BookNest: Where Stories Nestle - Project Documentation

1. Introduction

- **Project Title**: BookNest: Where Stories Nestle

Team Members:

- Gobburi Sreevidya: Project Manager
- G Upendra Chowdary: Frontend Developer
- Chimala Dinesh: Backend Developer

2. Project Overview

- **Purpose**: BookNest is a MERN (MongoDB, Express.js, React, Node.js) stack-based online bookstore designed to provide a seamless, intuitive platform for book enthusiasts to discover, browse, and purchase books. It addresses the needs of users like Sarah, who seek a convenient way to explore and buy books without compromising the joy of bookstore browsing. The platform combines robust functionality with a user-friendly interface, offering a tailored literary experience.
- **Features**:
    - **User Registration and Authentication**: Secure account creation and login with JWT-based authentication.
    - **Book Listings**: Comprehensive display of books with details like title, author, genre, description, price, and availability.
    - **Book Selection**: Filters for genre, author, ratings, and popularity to refine browsing.
    - **Purchase Process**: Add books to cart, specify quantities, and complete secure purchases with inventory updates.
    - **Order Confirmation**: Detailed order confirmation with book details, total price, and order ID.
    - **Order History**: View past and current orders with tracking and review options.
    - **Responsive UI**: Consistent experience across desktops, tablets, and smartphones.

3. Architecture

- **Frontend**: Built with React, utilizing React Router for navigation, Redux for state management, and Tailwind CSS for styling. Components are modular, ensuring a dynamic and responsive interface for browsing, cart management, and order tracking.
- **Backend**: Powered by Node.js and Express.js, the backend handles API requests, business logic, and server-side operations. It uses middleware for authentication (JWT), error handling, and request validation, ensuring efficient communication with MongoDB.
- **Database**: MongoDB, a NoSQL database, stores data in collections for users, books, orders, and categories. The schema is designed for scalability, with indexes on frequently queried fields (e.g., book title, genre) to optimize performance.

4. Setup Instructions

- **Prerequisites**:

- Node.js (v16 or higher)
- MongoDB (local or MongoDB Atlas)
- npm or yarn
- Git

- **Installation**:

0. Clone the repository: git clone https://github.com/upendra042/book.git
1. Navigate to the project directory: cd book
2. Install client dependencies: cd client && npm install
3. Install server dependencies: cd ../server && npm install
4. Set up environment variables:
    - Create .env in the server directory:
    - MONGO_URI=mongodb://localhost:27017/booknest
    - JWT_SECRET=your_jwt_secret_key

PORT=5000

    - Create .env in the client directory:

REACT_APP_API_URL=http://localhost:5000/api

5. Start MongoDB service (if local): mongod

5. Folder Structure

- **Client** (React Frontend):
- client/
- ├── public/
- │ ├── index.html # Main HTML file
- │ └── favicon.ico # Favicon
- ├── src/
- │ ├── components/ # Reusable UI components (BookCard, Navbar, CartItem)
- │ ├── pages/ # Page components (Home, BookList, Cart, OrderHistory)
- │ ├── redux/ # Redux store, actions, and reducers
- │ ├── assets/ # Images, styles, and static files
- │ ├── App.js # Main app component with routing
- │ └── index.js # Entry point
- ├── package.json

└── tailwind.config.js # Tailwind CSS configuration

- **Server** (Node.js Backend):
- server/
- ├── routes/ # API routes (auth.js, books.js, orders.js)
- ├── models/ # MongoDB schemas (User.js, Book.js, Order.js, Category.js)
- ├── controllers/ # Request handlers (authController.js, bookController.js)
- ├── middleware/ # Authentication and error handling (auth.js, error.js)
- ├── config/ # Database connection (db.js)
- ├── server.js # Entry point

└── package.json

6. Running the Application

- **Frontend**:
  - Navigate to client directory: cd client
  - Run: npm start
  - Access at: http://localhost:3000
- **Backend**:
  - Navigate to server directory: cd server
  - Run: npm start
  - API accessible at: http://localhost:5000/api

7. API Documentation

- **Base URL**: http://localhost:5000/api
- **Endpoints**:
  - **Auth**:
    - POST /auth/register
      - Parameters: { username, email, password }
      - Response: { userId, token }
      - Example:
      - Request: { "username": "john_doe", "email": "john@example.com", "password": "secure123" }

Response: { "userId": "12345", "token": "eyJhbGciOiJIUzI1Ni..." }

- POST /auth/login
  - Parameters: { email, password }
  - Response: { userId, token }
  - Example:

        - Request: { "email": "john@example.com", "password": "secure123" }

Response: { "userId": "12345", "token": "eyJhbGciOiJIUzI1Ni..." }

- **Books**:
    - GET /books
        - Response: [{ id, title, author, genre, price, availability, description, ratings }]
        - Example:

Response: [{ "id": "1", "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "genre": "Fiction", "price": 12.99, "availability": true, "description": "A classic novel...", "ratings": 4.5 }]

    - GET /books/:id
        - Response: { id, title, author, genre, price, availability, description, ratings }
    - GET /books/category/:category
        - Response: [{ id, title, author, genre, price, availability, description, ratings }]
- **Orders**:
    - POST /orders
        - Parameters: { userId, books: [{ bookId, quantity }], totalPrice }
        - Response: { orderId, books, totalPrice, status }
        - Example:
        - Request: { "userId": "12345", "books": [{ "bookId": "1", "quantity": 2 }], "totalPrice": 25.98 }

Response: { "orderId": "67890", "books": [{ "bookId": "1", "quantity": 2 }], "totalPrice": 25.98, "status": "Confirmed" }

    - GET /orders/:userId
        - Response: [{ orderId, books, totalPrice, status, date }]
- **Inventory**:
    - PUT /inventory/:bookId (Internal, authenticated)
        - Parameters: { quantity }
        - Response: { bookId, updatedStock }

8. Authentication

- **Method**: JSON Web Tokens (JWT)
- **Process**:
    - Users register or log in to receive a JWT, stored in the browser's local storage.

    - Protected routes (e.g., /orders, /profile) require the token in the Authorization header (Bearer <token>).
    - Middleware verifies tokens and restricts access to authorized users.
- **Security**:
    - Passwords are hashed using bcrypt before storage.
    - JWTs are signed with a secret key and have a 1-hour expiration.
    - HTTPS is recommended for production to encrypt data in transit.

9. User Interface

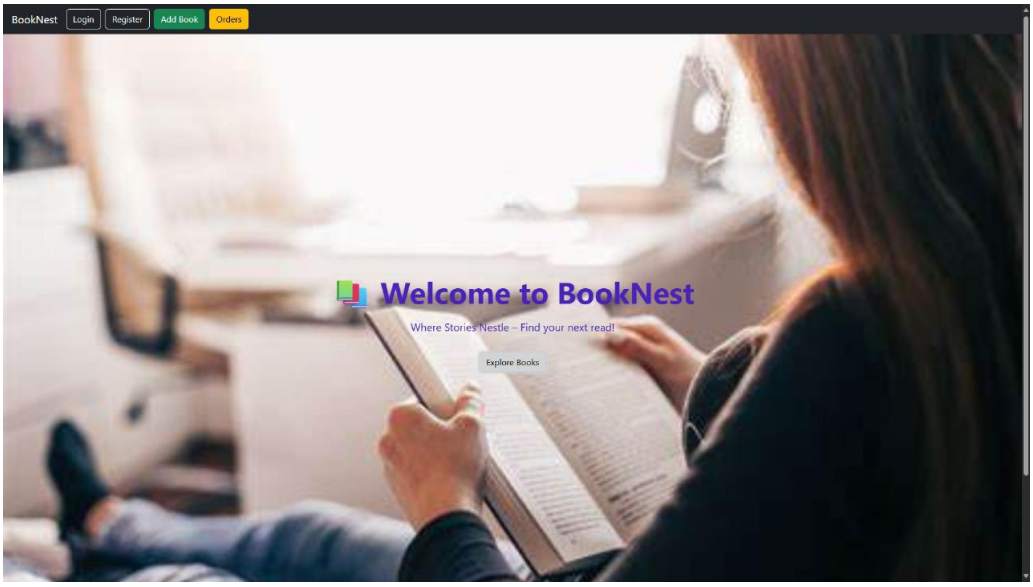
- **Features**:
    - **Home Page**: Welcomes users with featured books, categories, and a search bar.
    - **Book Listings**: Grid layout with filters for genre, author, ratings, and popularity.
    - **Cart**: Interactive cart for adding/removing books, updating quantities, and viewing totals.
    - **Order Confirmation**: Displays order ID, book details, and total price post-purchase.
    - **User Profile**: Shows order history, account settings, and tracking options.
- **Screenshots**: [To be added: Screenshots of Home, Book Listings, Cart, Order Confirmation, User Profile]
- **Responsiveness**: Tailwind CSS ensures a consistent experience across desktops, tablets, and smartphones.

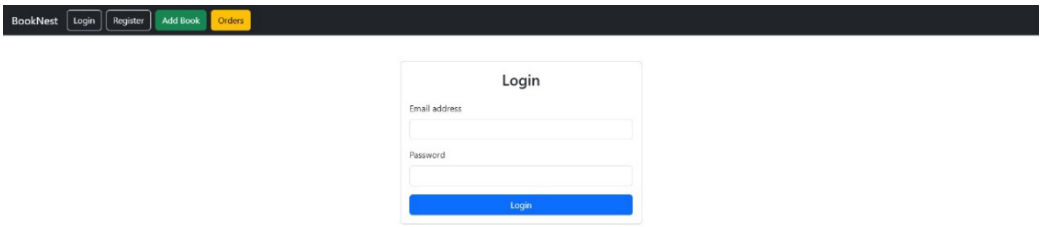
10. Testing

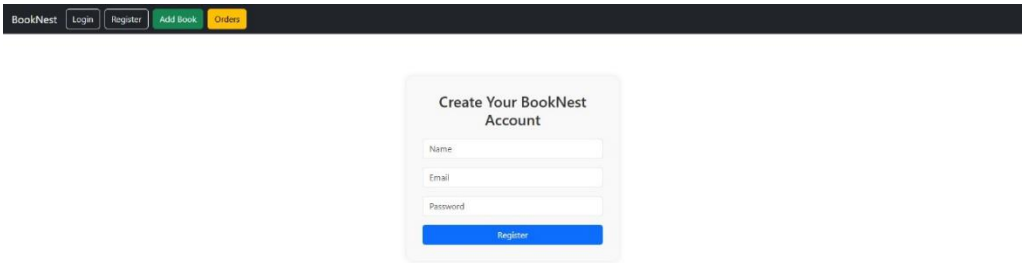
- **Strategy**:
    - **Unit Testing**: Jest for React components (e.g., BookCard rendering) and Node.js controllers (e.g., book retrieval logic).
    - **Integration Testing**: Supertest for API endpoints (e.g., /auth/register, /orders).
    - **End-to-End Testing**: Cypress for user flows (e.g., registration, book purchase, order tracking).
- **Tools**: Jest, Supertest, Cypress.
- **Coverage**: Aim for 80%+ code coverage for critical components.

11. Screenshots or Demo

- **Screenshots**:







- **Demo Link**: [To be added: Hosted demo URL, e.g., https://booknest-demo.herokuapp.com]

12. Known Issues

- Search functionality may lag with very large datasets (>10,000 books).
- Mobile responsiveness needs optimization for low-resolution devices (<320px width).
- Real-time inventory updates may delay during high traffic (>100 concurrent users).
- Cart persistence may fail if local storage is cleared unexpectedly.

13. Future Enhancements

- Personalized book recommendations using machine learning (e.g., collaborative filtering).
- Wishlist feature for users to save books for later.
- Integration with payment gateways like Stripe or PayPal.
- Support for eBooks and audiobooks with preview options.
- User review and rating system for books.
- Push notifications for order updates and new book releases.