BOOK STORE USING MERN STACK

NAAN MUDHALVAN PROJECT REPORT

Submitted by

HARITHA.S (311521243015)
KRISHNIKA.T (311521243026)
ABISHEK RAJ.R.B (311521243004)
SHAKKIL AHAMED.S.A.K (311521243047)

in partial fulfillment for the award of the

degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE,

KODAMBAKKAM, CHENNAI-24

ANNA UNIVERSITY: CHENNAI 600 025

DEC 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**BOOK STORE USING MERN STACK** " is the bonafide work of "**HARITHA.S (311521243015), KRISHNIKA.T (311521243026), ABISHEK RAJ.R.B (311521243004),SHAKKIL AHAMED.S.A.K (311521243047)"** who carried out the project work under my supervision.

SIGNATURE

Mrs.N.MATHANGI, M.E., M.B.A, [Ph.D.]

HEAD OF THE DEPARTMENT

Artificial Intelligence and Data Science

Meenakshi Sundararajan Engineering College

363, Arcot Road, Kodambakkam,

Chennai -600024

SIGNATURE

Mrs. P. Muthulakshmi.,M.E

ASSISTANT PROFESSOR

Artificial Intelligence and Data Science

Meenakshi Sundararajan Engineering College No.

No. 363, Arcot Road, Kodambakkam,

Chennai – 600024

Submitted for the project viva voce of Bachelor of Technology in Artificial Intelligence and Data Science held on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi,** our beloved Secretary Mr. N. Sreekanth, Principal Dr. S. V. Saravanan for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Mrs.N.MATHANGI** M.E., M.B.A, [Ph.D.], Associate Professor Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We're grateful to **Mrs. P. Muthulakshmi.**,M.E,Internal Guide, Assistant Professor as our project coordinators for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and Blessings it wouldn't have been possible.

ABSTRACT

This project aims to develop a dynamic and user-friendly online bookstore application using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The platform provides an efficient, scalable, and interactive solution for purchasing, managing, and browsing books in a digital environment.

The application is designed to cater to book enthusiasts by offering features such as user authentication, personalized book recommendations, and a seamless shopping cart experience. Customers can browse books by categories, search for specific titles or authors, and view detailed descriptions, reviews, and ratings. Additionally, an admin panel facilitates inventory management, order tracking, and sales reporting.

The frontend is built using React.js, ensuring a responsive and intuitive user interface. Node.js and Express.js handle the backend operations, including user management, order processing, and API endpoints. MongoDB serves as the database, providing a robust solution for storing book details, user data, and transaction histories.

Key features include secure payment gateway integration, advanced search functionality, real-time book availability updates, and a wishlist. The application is designed with scalability and performance in mind, leveraging RESTful APIs and modern web development practices.

This project demonstrates the practical application of full-stack development using the MERN stack and highlights the potential of modern web technologies in e-commerce solutions.

TABLE OF CONTENTS

CHAPTER 1

1. PROJECT OVERVIEW

1.1 PURPOSE

The purpose of this project is to create a scalable and user-friendly online bookstore platform using the MERN stack (MongoDB, Express.js, React.js, Node.js) to revolutionize the traditional book-shopping experience. The application aims to provide users with a convenient way to browse, search, and purchase books online, offering features such as personalized recommendations, detailed book descriptions, and a wishlist. For administrators, it streamlines inventory management, enabling efficient tracking of stock levels and order processing. With a responsive interface and secure payment gateway integration, the platform ensures a seamless and trustworthy shopping experience. Designed for scalability and performance, it can accommodate an expanding user base and book collection. Ultimately, this project promotes a culture of reading by offering easy access to a diverse range of books while demonstrating the potential of full-stack web development for modern e-commerce solutions.

1.2 FEATURES

User-Friendly Interface:

A responsive and intuitive React-based frontend for seamless navigation and enhanced user experience across devices.

User Authentication and Authorization:

Secure login and registration using encrypted credentials.Role-based access control for users and administrators.

Advanced Book Browsing:

Browse books by categories, genres, authors, and publishers.Advanced search functionality with filters for price, ratings, and availability.

Personalized Recommendations:

Machine learning or rule-based algorithms to suggest books based on user preferences and purchase history.

Shopping Cart and Wish-list:

Add items to the cart for immediate purchase or save them to a wish-list for future reference.

Real-Time Book Availability:

Dynamic updates on stock levels and availability status.

Order Management:

Track order history, status, and delivery updates for users.Admin panel for managing orders, cancellations, and returns.

Inventory Management:

Admin capabilities to add, update, or remove books from the inventory.

Payment Gateway Integration:

Secure payment processing through trusted gateways with support for multiple payment methods.

User Reviews and Ratings:

Allow users to provide feedback on books to help others make informed choices.

.

CHAPTER 2

2. ARCHITECTURE

The **Bookstore** application is built on the MERN stack (MongoDB, Express.js, React, Node.js), delivering a full-stack solution with efficient data handling, responsive user interactions, and real-time updates. By leveraging the strengths of each component in the stack, the architecture supports a seamless shopping experience while ensuring scalability and secure management of inventory and user data. The client-server model separates concerns between the frontend and backend. The frontend provides a dynamic user interface for customers and admins, while the backend manages business logic, data processing, and secure communication.

2.1 FRONTEND ARCHITECTURE

- **React:** React is used to build a user-friendly interface with reusable components, offering customers features like book browsing, searching, and purchasing. Admins can efficiently manage inventory and orders. React's virtual DOM ensures optimal performance by updating only the necessary components.
- **Axios:** Axios facilitates communication between the frontend and backend through API calls, streamlining the process of fetching book details, managing user accounts, and handling transactions.

2.2 BACKEND ARCHITECTURE

- **Node.js:** Node.js powers the backend as a runtime environment, supporting asynchronous operations for handling multiple requests, such as order processing and user authentication.
- **Express.js:** Express serves as the backend framework for managing server-side logic, including routing, inventory updates, order tracking, and RESTful API creation.
- **JWT Authentication:** JSON Web Tokens (JWT) secure the application by validating user identities, roles, and permissions for access to restricted features like admin functionalities or order histories.

2.3 DATABASE ARCHITECTURE

- **MongoDB:** MongoDB is the NoSQL database chosen for storing flexible and scalable data structures. It houses records for books, users, orders, and inventory.
- **Document Structure:** MongoDB's document-based schema simplifies the management of collections like user accounts, book listings, shopping carts, and order histories.

CHAPTER 3

3. SETUP INSTRUCTIONS

3.1 PREREQUISITES

Before setting up the "Book Store" application, make sure the following prerequisites are met:

1. **Operating System**

   A Windows 8 or higher machine is recommended, though the setup should also work on macOS and Linux systems.

2. **Node.js**

   Download and install Node.js (version 14 or above). Node.js is required to run both the backend server (Node and Express) and the frontend server (React).

3. **MongoDB**

   Local MongoDB Installation: Install MongoDB Community Edition from MongoDB's official site if you prefer to use a local database.

   MongoDB Atlas (Optional): Alternatively, you can set up a free MongoDB Atlas account to use MongoDB in the cloud. MongoDB Atlas provides a connection string that will be needed to configure the backend.

4. **Two Web Browsers**

   The application works best with two web browsers installed for simultaneous testing (e.g., Google Chrome, Mozilla Firefox).

5. **Internet Bandwidth**

A stable internet connection with a minimum speed of 30 Mbps is recommended, especially if using MongoDB Atlas or deploying to a remote server.

6. **Code Editor**

Visual Studio Code or any other preferred code editor for easy management of the codebase and environment configuration.

3.2 INSTALLATION

1. **Install Node.js and MongoDB**

Node.js: Download and install Node.js (Version 14 or above).

MongoDB: Download and install the MongoDB Community Edition and set up MongoDB on your machine. Alternatively, you can use MongoDB Atlas for cloud hosting.

2. **Clone the Repository**

Open a terminal and clone the project repository:

git clone <repository_url> cd book-a-doctor

3. **Install Backend Dependencies**

Navigate to the backend folder and install the necessary Node.js packages:

cd backend npm install

4. **Install Frontend Dependencies**

Open a new terminal window or navigate back to the root directory, then go to the frontend folder and install dependencies.

cd ../frontend npm install

5. **Configure Environment Variables**

- In the backend folder, create a .env file.
- Add the following environment variables:

  MONGO_URI=<your_mongodb_connection_string>

  PORT=8000

  JWT_SECRET=<your_jwt_secret>

6. **Run MongoDB**

   If using MongoDB locally, ensure the MongoDB server is running:

   mongod

7. **Start the Backend Server**

   In the backend folder, start the server with the following command:

   npm start

   This will start the backend server on http://localhost:8000.

8. **Start the Frontend Server**

   In the frontend folder, start the React development server:

   npm start

   This will start the frontend server on http://localhost:3000.

9. **Access the Application**

   Open your web browser and navigate to http://localhost:3000 to view and interact with the application

CHAPTER 4

4. FOLDER STRUCTURE

The "Book Store" application follows a well-organized folder structure for both the React frontend and Node.js backend. This structure ensures that components, APIs, and utilities are easy to locate, modify, and scale.

4.1 CLIENT: REACT FRONTEND STRUCTURE

The frontend is structured using the following folders:

```
client/
│
├── public/
│   └── index.html                    # Root HTML file for React application
│
├── src/
│   ├── components/                   # Reusable UI components (e.g., forms, buttons)
│   ├── pages/                        # Specific page components (e.g., Home, Booking, Profile)
│   ├── services/                     # API request handlers using Axios
│   ├── context/                      # Context API for global state management
│   ├── assets/                       # Images, fonts, and other static resources
│   ├── App.js                        # Main App component with routing
│   ├── index.js                      # ReactDOM rendering entry point
│   └── styles/                       # Global CSS files and theming
│
└── package.json                      # Frontend dependencies and scripts
```

- ✓ **public/index.html**: The root HTML file that React renders to.
- ✓ **src/components/**: Houses reusable UI components, such as buttons, input fields, or modals.
- ✓ **src/pages/**: Contains main pages like home, booking, and profile, structured as components to simplify routing and navigation.
- ✓ **src/services/**: Manages API calls using Axios, allowing components to interact with the backend.
- ✓ **src/context/**: Handles global state management (e.g., user session) using React Context API.
- ✓ **src/assets/**: Stores images, fonts, and other static files.
- ✓ **src/styles/**: Holds global styling and theme files for consistent UI appearance.

4.2 SERVER: NODE.JS BACKEND STRUCTURE

The backend server is structured as follows: server/

```
│
├── config/
│   └── db.js                          # Database connection configuration
│
├── controllers/                       # Logic for handling API requests
│   ├── authController.js              # Authentication (signup/login) logic
│   ├── doctorController.js            # Logic for doctor-related actions
│   └── appointmentController.js       # Logic for appointment handling
│
```

```
├── models/                          # MongoDB data models (schemas)
│   ├── User.js                      # User schema for patient and doctor data
│   ├── Appointment.js               # Appointment schema
│   └── Doctor.js                    # Doctor schema and profile information
│
├── routes/                          # API routes
│   ├── authRoutes.js                # Routes for authentication endpoints
│   ├── doctorRoutes.js              # Routes for doctor-related endpoints
│   └── appointmentRoutes.js         # Routes for appointment management
│
├── middlewares/                     # Middleware for request handling
│   ├── authMiddleware.js            # Middleware for user authentication
│   └── errorMiddleware.js           # Custom error handling middleware
│
├── utils/                           # Utility functions
│   └── jwt.js                       # JWT token creation and verification functions
│
├── .env                             # Environment variables (e.g., MongoDB URI)
├── server.js                        # Main entry point for the backend server
└── package.json                     # Backend dependencies and scripts
```

- ✓ **config/db.js**: Establishes and exports the MongoDB connection.
- ✓ **controllers/**: Houses business logic for various resources (authentication, doctors, appointments).
- ✓ **models/**: Contains Mongoose schemas representing MongoDB collections (e.g., User,Doctor)

- ✓ **routes/**: Defines API endpoints and maps them to controller functions.
- ✓ **middlewares/**: Manages middleware functions, such as authentication and error handling.
- ✓ **utils/**: Includes helper functions, such as JSON Web Token (JWT) handling for secure authentication.
- ✓ **server.js**: The main server file that initiates Express, connects to MongoDB, and starts listening on a port.

CHAPTER 5

5. RUNNING THE APPLICATION

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

5.1 SET UP THE FRONTEND (SERVER)

1. Open another terminal window and navigate to the cl i ent directory:

   cd client

2. Install the frontend dependencies:

   npm install

3. Start the frontend server:

   npm start

The frontend server will launch at ht t p: / / l oc al hos t : 3000

5.2 SET UP THE BACKEND (SERVER)

1. Navigate to the s er ver directory:

   cd server

2. Install the necessary backend dependencies:

   npm install

3. Create a . e nv file in the root of the s er ver directory and add your environment variables (e.g., MongoDB URI, JWT secret, etc.)

   MONGODB_URI=your_mongo_connection_url JWT_SECRET=your_jwt_secret PORT=8000

4. Start the backend server:

   npm start

   The server will run on http://localhost:8000 by default (unless you specify a different port in the .env file).

CHAPTER 6

6. API DOCUMENTATION

The following section documents the endpoints exposed by the backend server of the "Book a Doctor" application. Each endpoint includes details on the HTTP request methods, parameters, and example responses.

6.1 ORDER A BOOK BY THE USER:

- Endpoint: / api / aut h/ r egi st er
- Method: GET
- Description: Order a book.

REQUEST BODY

{

"_id": "673f3e014cde611401efe5ff",

"name": "A Good Girl's Guide to Murder",

"price": "245.67",

"description": "Holly Jackson. Holly Jackson was born in 1992. She grew up in Buckinghamshire and started writing stories from a young age, completing her first (poor) attempt at a book aged fifteen. 'A Good Girl's Guide to Murder' is a YA Mystery Thriller and her debut novel.",

"ratings": "4.5",

"category": "Thriller",

"seller": "Amazon",

"stock": "5"

}

EXAMPLE RESPONSE:

{
_id: objectID('673f3ba2855fe3dcadcffb15')

cartItems : Array (1)
Amount : "NaN"
Status : pending"
__v : 0}

CHAPTER 7

7. TESTING

To ensure a robust and reliable "Book Store" application, a comprehensive testing strategy has been implemented, covering both frontend and backend functionality. Here is an overview of the testing approach and tools used:

7.1 UNIT TESTING:

Description: Unit tests are written to verify individual functions and modules. This helps identify any issues at the component level early in the development process**.**

Tools Used:

Jest for testing JavaScript functions, especially on the backend. Mocha and

Chai for testing API endpoints and other backend logic.

Example Tests: Checking API responses for login, registration, and appointment booking.

7.2 INTEGRATION TESTING:

Description: Integration tests are performed to verify that different modules and services work well together. This includes interactions between the client and server, as well as interactions with the database.

Tools Used:

Jest and Enzyme (for React) to test component interactions on the frontend. Supertest in

combination with Mocha for testing API routes and their responses.

Example Tests: Testing the flow from user registration to booking an appointment and retrieving user-specific data from the database.

7.3 END-TO-END (E2E) TESTING:

Description: E2E tests simulate user behavior to ensure the application flows as expected from start to finish. This includes testing the entire user journey, from logging in to scheduling an appointment.

7.3.1 Tools Used:

Cypress is used to automate browser-based tests, simulating real-world user interactions.

Example Tests: Verifying that a patient can search for a doctor, view their profile, and successfully book an appointment.

7.3.2 Manual Testing:

Description: In addition to automated tests, manual testing is conducted to check the application's usability, accessibility, and responsiveness on various devices and screen sizes.

Scope: Verifying layout consistency, button functionality, error messages, and mobile responsiveness.

7.3.3 Code Coverage:

Description: Code coverage reports help ensure that a significant portion of the codebase is tested. It highlights untested areas that may need attention.

Tools Used: Istanbul for measuring code coverage with Jest and Mocha tests**.**

7.3.4 Continuous Integration (CI):

Description: CI is set up to automatically run tests on every commit or pull request, ensuring that new changes do not introduce regressions.

CHAPTER 8

ADVANTAGES

- **Full-Stack Solution**

The MERN stack provides a cohesive environment with all components (MongoDB, Express.js, React, Node.js) working seamlessly together. This simplifies development and maintenance.

- **Scalability**

MongoDB's flexible schema allows the easy addition of new fields to the database, making it ideal for managing a growing inventory of books, user accounts, and orders.Node.js efficiently handles multiple simultaneous requests, ensuring the app remains responsive during peak traffic.

- **Efficient Data Handling**

MongoDB, a NoSQL database, excels at storing complex data such as book descriptions, categories, user preferences, and reviews, enabling faster retrieval and updates.React's virtual DOM optimizes rendering, improving performance during data-heavy interactions, like browsing large book catalogs or filtering search results.

- **Interactive User Experience**

React enables the development of dynamic and responsive user interfaces, enhancing the customer experience through smooth browsing, searching, and cart management.

- **Cost-Effective Development**

Open-source technologies across the MERN stack eliminate licensing costs.A single programming language, JavaScript, is used across the stack, reducing development complexity and resource requirements.

- **Secure Transactions**

JSON Web Tokens (JWT) ensure secure user authentication and authorization, protecting sensitive data like payment information and order histories.MongoDB supports encryption at rest and during data transmission, enhancing security.

- **Rapid Development**

Pre-built libraries and packages in Node.js and React expedite development.Developers can reuse React components for features like book cards, search filters, and shopping cart items.

.

CHAPTER 9

DISADVANTAGES

Learning Curve

1. Although JavaScript is used across the entire stack, developers need to learn multiple technologies (MongoDB, Express.js, React, and Node.js). Each of these has its own set of conventions and best practices, which can make it challenging for beginners or new team members to get up to speed quickly.

NoSQL Database Limitations

1. MongoDB is a NoSQL database, which offers flexibility in data modeling. However, for complex relationships (e.g., handling complex transactions or multi-table joins), MongoDB may not be the best choice compared to relational databases like MySQL or PostgreSQL. This can result in inefficiencies when managing complex queries or ensuring data consistency.

2. Data integrity and consistency in a NoSQL setup like MongoDB could be harder to maintain in certain scenarios, especially when scaling.

Performance with Large Datasets

1. As the bookstore grows and the inventory or customer base expands, MongoDB can face performance issues, particularly with large datasets. Handling a large number of transactions or queries simultaneously can lead to slower response times unless carefully optimized (e.g., through indexing or sharding).

2. Complex data aggregation operations can be slower in MongoDB compared to SQL databases, which are optimized for such queries.

Overhead with Real-Time Features

1. While Node.js is excellent for handling real-time updates (e.g., for inventory updates or order status), the asynchronous nature of JavaScript requires careful management of asynchronous operations. If not handled properly, it can result in callback hell, memory leaks, or performance bottlenecks.

Limited Built-In Features for Enterprise Applications

1. While the MERN stack is great for building general web applications, it may lack some of the enterprise-level features and tools that more specialized frameworks or stacks (e.g., Java Spring or .NET) provide. This includes advanced caching, reporting, or analytics out of the box.

CHAPTER 10

FUTURE ENHANCEMENTS:

The "Order a Book" application is designed to be scalable and open to future improvements. Here are some potential features and enhancements planned to improve user experience and expand functionality:

Enhanced User Experience

- Search and Filter Options: Implement advanced search with filters like genre, price range, author, or publication year.

- Personalized Recommendations: Use machine learning or collaborative filtering to suggest books based on user preferences or browsing history.

Improved Security

- Role-Based Access Control (RBAC): Limit actions based on user roles (admin, customer, vendor).

- Two-Factor Authentication (2FA): Add an extra layer of security during user login.

Expanded Features

- Wishlist and Notifications: Allow users to save items for later and notify them of price drops or new arrivals.

- Ratings and Reviews: Enable customers to leave reviews and rate books.

Scalability Enhancements

- Pagination and Infinite Scrolling: Improve performance for browsing large inventories.

- Microservices Architecture: Transition to a microservices approach for better scalability and maintenance.

Integration Capabilities

- Third-Party Payment Gateways: Support for multiple payment providers like Stripe or PayPal.

- Delivery Tracking: Integrate APIs to provide real-time order tracking.