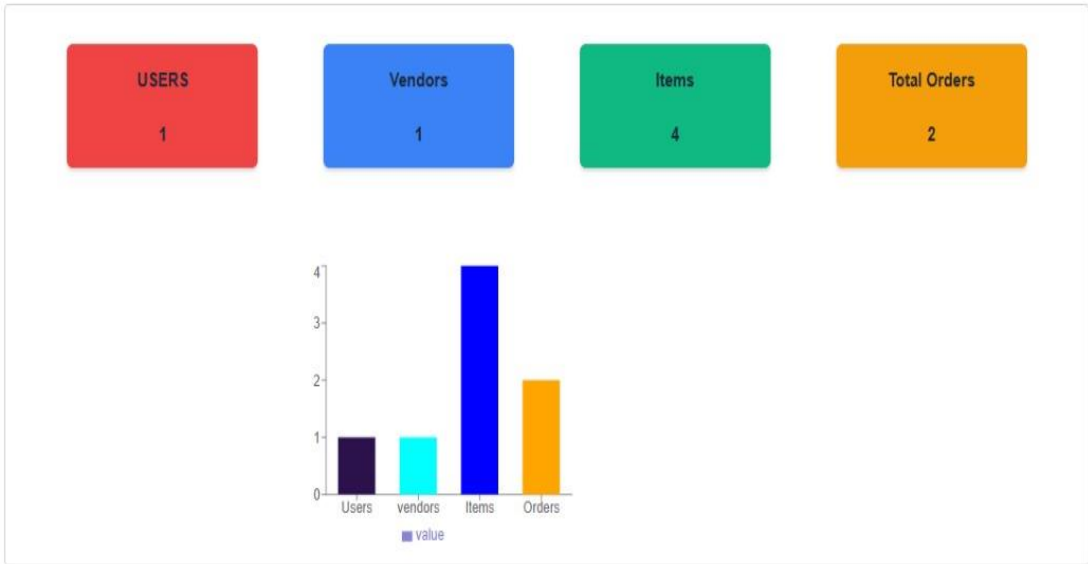


## Admin Login page:



## Admin Dashboard:

DashBoard



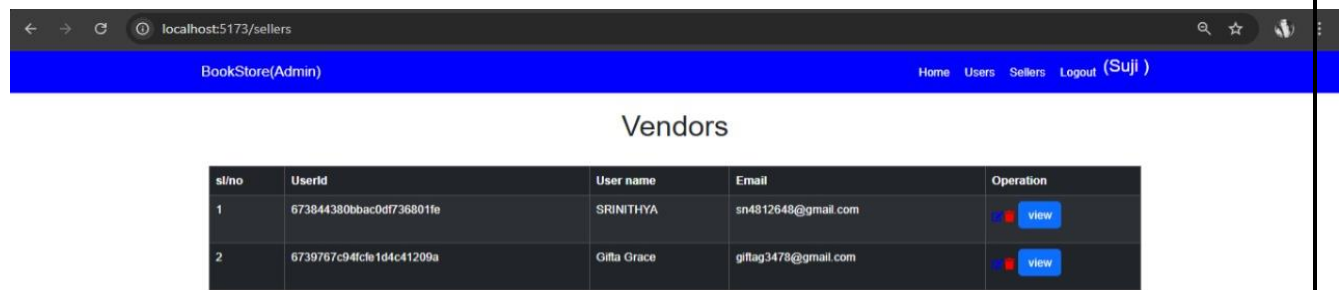
Admin User:



Users

sl/no	Userid	User name	Email	Operation
1	673845de0bbac0df7368023e	Ayishwariya	ayishwariya@gmail.com	<div>🔍</div> <div>view</div>
2	6739764194fde1d4c412097	Joice abinaya	joice1275@gmail.com	<div>🔍</div> <div>view</div>



## Admin Vendors:



sl/no	Userid	User name	Email	Operation
1	673844380bbac0df736801fe	SRINITHYA	sn4812648@gmail.com	 <a href="#">view</a>
2	6739767c94f1c1e1d4c41209a	Gita Grace	gitag3478@gmail.com	 <a href="#">view</a>

## 9. Testing:

### 1. Unit Testing:

- Tools: Jest, React Testing Library (for React), Mocha, Chai (for Node.js).
- Purpose: Test individual functions/components.

### 2. Integration Testing:

- Tools: Supertest, Jest, Mocha, Chai.
- Purpose: Test interactions between server, APIs, and database.

### 3. End-to-End (E2E) Testing:

- Tools: Cypress, Puppeteer, Selenium.
- Purpose: Simulate real user interactions and test full flow.

### 4. Test Coverage:

- Tools: Jest, Istanbul.
- Purpose: Ensure all code is tested.

### 5. CI Testing:

- Tools: GitHub Actions, Travis CI, CircleCI.
- Purpose: Automate tests on code changes.

#### 6. Performance Testing:

- Tools: Lighthouse, Artillery.
- Purpose: Test app speed and scalability.

#### 7. Mocking/Stubbing:

- Tools: Sinon.js, Mock Service Worker (MSW).
- Purpose: Mock APIs and database calls for isolated testing.

### 10. Screenshot or Demo:

**Here is our Project Demo link:**

<https://www.kapwing.com/videos/6739a4fadafb76a17c6d6585>

### 11. Known Issues:

#### 1. Slow Loading on High Traffic:

- **Description:** During periods of high traffic, the application may experience slow page load times, especially on the product listing page.
- **Workaround:** We are working on optimizing database queries and implementing caching mechanisms to improve performance.

#### 2. Admin Dashboard Rendering Issues:

- **Description:** Occasionally, the Admin Dashboard may fail to render certain sections correctly on older browsers or with low screen resolution.
- **Workaround:** We recommend using the latest versions of Chrome, Firefox, or Edge for the best experience.

#### 3. Cross-Browser Compatibility:

- **Description:** Some visual elements, such as buttons and fonts, may not render as expected on Internet Explorer and older versions of Safari.

- **Workaround:** We suggest using modern browsers for optimal performance and layout consistency.

#### 4. **Mobile Responsiveness:**

- **Description:** On smaller mobile screens (less than 320px wide), the product images and text may overlap, affecting the layout.
- **Workaround:** A fix is in progress to improve mobile responsiveness on all devices.

## **12.Future Enhancements:**

### **Potential Future Enhancements for the BookEase Book Store Web Application**

To further improve and expand the functionality of the BookEase web application, here is an outline of potential future features and implementations:

#### **1. User Enhancements:**

**Enhanced Search Filters:** Introduce advanced search filters, such as publication date, language, publisher, and ratings, to improve book search functionality.

**Book Recommendations:** Implement a personalized recommendation engine based on user browsing and purchase history.

**Order Tracking:** Provide real-time tracking updates for orders, showing estimated delivery times and order status.

**Multiple Payment Options:** Offer additional payment methods such as digital wallets, UPI, and international cards.

**Social Sharing:** Allow users to share book details or reviews on social media platforms.

**User Dashboard:** Introduce a user dashboard that provides insights into past purchases, reviews given, and personalized deals.

#### **2. Seller Enhancements**

**Bulk Uploads for Book Listings:** Enable sellers to upload multiple book details using spreadsheets or bulk-upload tools.

**Sales Analytics Dashboard:** Provide sellers with an analytics dashboard to track sales performance, top-selling books, and user feedback trends.

**Promotional Tools:** Allow sellers to create and manage promotions or discounts for specific books.

**Inventory Alerts:** Send notifications when stock levels are low or when books are back in stock.

**Messaging System:** Introduce a built-in messaging system for direct communication between sellers and users for queries.

### **3. Admin Enhancements**

**Role-Based Access Control (RBAC):** Implement role-based access control for admin functionalities, allowing different levels of admin permissions (e.g., super admin, moderator).

**Automated Fraud Detection:** Introduce algorithms to detect and flag suspicious activity, such as fraudulent orders or fake reviews.

**Detailed Reporting:** Offer comprehensive reports on system performance, user engagement, and seller activity.

**Content Moderation Tools:** Provide tools to moderate user reviews and feedback, ensuring a healthy and trustworthy user environment.

**Support Ticket System:** Add a support ticketing system for users and sellers to raise issues that admins can resolve.

### **4. Security Enhancements**

**Two-Factor Authentication (2FA):** Enable 2FA for users, sellers, and admins to enhance account security.

**Data Encryption:** Ensure sensitive data, such as passwords and payment information, is encrypted at all stages.

**Regular Security Audits:** Schedule periodic security checks and updates to maintain system integrity.

### **5. User Engagement Features**

**Community Forums:** Introduce forums where users can discuss books, share recommendations, and interact with like-minded readers.

**Gamification:** Implement gamification elements, such as badges, points, or rewards, for actions like leaving reviews, making purchases, or referring new users.

**Referral Program:** Create a referral program to incentivize existing users to bring new users to the platform.

---