Bookstore-Management-System

using MERN stack.

Team Members

1. **SAI KUMAR S - D059CC9A26D9298BD89402A6981F397F .**
2. **SARAN S - 0080332B6EF01D288BF4DDBF65A043E7 .**
3. **SARATHI M - 24F70DFA5ECC63493D746B39864994FE .**
4. **SATHISH KUMAR S - 5B44D0178B84CF294E043D4EE4786697 .**

Institution : TJS ENGINEERING COLLEGE

Institute code : 1128

OBJECTIVE :

Creating a Bookstore Management System using the MERN stack involves many steps, from database design to frontend development, API implementation, security, and deployment. Below is a comprehensive project document for the Bookstore Management System using the MERN stack. This document outlines the complete system design and development process in detail ....

TABLE OF CONTENT

INTRODUCTION :

The Bookstore Management System is a web-based application designed to manage the operations of a bookstore. It provides an easy-to-use interface for customers to browse books, add them to their cart, and complete purchases. The system also allows admins to manage books, orders, and customers. This project leverages the MERN stack (MongoDB, Express.js, React.js, Node.js) to build both the frontend and backend of theapplication. The goal is to create a scalable, secure, and efficient application that provides seamless user and admin interactions .

MERN STACK OVERVIEW :

The MERN Stack is a combination of JavaScript-based technologies that help build dynamic and scalable web applications.

MongoDB :

- NoSQL database for storing data in a flexible, JSON- like format.
- Offers high scalability and performance, making it suitable for handling large amounts of data

Express.js :

- A web application framework for Node.js to handle HTTP requests and route logic.

•React.js:

A JavaScript library for building dynamic user interfaces by creating reusable UI components.

- **Reack.js:**
  - A JavaScript library for building dynamic user interfaces by creating reusable UI components
  - Enables the creation of reusable UI components, leading to faster

- **Node.js :**

A server-side JavaScript runtime used to execute the application code on the backend.The stack enables full-stack JavaScript development, meaning bothfrontend and backend use JavaScript as the programming language, allowing for a more streamlined development process

SYSTEM REQUIREMENT :

To build the Bookstore Management System, the following software and tools are required:

• **MongoDB :**
Orders, Users). For storing data (Books,
• **Node.js & npm :** To run the backend server.
• **Express.js :** A backend framework for handling routing andserver-side logic.
• **React.js :** For the frontend development.
• **Stripe API :** For payment processing.

• **Git :** For version control.
• **Visual Studio Code :** For coding.
• **Postman :** For testing APIs.
• **Heroku/Netlify :** For deployment.

Hardware :

Any modern computer with sufficient RAM (8GB or more) and disk space for development .

KEY FEATURES :

The **Bookstore Management System** includes the following key features:

• **User Management:** Allows users to register, log in, and manage their profiles.

• **Book Catalog:** Displays a list of available books, with detailed descriptions, images, and pricing.

• **Shopping Cart:** Users can add, update, and remove books from their cart.

• **Checkout System:** Users can proceed with checkout, provide shipping information, and complete payment.

• **Order Management:** Admins can view, manage, and track orders.

• **Admin Dashboard:** Provides admin features for managing books, users, and orders.

• **Payment Integration:** Integration with Stripe for secure online payments.

• **Search & Filtering:** Users can search and filter books based on category, price,or author.

SYSTEM ARCHITECTURE :

The **Bookstore Management** System follows a **client-server architecture**,where :

•**Frontend:**
Built using **React.js** to provide an interactive user interface.

•**Backend:**
Powered by Node.js and Express.js to manage API requests and handle business logic.

•**Database:**
MongoDB stores all data, including user profiles, books, orders, and shopping cart details. Communication between the frontend and backend happens through **RESTful APIs**.

DIAGRAM OF SYSTEM ARCHITECTURE :

```
+-------------+ +-------------+ +---------------+
| Client | <----> | Express.js | <----> | MongoDB |
| (React.js UI)| | (Backend) | | (Database) |
+-------------+ +-------------+ +---------------+
```

The above flowchart indicates the connectivity exist in required system.

DATABASE DESIGN(MongoDB) :

The **MongoDB** database is structured with the following collections:

- **Users Collection:**
  - **userId:** Unique identifier (auto-generated).
  - **username:** User's chosen name.
  - **email:** User's email address.
  - **password:** Encrypted password.
  - **role:** User's role (admin or customer).
- **Books Collection :**
  - **bookId:** Unique identifier for each book.
  - **title:** Name of the book.
  - **author:** Author of the book.

- **category:** Genre/category (e.g., Fiction, Non-Fiction).
- **description:** Short description of the book.
- **stock:** Quantity available in stock.
- **image:** URL of the book's image.

- **Orders Collection:**
  - **orderId:** Unique identifier for each order.
  - **userId:** User who placed the order.
  - **orderItems:** List of books in the order (bookId, quantity).
  - **totalPrice:** Total order value.
  - **status:** Order status (pending, completed).

- **Cart Collection:**
  - **userId:** User's ID associated with the cart.
  - **cartItems:** Liems in the cart (bookId, quantity).

BACKEND DEVELOPMENT (Node.js & Express) :

•The backend of the application is built using **Node.js** and **Express.js**, which allows for handling HTTP requests, managing business logic, and interacting with the database.

Setup:

•Install dependencies: **npm install express mongoose cors bcryptjs jsonwebtoken.**

•The **Node.js** and **Express.js** backend provides the API for handling requests.

Backend Setup:

```
npm init -y
npm install express mongoose cors
bcryptjs jsonwebtoken
```

Express.js handles routing.

Code Snippet: Basic Server Setup in Node.js/Express is given as follows :

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect('mongodb://localhos
t /bookstore', { useNewUrlParser: true,
useUnifiedTopology: true }); app.get('/',
(req, res) => { res.send('Bookstore API');
});

const port = process.env.PORT || 5000;
app.listen(port, () => {
console.log(`Server is running on port
${port}`);
});
```

• **Express.js** serves as the framework for defining routes and middleware.

• **Mongoose** is used to define MongoDB models and interact with the database.

• **API Endpoints:**

• **Authentication:**
• **POST /api/auth/register**: Registers a new user.
• **POST /api/auth/login**: Logs a user in and returns a JWT.

• **Books:**

• **GET /api/books:** Fetch all books.
• **POST /api/books:** Add a new book (admin only).
• **PUT /api/books/:id:** Update book details.
• **DELETE /api/books/:id:** Delete a book

- **Cart:**

- **GET /api/cart:** View the user's cart.
- **POST /api/cart:** Add items to the cart.
- **DELETE /api/cart:** Remove items from the cart.

- **Orders:**

- **POST /api/orders:** Place a new order.
- **GET /api/orders/:id:** View order details.

FRONTEND DEVELOPMENT (React.js) :

•The frontend is developed using **React.js**, which provides an interactive and dynamic user interface.

•Components:

•**App.js:** Main component with routing.
•**HomePage:** Displays a list of books with search and filter options.
•**BookDetail:** Shows details of a selected book.
•**Cart:** Displays items in the shopping cart.
• **Checkout:** Handles the checkout process and payment.
• **AdminDashboard:** Allows the admin to manage books, orders, and users.

• State Management:

• Use React Context API or Redux for global state management,
particularly for handling the cart and user authentication.

USER AUTHENTICATION & AUTHORIZATION :

• **Authentication** is handled using JWT (JSON Web Tokens) to secure the user’s session.

• **Register:** User creates an account by providing their email, username, and password (which is hashed).

• **Login:** User logs in with credentials; a JWT token is returned for session management.

• **Authorization:** Routes are protected using middleware to ensure only authorized users (admins) can access certain features (like adding/removing books).

API DESIGN :

The **RESTful API** is designed to allow communication between the frontend and backend. The following endpoints are implemented:

1.Authentication:

1.POST /api/auth/register: To register a user.

2.POST /api/auth/login: To log in and retrieve a token.

2.Books:

1.GET /api/books: Fetch all books.
2.POST /api/books: Admin can add new books.
3.PUT /api/books/:id: Admin can update a book.
4.DELETE /api/books/:id: Admin can delete a book.

3.Cart:

1.GET /api/cart: Get the current user's cart.
2.POST /api/cart: Add book to the cart.
3.DELETE /api/cart/:id: Remove book from the cart.

4.Orders:
1.POST /api/orders: Place an order.
2.GET /api/orders/:id: Get order details.

ADMIN DASHBOARD :

The **Admin Dashboard** allows the administrator to manage the bookstore efficiently. It includes features for:

- **Book Management:** Add, update, and delete books.
- **Order Management:** View and update the status of customer orders.
- **User Management:** View and manage users, including roles.

Book Management :

•Admins can manage books in the store by performing CRUD (Create, Read, Update, Delete) operations. The admin can:
•Add a new book with details like title, author, price, and stock.
•Edit or delete existing books from catalog.

Order Management :

Admins can track customer orders, including:
• View order details (e.g., items, total price, customer).
• Update the order status (e.g., pending, shipped).

Cart Management :

• Users can manage their cart by adding, removing, or updating book quantities. The cart persists until checkout or removal.

Payment Integration (Stripe) :

• The system integrates with Stripe for secure payment processing.
Users can pay for their orders using their credit cards.
• Setup Stripe API keys for test and live environments.
• Implement payment flow on the frontend.
• Handle payment confirmation and order creation in the
backend.

TESTING & DEBUGGING :

Testing is an essential part of any project. The following types of testing are implemented:

• **Unit Testing:** Using Jest to test individual functions in the backend.

• **Integration Testing:** Using Mocha and Chai to test API routes.

• **End-to-End Testing:** Using Cypress to test the complete user flow.

Deployment :

The application is deployed on:
• **Frontend:** Deploy React.js on **Netlify** or **Vercel.**
• **Backend:** Deploy Node.js/Express on **Heroku** or **DigitalOcean**.

SECURITY CONSIDERATION :

To secure the application:

• **JWT Authentication** is used to protect routes.

• **Password Hashing:** User passwords are hashed using bcrypt.

• **Data Encryption:** Use HTTPS for secure data transmission.

Performance Optimization :

Performance is optimized by:

• Caching frequently accessed data using **Redis**.

• Using **pagination** for large datasets (books list).

• Implementing **lazy loading** for images and content.

Scalability & Future Enhancements :

The system can scale horizontally with:

• Multiple instances of the backend.

- Load balancing to handle high traffic.

Future features may include:

- **Recommendation System:** Based on user activity.
- **Mobile App:** Develop a mobile app using **React Native.**

Challenges & Solutions :

1. State Management: Complex state management was simplified With **React Context API.**

2. Handling Large Data: We implemented pagination for managing large books catalogs.

CONCLUSION :

The **Bookstore Management System** demonstrates the power of the **MERN stack** for building full-stack applications . The system offers robust features for managing books, orders, users, and payments, providing a seamless experience for both customers and admins .

REFERENCES :

• **MERN Stack Documentation:** Official documentation for MongoDB, Express.js, React.js, and Node.js.

• **Stripe API Documentation:** For integrating Stripe payment gateway.

THANK YOU !!!