Full Stack Development with MERN

1.INTRODUCTION:

PROJECT TITLE : Book store

TEAM MEMBERS: 1. Soni S-Backend implementation

2.Halips sharmila K – Fronted implement

3.Joice Evangilin N – Framework developing

4.Srimathi -packages & command line organizing

2.PROJECT OVERVIEW:

PURPOSE: Online bookstore is to make books more accessible, affordable, and discoverable to readers, while offering convenience, variety, and personalized experiences. For authors and publishers, it provides a broader platform for distribution and potential sales.

FEATURES: The key features and functionalities of an online bookstore revolve around creating a seamless, engaging, and personalized shopping experience for users. A successful online bookstore combines ease of navigation, comprehensive product listings, multiple payment options, and enhanced customer support with features like personalized recommendations, digital content, and community-building tools.

3.ARCHITECTURE:

FRONTEND:

```
├── /assets          # Static assets (images, icons, etc.)
├── /components      # Reusable UI components
├── /context         # React context for global state management (optional)
├── /hooks           # Custom hooks
├── /pages           # Page components for different routes
├── /services        # API services (functions for interacting with backend)
├── /store           # Redux store (optional, for state management)
├── /styles          # Global styles and theme configuration
├── /utils           # Utility functions and helpers
├── App.js            # Main component that houses routing and layouts
├── index.js           # Entry point for the app
├── serviceWorker.js     # For PWA functionality (optional)
```

BACKEND:

```
├── /config          # Configuration files (e.g., database, environment variables)
├── /controllers       # Logic for handling API requests
├── /middleware        # Custom middleware for validation, authentication, etc.
├── /models          # Mongoose models for the database schema (if using MongoDB)
├── /routes          # API routes
├── /services         # Business logic and interaction with external APIs or databases
├── /utils          # Utility functions (e.g., email service, payment gateway integration)
├── /validators        # Request body validation (using Joi or express-validator)
├── /app.js          # Express app setup
├── /server.js        # Server entry point (handles app initialization)
├── /db.js          # Database connection logic
├── /cron.js          # (Optional) For scheduled tasks (e.g., sending daily emails)
```

DATABASE:

This schema and the associated database interactions should provide the core functionality needed for a Bookstore application, including user management, book inventory, cart handling, and order processing. You can extend these models and operations based on specific requirements like payment gateway integration, book reviews, and user feedback.

4.SETUP INSTRUCTIONS:

PREREQUISTIES: Java scrip , node js , mongo db , react js framework.

INSTALLATION : Installing packages json, npm to run the module ,react installation for frame work.

5.FOLDER STRUCTURE:

CLIENT:

- **HomePage.js**: Displays the homepage with a list of books and a search bar.
- **BookDetailsPage.js**: Displays detailed information about a specific book.
- **CartPage.js**: Displays the user's shopping cart.
- **LoginPage.js**: Displays the login form.
- **ProfilePage.js**: Displays the user profile and order history.
- **OrderConfirmationPage.js**: Displays confirmation after an order is placed.

SERVER:

- **`/config/db.js`** - Sets up MongoDB connection using Mongoose.
- **`/controllers/bookController.js`** - Handles CRUD operations for books.
- **`/models/book.js`** - Mongoose schema for books.
- **`/routes/bookRoutes.js`** - Defines API routes for books (e.g., `GET /books`, `POST /books`).
- **`/middleware/authMiddleware.js`** - Middleware for verifying JWT tokens and securing routes.
- **`/utils/generateToken.js`** - Utility function to generate JWT tokens for user authentication.

6.RUNNING THE APPLICATION :

Frontend : npm start in the client directory.

Backend : npm start in the server direct (npm run start).

7.API DOCUMENTATION:

Register user:

- Endpoint: `POST /api/auth/register`
- Description: Register a new user.

Login User:

- Endpoint: `POST /api/auth/login`
- Description: Login a registered user and return a JWT token.

2.BOOK API:

Get all books:

- **Endpoint**: `GET /api/books`
- **Description**: Retrieve a list of all books.
- **Query Parameters** (optional):
    - `limit`: Number of books to return (default: 10)
    - page: Page number for pagination (default: 1

Get book id:

- **Endpoint**: `GET /api/books/:id`
- **Description**: Retrieve a book by its ID.
- **Path Parameters**:
    - `id`: The unique ID of the book.

Add new book (Admin only):

- **Endpoint**: `POST /api/books`
- **Description**: Add a new book to the inventory (requires admin privileges).

UPDATE BOOK INFORMATION:

- **Endpoint**: `PUT /api/books/:id`
- **Description**: Update details of an existing book (requires admin privileges)
- `id`: The unique ID of the book

Delate(Admin only):

- **Endpoint**: `DELETE /api/books/:id`
- **Description**: Delete a book from the inventory (requires admin privileges).
- **Path Parameters**:
- `id`: The unique ID of the book.

3.ORDER API:

Get User order:

- **Endpoint**: `GET /api/orders/user/:userId`
- **Description**: Retrieve all orders placed by a specific user.
- **Path Parameters**:
- `userId`: The unique ID of the user.

4.Cary API:

Add item cart:

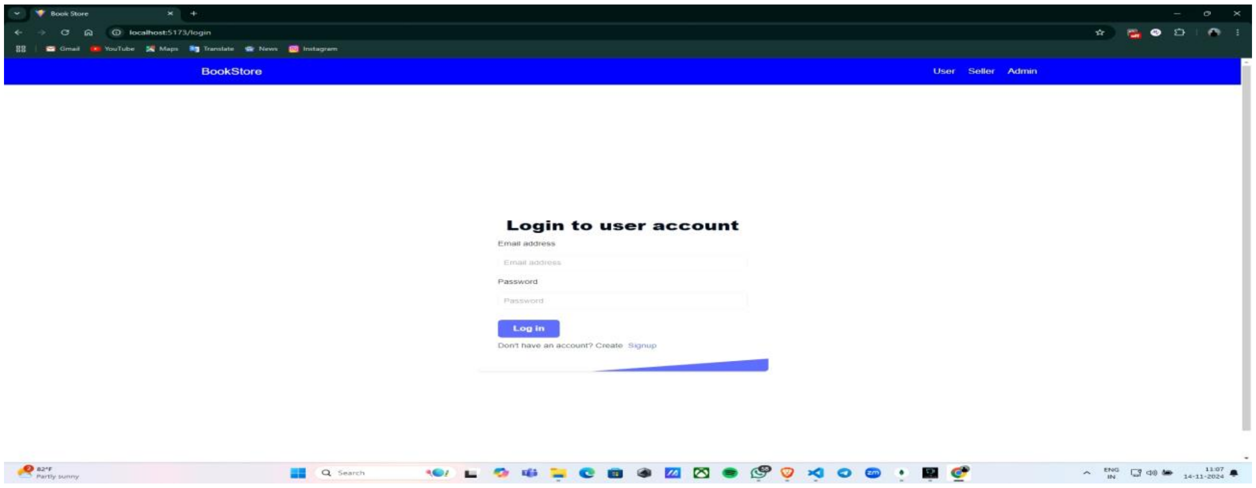
- **Endpoint**: `POST /api/cart`
- **Description**: Add a book to the user's cart.

8.AUTHENTICATION :

Authentication is a critical part of any application that involves user data and interactions. In this case, for the **Bookstore Application**, users must be able to **log in** to access their personalized features (such as placing orders, managing their profile, etc.).

Here, we will use **JWT (JSON Web Token)** for authentication. JWT allows the application to securely transmit user data in a stateless manner and is commonly used for API authentication.

9.USER INTERFACE:

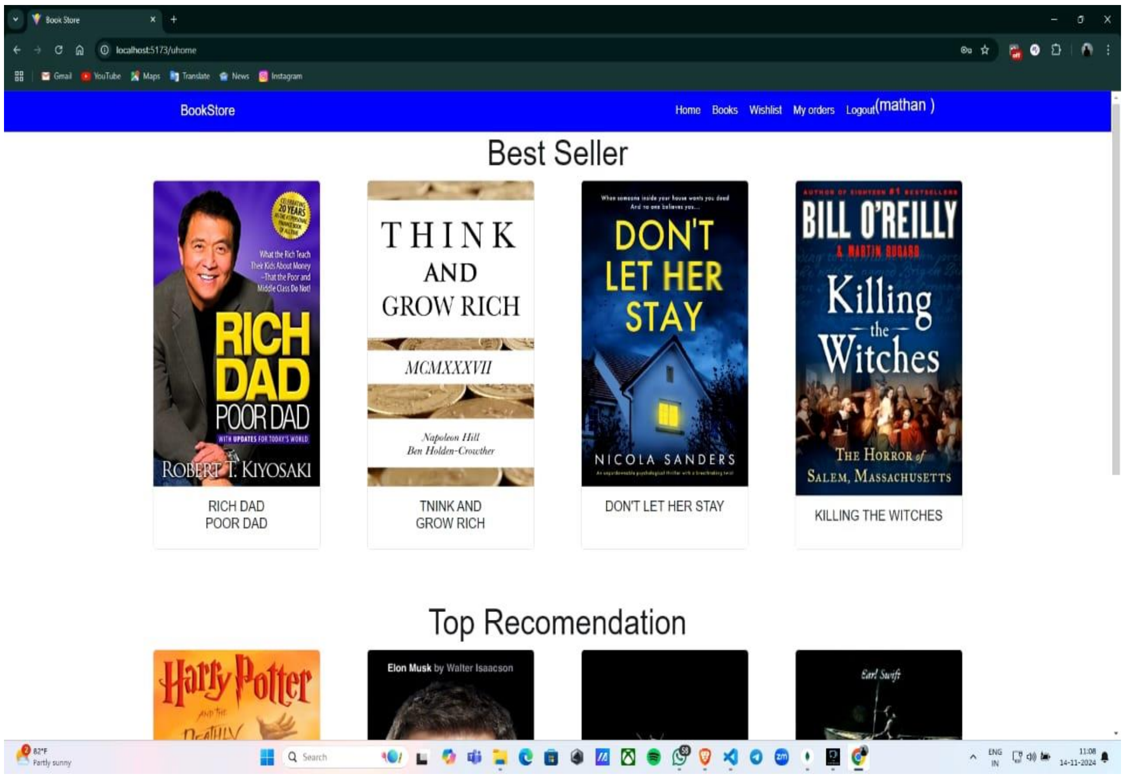


10.TESTING:

- **Unit testing** for individual components.
- **Integration testing** to ensure components interact as expected.
- **End-to-end testing** to simulate real-world user interactions.
- **API testing** to validate backend functionality.
- **Performance testing** to ensure scalability.
- **Security testing** to protect against common vulnerabilities.

11.SCREENSHOTS OR DEMO:

https://drive.google.com/file/d/151twyKMWpOrASqxqaUMREvpN7yyvupUv/view?usp=drivesdk



12.KNOWN ISSUES:

- **Database Issues**: Slow queries and poor schema design.
- **Authentication Problems**: Insecure password handling and token management.
- **Frontend Challenges**: Non-responsive UI and poor user experience.
- **Payment Integration**: Security and API issues with payments.
- **Real-Time Updates**: Stock synchronization and concurrency issues.

13.FUTURE ENHANCEMENT:

- **AI Chatbot**: Implement a chatbot for support and recommendations.
- **Analytics Dashboard**: Provide admins with detailed sales and user insights.
- **Augmented Reality (AR)**: Offer virtual previews of books.
- **Donations**: Enable users to donate books or funds to charities.
- **Book Clubs**: Create virtual book clubs and author events.
- **Audiobooks & eBooks**: Add digital books to the platform.