**FULL STACK DEVELOPMENT WITH MERN**

**PROJECT DOCUMENTATION**

**BOOK SELLING WEBPAGE**

**FULL STACK DEVELOPMENT WITH MERN**

**CONTEXT**

|  |  |
| --- | --- |
| **S.NO** | **TITLE** |
| **1** | **INTRODUCTION** |
| **2** | **PROJECT OVERVIEW** |
| **3** | **ARCHITECTURE** |
| **4** | **SETUP INSTRUCTION** |
| **5** | **RUNNING THE APPLICATIION** |
| **6** | **FOLDER STRUCTURE** |
| **7** | **AUTHENTICATIION** |
| **8** | **USER INTERFACE AND SCREENSHOT** |
| **9** | **DEMO LINK** |

**FULL STACK DEVELOPMENT WITH MERN**

* **PROJECT TITLE: BOOK SELLING WEBPAGE**
* **TEAM MEMBERS:**

**1. TRISHA.R**

**2. PRIYA DHARSHINI.L**

**3. PREETHA.M**

**4. SWATHI.G**

**5. MUTHU LAKSHMI.S**

**FULL STACK DEVELOPMENT WITH MERN**

**PROJECT DOCUMENTATION**

**INTRODUCTION**

Discover a world of stories, knowledge, and inspiration at your fingertips. Our online bookstore offers an extensive collection of books across all genres—fiction, non-fiction, academic, self-help, and more. Whether you're a bookworm, a student, or just looking for your next great read, we’ve got something for everyone.

Explore top-selling titles, rare finds, and personalized recommendations. With seamless browsing, secure payments, and fast delivery, we ensure an exceptional shopping experience.

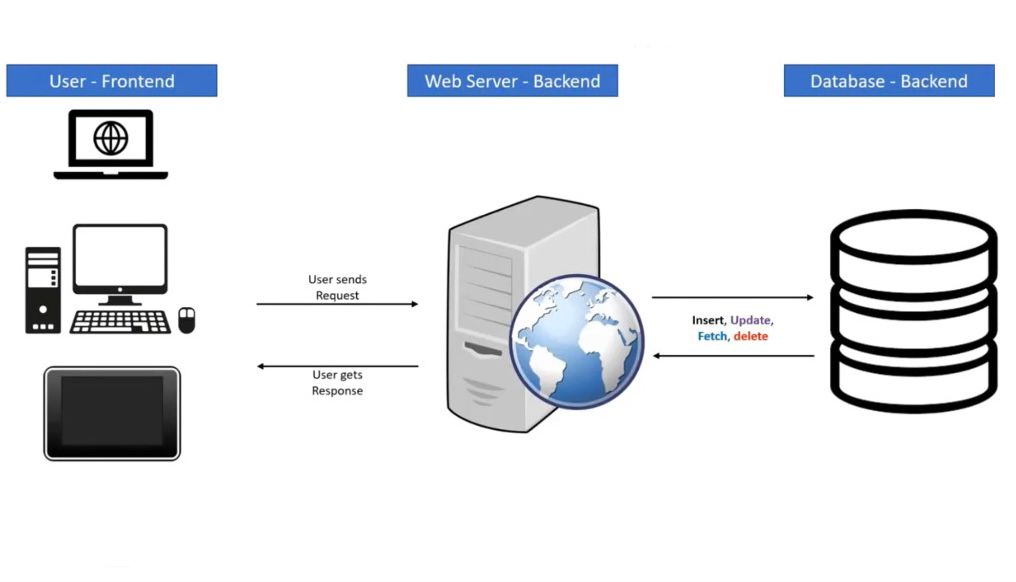
**PROJECT OVERVIEW**

* **PURPOSE:** 
  + - The purpose of the BookEase project is to create a user-friendly and efficient online platform for book enthusiasts to buy, sell, and explore a wide range of books. By leveraging the MERN (MongoDB, Express.js, React.js, and Node.js) stack, the project aims to streamline the process of book transactions and provide a seamless experience for buyers, sellers, and administrators. The platform aspires.
    - Bridge the gap between book sellers and buyers through a convenient digital interface.
    - Offer personalized services like wishlists, order tracking, and seller dashboards.
    - Enhance accessibility to books by supporting diverse genres and categories.
* **FEATURES:**

The platform is designed with the following features in mind:

1. **Ease of Use**  
   Create an intuitive and visually appealing user interface that ensures a smooth navigation experience for all users, including buyers, sellers, and administrators.
2. **Efficient Book Management**  
   Provide tools for sellers to manage their inventories, such as adding, editing, and removing books, while also enabling buyers to search, browse, and purchase books effortlessly.
3. **Seamless Transactions**  
   Facilitate secure and straightforward transactions between buyers and sellers, ensuring proper order management and tracking.
4. **Dynamic Features**  
   Offer features like personalized wishlists, seller performance tracking, and real-time order updates to enhance user engagement and satisfaction.
5. **Scalability**  
   Develop the platform with a modular architecture to accommodate future growth and additional features like book recommendations, reviews, and advanced search capabilities.
6. **Community Building**  
   Promote a vibrant community of book lovers by connecting individuals with similar interests through a shared platform for discovering and purchasing books.

**ARCHITECTURE**



**FIGURE 1**

* **FRONTEND:**

React is used in front-end development to build user interfaces and add client-side functionality, enabling the creation of dynamic pages, interactive elements, and features for websites. React is a popular JavaScript library developed by Facebook in 2013 for building web app UIs. It enables developers to create reusable UI components and simplifies building complex interfaces. React uses a virtual DOM for efficient state management and fast updates.

* **Component-based architecture:** With React, you can create reusable user interface (UI) components that can be swiftly combined to create intricate UIs.
* **JSX:** By allowing you to write HTML-like code in JavaScript files, the JSX syntactic extension for JavaScript makes it easier to write and understand code.

React supports a wide range of libraries and frameworks, including Redux, React Router, and Material UI. Additionally, it is quite expandable. React is a robust and flexible framework that can be used to build user interfaces for internet applications.

* **BACKEND:**

The backend of the **Book Selling Webpage** is designed to handle requests, manage business logic, and interact with the database. It is built using **Node.js** and **Express.js** with MongoDB as the database.

**Backend Features**

The backend architecture using Node.js and Express.js includes:

**1. Server Setup:** Node.js runs JavaScript on the server, with Express.js handling routing and middleware**.**

**2. Routing:** Manages API endpoints for handling requests (GET, POST, etc.).

**3. Middleware:** Processes requests (e.g., authentication, body parsers).

**4. Error Handling:** Centralized error handling for graceful failure.

**5. Authentication & Authorization:** Manages user sessions with JWT or OAuth.

**6. Environment Configuration:** Stores sensitive data in environment variables.

**7. Logging & Monitoring:** Tracks app activity and performance.

**8. Scalability:** Deployed on cloud platforms with load balancing for traffic handling.

* **DATABASE:**

**1. Schema Design**

In MongoDB, data is stored in collections (similar to tables in relational databases) as documents (like rows in SQL). A schema defines the structure of these documents. In Node.js, Mongoose is commonly used to define and enforce schemas. A schema can specify fields, types, constraints (like required or unique), and defaults for data.

**2. MongoDB Interactions**

Mongoose provides methods to perform basic CRUD (Create, Read, Update, Delete) operations with MongoDB.

**Create:** Add new records to a collection.

**Read:** Retrieve documents using queries (e.g., find users by email).

**Update:** Modify existing documents (e.g., updating a user’s password).

**Delete:** Remove documents from the collection (e.g., delete a user by email).

These operations are done through the Mongoose model, which acts as an interface to interact with the collection based on the schema.

**3. Advanced MongoDB Features**

MongoDB offers several advanced features that improve performance and capabilities:

**Indexes:** Improve query speed by creating indexes on fields frequently queried (e.g., email).

**Aggregation:** Perform complex queries to manipulate or analyze data, such as counting the number of documents or grouping them by specific fields.

**Population:** Reference documents from other collections (similar to SQL joins), allowing you to fetch related data in one query.

**4. Schema Validation & Middleware**

MongoDB and Mongoose allow for data validation and pre/post-processing using middleware:

**Validation:** Ensures data integrity, for example, ensuring an email is unique or a password is required.

**Middleware:** Functions that run before or after certain actions. For instance, hashing passwords before saving them or logging actions after saving data.

**5. Connecting to MongoDB**

To interact with a MongoDB database, you need to establish a connection using Mongoose. Once connected, your Node.js application can perform the necessary operations on your MongoDB database.

**6. Error Handling & Optimizations**

MongoDB provides error handling for cases like validation failures or query errors. To optimize performance, techniques like indexing, pagination, and caching are often used**.**

**SETUP INSTRUCTION**

* **PREREQUISTES**

Before you start implementing the Book Store application, ensure the following prerequisites:

**1. Node.js:** JavaScript runtime to run server-side code

**2. npm:** Manages dependencies and installs packages.

**3. MongoDB:** NoSQL database for flexible data storage.

**4. Mongoose:** ODM library for MongoDB to handle schemas and queries.

**5. Express.js:** Web framework for handling HTTP requests.

**6. Postman (optional):** Tool for testing APIs**.**

**7. Text Editor/IDE:** Code editor like VS Code or Sublime.

**8. Git (optional):** Version control system for code management.

* **INSTALLATION:**

**Step-by-Step Guide for Installation:**

**1. Clone the Repository:**

* Open a terminal and navigate to the folder where you want to clone the project.

**Run the following command to clone the repository:**

* **git clone:** https: //github.com/priya-dharshini143/NM-Book-Store.git

**2. Navigate to Project Directory:**

* After cloning, navigate into the project folder:
* cd backend
* cd frontend

**3. Install Node.js Dependencies:**

* Ensure Node.js and npm are installed. You can check by running:
* node -v
* npm -v
* If Node.js is not installed, download and install it from Node.js official website.
* Install the project dependencies listed in package.json:
* npm install

**4. Set Up Environment Variables:**

* Add necessary environment variables such as database URI, port, and API keys.
* Example:**DB\_URI=**mongodb+srv://pd4008208:RIYA143@cluster0.pqg7g.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

**5. Start the Application:**

* Once dependencies are installed and environment variables are set, start the application:
* npm start
* This will run the server,
* npm run dev
* This will run the client, and you should see the application running at http://localhost:5173.

**6. Verify Setup:**

* Open your browser or use Postman to test the API and ensure everything is working properly.

That's it! Your Node.js and MongoDB environment should now be set up and ready for development.

**RUNNING THE APPLICATION**

Provide commands to start the frontend and backend servers locally.

* + Frontend: **npm run dev** in the client directory.
  + Backend: **npm start** in the server directory.

**FOLDER STRUCTURE**

**CLIENT:**

**SERVER:**

****

**AUTHENTICATION**

Authentication is the process of verifying the identity of a user. It ensures that the person or system trying to access the application is who they claim to be. Here's how it's typically handled:

**1. Login Credentials**: Users provide a username and password. The system checks these credentials against a database. If the credentials match, the user is authenticated.

**2. Token-based Authentication:** After successful login, the system may issue an authentication token (e.g., JSON Web Token - JWT) that the user includes in subsequent requests. This token proves the user's identity without needing to re-enter credentials.

**3. Session Management**: In some systems, a session is created after authentication, allowing the user to access resources until they log out or the session expires.

**AUTHORIZATION:**

Once the user is authenticated, authorization determines what resources they are allowed to access and what actions they can perform. It ensures that users can only access information or perform actions based on their roles and permissions.

**1. Role-based Access Control (RBAC):** Users are assigned roles (e.g., admin, user, manager) based on their function within the system. Each role has specific permissions that grant or restrict access to certain resources or actions.

**2. Permissions:** Permissions are the rules that define what users can do. For example, an admin may have permissions to view, edit, and delete resources, while a regular user may only be able to view resources.

By separating authentication and authorization, the system can maintain strict control over who can log in and what they can do once logged in, ensuring security and minimizing the risk of unauthorized access.

**TOKENS:**

* **Definition:** Tokens are pieces of data used to verify a user's identity or session. They are commonly used for authentication and authorization purposes, especially in web applications and APIs.
* **How Tokens Work:**
* **Authentication:** A user logs in, and the server issues a token (e.g., JWT). The user sends this token in subsequent requests to authenticate themselves.
* **Expiration:** Tokens can have an expiration time after which they are no longer valid. Some systems use refresh tokens to generate new tokens when expired.

**SESSIONS:**

* **Definition:** A session is a server-side storage mechanism used to keep track of a user's state during their interaction with an application. It allows the system to remember user information, preferences, and the user's current actions across multiple requests.
* **How Sessions Work:**
* **Session ID:** When a user logs in, a unique session ID is created and stored on the server. This ID is sent to the user’s browser as a cookie.
* **Session Management:** On each subsequent request, the browser sends the session ID in the cookie, allowing the server to retrieve the associated session data (like user information or authentication status).
* **Expiration:** Sessions typically expire after a certain period of inactivity or when a user logs out.

**USER INTERFACE AND SCREENSHOTS**

**ADMIN:**

**SELLER:**

**USER:**

**DEMO LINK:**

**THANK YOU!**