BOOK NEST

# 1. Introduction

Project Title: BookNest – Online Bookstore

Welcome to the literary haven of the digital age-introducing our revolutionary Book Nest Application, a masterpiece crafted with precision using the powerful MERN (MongoDB, Express.js, React, Node.js) Stack. Immerse yourself in a world where the love for reading converges seamlessly with cutting-edge technology, redefining the way bibliophiles explore, discover, and indulge in their literary pursuits.

Tailored for the modern book enthusiast, our MERN-based Book-Store Application seamlessly blends robust functionality with an intuitive user interface. From the joy of discovering new releases to the nostalgia of revisiting timeless classics, our platform promises an immersive reading experience customized to cater to your literary preferences.

Team Members:

1. Badavath Sai Karthik Nehru
2. Battu Ramarao
3. Balagam Pavansai
4. Balla Bhanu Akash

# 2. Project Overview

## Purpose:

BookNest is a full-stack e-commerce platform for books. It allows users to browse, purchase, and manage books while providing sellers a dashboard to upload and manage their listings.

## Key Features:

- User & Seller registration/login  
- Book listing with image URLs  
- Shopping cart & order history  
- Favorite books management  
- Seller dashboard to manage inventory

* **ER diagram**

**A diagram of a network

AI-generated content may be incorrect.**

**User-Book Relationship:**

Type: Many-to-Many (M:M). A single user can read or interact with many books, and a single book can be accessed by many users.

Implementation: Introduce an intermediate entity, "Interaction", with foreign keys to both User and Book tables. This table could store additional information like reading progress, reviews, or ratings.

**Book-Inventory Relationship:**

Type: One-to-Many (1:M). Each book can have multiple copies in inventory, but each copy belongs to one book.

Implementation: Maintain a separate Inventory table with fields like BookID (foreign key), quantity, location, and condition.

**User-Order Relationship:**

Type: One-to-Many (1:M). A single user can place multiple orders, but each order belongs to one user.Implementation: Keep the UserID foreign key in the Order table to track user purchase history.

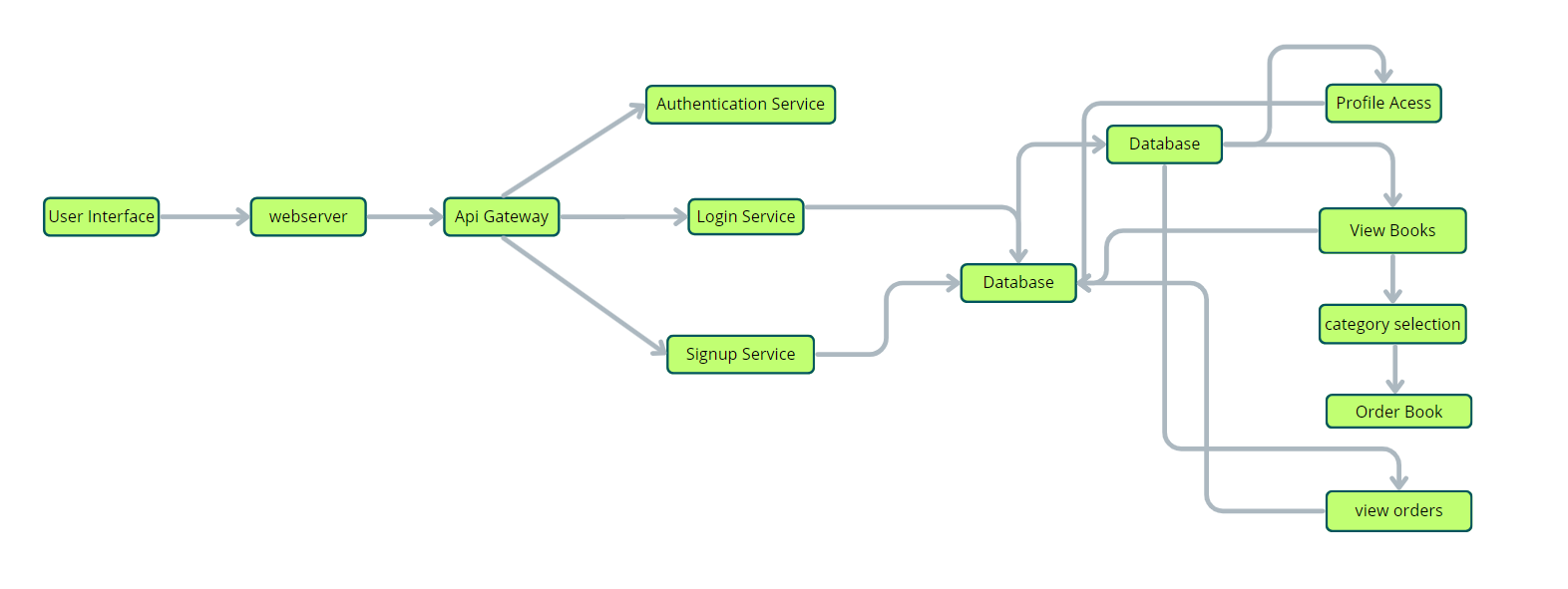
**Additional Relationships:**

Book-Author Relationship: Many-to-Many (M:M). A book can have multiple authors, and an author can write multiple books. (Similar to User-Book, use an intermediate "WrittenBy" table)

Book-Genre Relationship: Many-to-Many (M:M). A book can belong to multiple genres, and a genre can have many books. (Similar to User-Book, use an intermediate "CategorizedAs" table)

Review-User Relationship: Many-to-One (M:1). A review is written by one user, but a user can write many reviews. (Keep UserID as a foreign key in the Review table)

# 3. Architecture



## Frontend (React):

- React.js with React Router for navigation  
- State managed using hooks  
- REST API integration using fetch

## Backend (Node.js & Express.js):

- RESTful API built with Express  
- Authentication handled with JWT  
- Middleware for route protection

## Database (MongoDB):

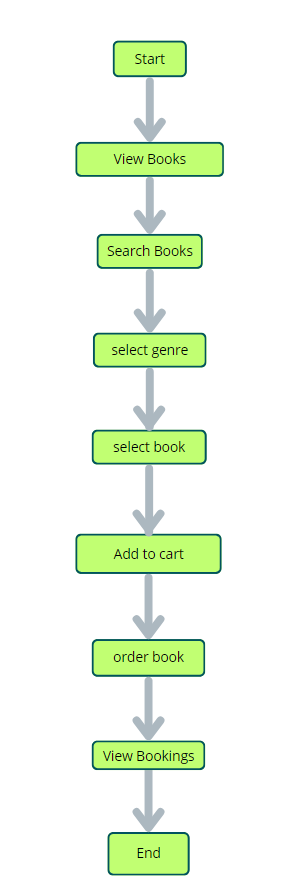
- Collections: Users, Sellers, Books, Orders  
- Mongoose models for schema definition and interaction

# 4. Setup Instructions

## Prerequisites:

- Node.js ≥ v16  
- MongoDB (local or cloud)  
- npm

* **User Flow:**

****

# 5. Folder Structure

## Client (React):

frontend/  
├── src/  
│ ├── App.jsx  
│ ├── Home.jsx  
│ ├── User/  
│ ├── Seller/  
│

## Server (Node.js/Express):

backend/

├── uploads/  
├── middleware/  
├── routes/  
│ ├── users.js  
│ ├── seller.js  
│ ├── books.js  
│ └── auth.js  
├── models/  
│ ├── User.js  
│ ├── Seller.js  
│ └── Book.js

├── server.js

# 6.Running the Application

Backend:  
cd backend

npx nodemon server.js  
  
Frontend:  
cd frontend  
npm run dev

# 7.API Documentation

|  |
| --- |
| Endpoint | | Method | Description |
| | /api/users/signup POST | Register new user | | /api/seller/signup POST | Register new seller | | /api/user/login POST | Login user | | /api/seller/login POST | Login seller | | /api/books GET | Get all books | | /api/books/seller/:sellerId GET | Get books by seller | | /api/books POST | Add a new book | |

# 8. Authentication

- Login routes issue JWTs on successful login  
- Token stored in localStorage on frontend  
- Protected routes check for token using middleware  
- Separate flows for users and sellers

# 9. User Interface

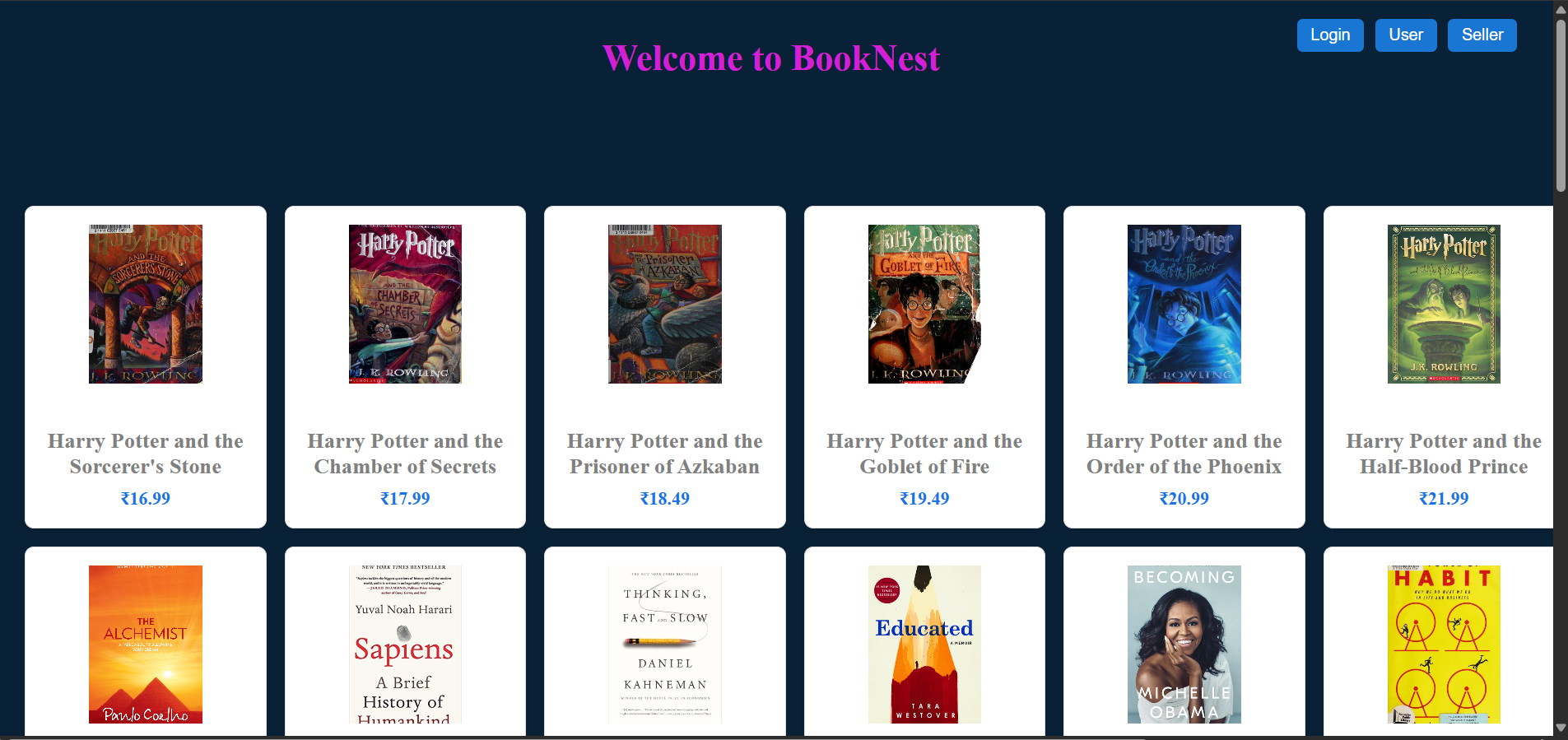
- Home Page with login/signup options  
- User Dashboard: View books, cart, favorites  
- Seller Dashboard: Upload books, view listings

# 10. Testing

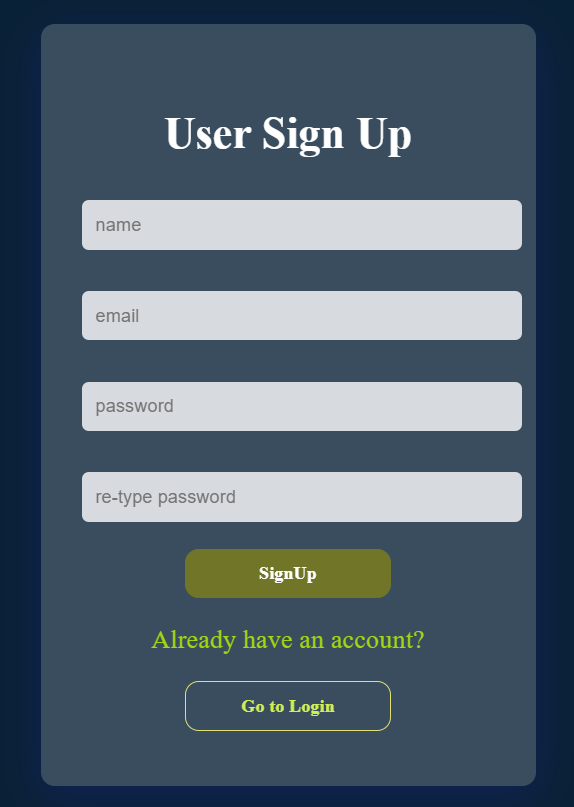
- Manual testing using Postman for backend routes  
- Browser-based testing for frontend features  
- Future plans for automated unit testing with Jest

# 11. Screenshots / Demo

Landing page:-



User Signup page :-

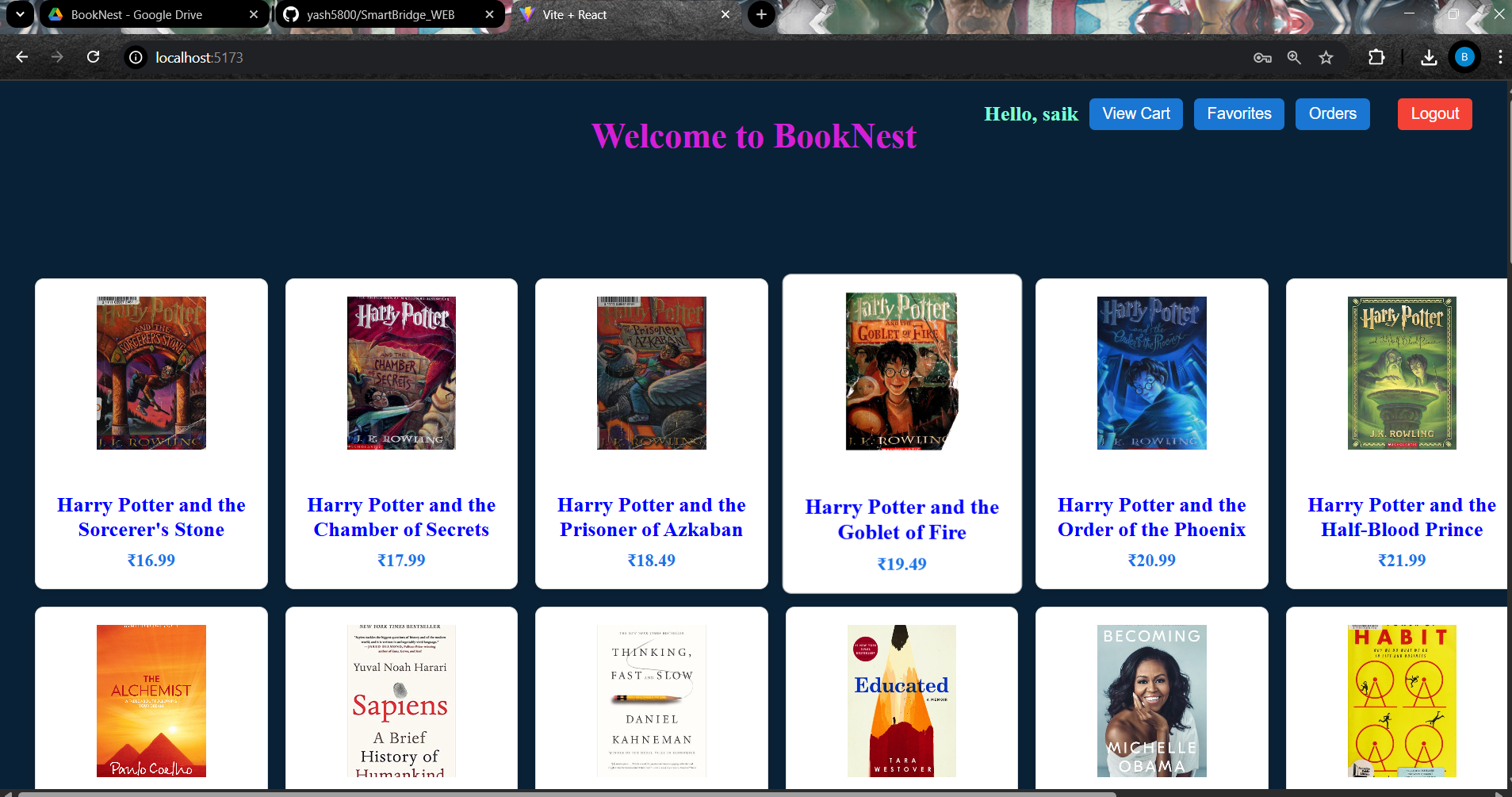


Login Page:-

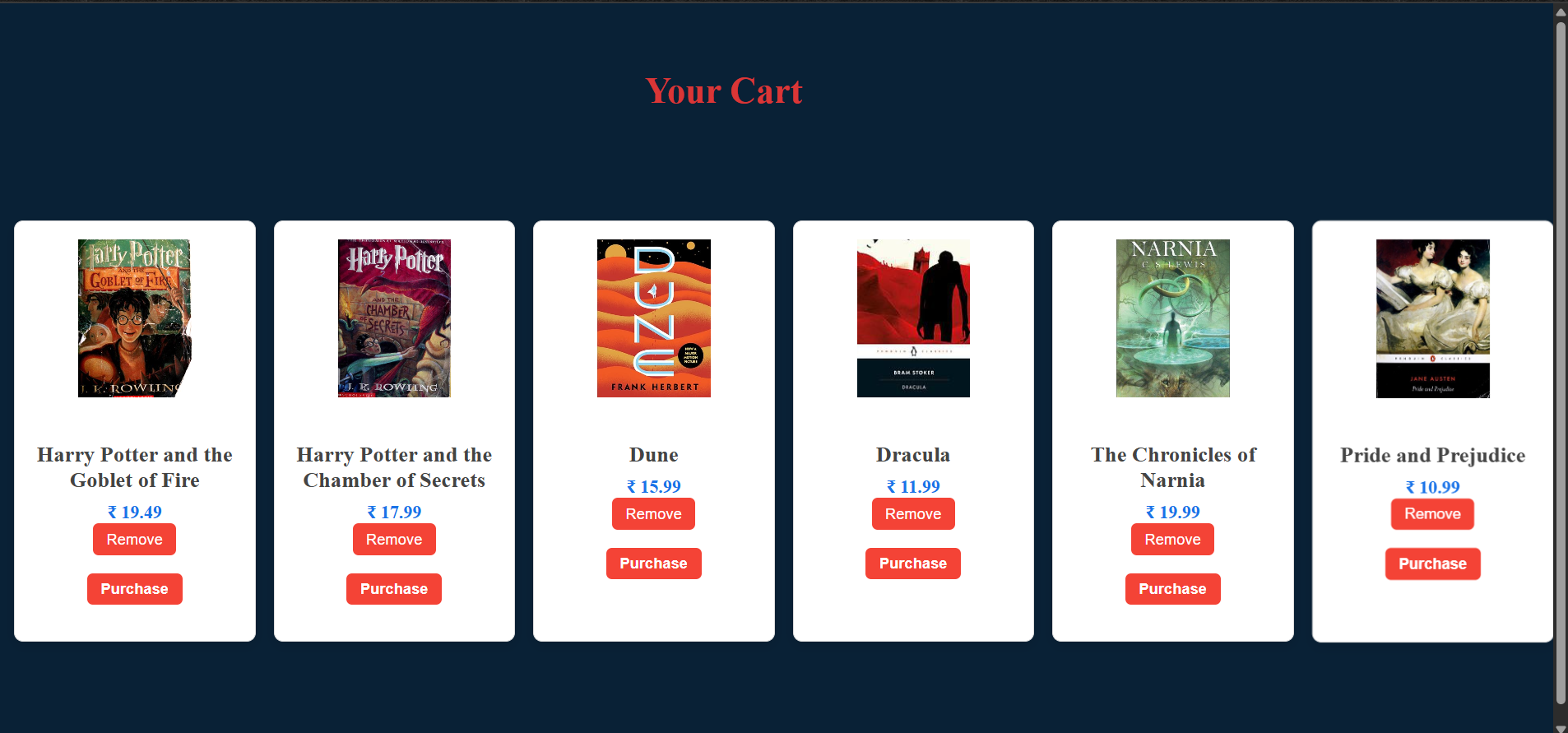
A screenshot of a computer

AI-generated content may be incorrect.

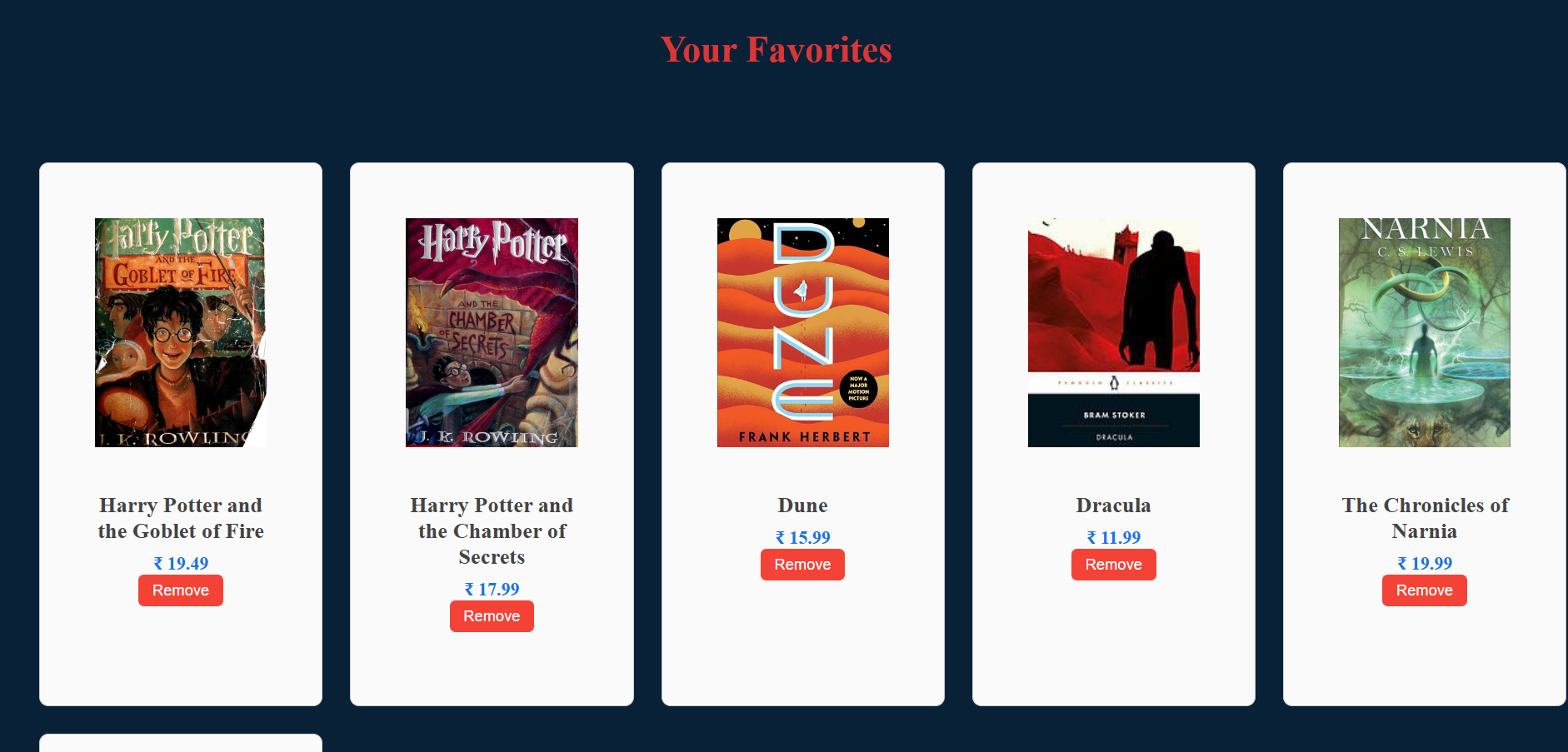
Home Page:-



Books Page:-



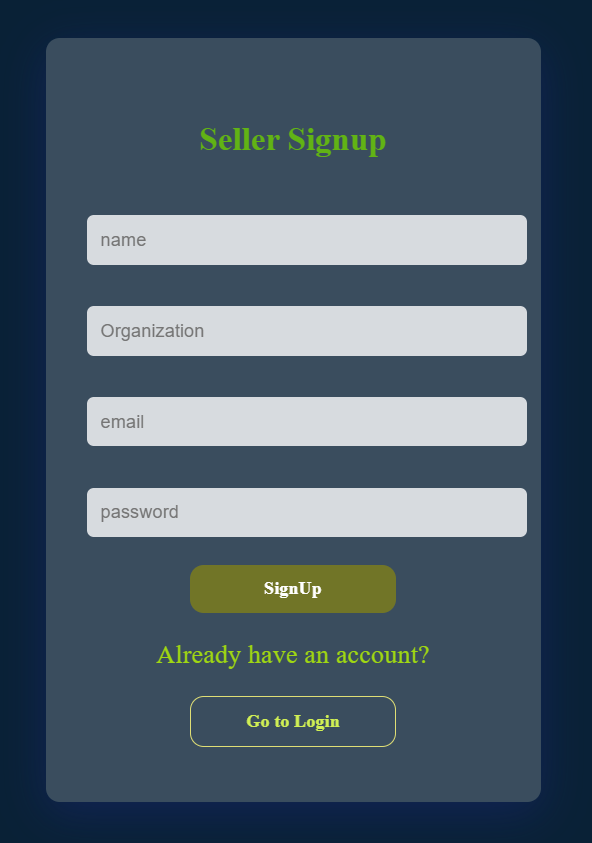
# Favorites Page:-



My Bookings Page :-



Seller Signup page :-

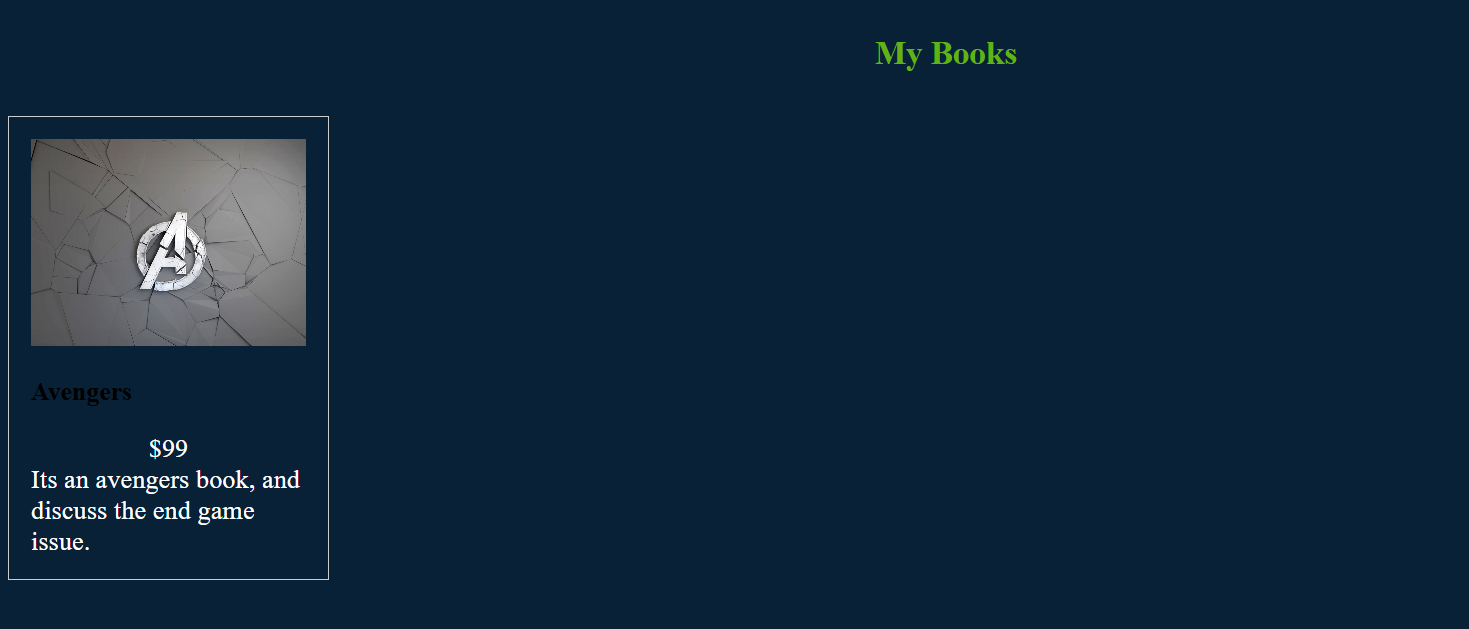


Seller Dashboard:-

A blue square with white lines

AI-generated content may be incorrect.

Seller Items:-



# 12. Known Issues

- Cart state doesn't persist on refresh  
- Book image validation is minimal  
- No password encryption yet

# 13. Future Enhancements

- Password hashing (bcrypt)  
- Add image upload support (Cloudinary)  
- Add admin panel  
- Improve UI/UX with animations and toasts  
- Search and filter functionality

# 14. Git hub link

[***https://github.com/Bskn1412/book-nest.git***](https://github.com/Bskn1412/book-nest.git)