1. Project Title: [Book-store]

Team Members:

HEMA LATHA .R (frontend developer)

HEMA LATHA.R(backend developer)

SANJEEVI.R(databasemanagement)

SANTHOSH.G(testing and platform)

VIKNESH.C(deployment)

Submitted By

Institution/Organization Name: [SRI MUTHUKUMARAN INSTITUTE OF TECHNOLOGY]

Project Overview

Purpose: The Book Store project aims to create an engaging and user-friendly online platform for buying, selling, and discovering books. This project will focus on providing a comprehensive and interactive experience for book lovers, making it easy for them to find their favorite books, explore new releases, and manage their collections.

Goals:

- Provide a vast and diverse collection of books across various genres.
- Enhance the shopping experience with personalized recommendations and user reviews.
- Enable seamless transactions and secure payment options.

Features:

1. **Extensive Book Catalog:**
    - Wide range of books including fiction, non-fiction, academic, and more.
    - Detailed book descriptions, author information, and ratings.
2. **User Accounts and Profiles:**
    - Secure login and user registration.
    - Personalized user profiles with the ability to track purchase history and wishlist.
3. **Search and Filter:**
    - Advanced search functionality to find books by title, author, genre, and ISBN.
    - Filters to narrow down search results based on preferences.
4. **Recommendations and Reviews:**
    - Personalized book recommendations based on user preferences and purchase history.

    - User reviews and ratings to help readers make informed decisions.

5. **Shopping Cart and Checkout:**
    - Easy-to-use shopping cart for managing book purchases.
    - Secure and flexible payment options including credit/debit cards and digital wallets.
6. **Discounts and Promotions:**
    - Special offers, discounts, and promotions on various book categories.
    - Loyalty programs and reward points for frequent shoppers.
7. **Responsive Design:**
    - Mobile-friendly design for a seamless experience on all devices.
    - User-friendly interface with intuitive navigation.
8. **Community Features:**
    - Discussion forums for book enthusiasts to share recommendations and reviews.
    - Author interviews and book signing event announcements.

Architecture Overview: Book Store

Frontend: React

- **Component-Based Architecture**:
    - **Reusable Components**: Leverage reusable components for elements like book listings, navigation bars, and user profiles.
    - **State Management**: Utilize React's useState and useEffect hooks, along with Context API or Redux for more complex state management.
    - **Routing**: Use React Router for seamless navigation between different pages (e.g., home, book details, user profile, cart).
    - **UI Framework**: Integrate a UI framework like Material-UI or Bootstrap for a consistent and responsive design.
    - **API Calls**: Utilize Axios or Fetch API to interact with the backend services for fetching book data, user authentication, and transactions.

Backend: Node.js and Express.js

- **RESTful API**:
    - **Express.js**: Set up a server using Express.js to handle API requests and responses.
    - **Routing**: Define routes for different functionalities (e.g., /books, /users, /orders).
    - **Middleware**: Implement middleware for logging, error handling, and authentication (e.g., jsonwebtoken for JWT authentication).

- **Business Logic**:
  - Handle business logic related to user authentication, book listings, order processing, and payment integration.
  - Validate data using libraries like Joi or Validator.
- **Security**:
  - Implement security best practices such as input validation, secure headers, and rate limiting.
  - Use environment variables to manage configuration settings securely.

    Database: MongoDB
- **Schema Design**:
  - **Users Collection**: Store user details such as name, email, hashed password, and purchase history.
  - **Books Collection**: Store book details including title, author, genre, price, stock, and reviews.
  - **Orders Collection**: Track user orders, including book IDs, quantities, order status, and timestamps.
- **Interactions**:
  - **Mongoose**: Use Mongoose as an ODM (Object Data Modeling) library to define schemas and interact with MongoDB.
  - **CRUD Operations**: Perform Create, Read, Update, and Delete operations on the database collections.
  - **Aggregation**: Use MongoDB's aggregation framework for advanced data querying and reporting (e.g., total sales, popular books).
- **Indexes**:
  - Create indexes on frequently queried fields such as book titles, authors, and user emails to optimize search performance.
  - 4. Setup Instructions •Prerequisites: List software dependencies (e.g., Node.js, MongoDB). •Installation: Step-by-step guide to clone, install dependencies, and set up the environment variables. \\ TOPIC book store.
  - **Setup Instructions for Book Store**
  - **Prerequisites**
  - Before setting up the Book Store project, ensure you have the following software installed:
  - **Node.js**: JavaScript runtime environment. Download and install from nodejs.org.
  - **npm**: Node.js package manager, typically installed with Node.js.

- **MongoDB**: NoSQL database. Download and install from mongodb.com.
- **Installation**
- Follow these steps to clone the repository, install dependencies, and set up the environment variables:
- **Clone the Repository**: Open your terminal and run the following command to clone the project repository:
- sh
- git clone https://github.com/yourusername/book-store.git
- cd book-store
- **Install Dependencies**: Navigate to the project directory and install the required dependencies using npm:
- sh
- npm install
- **Set Up Environment Variables**: Create a .env file in the root directory of your project and add the necessary environment variables. For example:
- env
- PORT=3000
- MONGODB_URI=mongodb://localhost:27017/bookstore
- JWT_SECRET=your_jwt_secret
- **Run the Application**: Start the development server with the following command:
- sh
- npm start
- Your application should now be running locally at http://localhost:3000.
- **Summary of Commands:**
- Clone the repository:
- sh
- git clone https://github.com/yourusername/book-store.git
- cd book-store
- Install dependencies:
- sh
- npm install
- Set up environment variables:

- Env

- PORT=3000
- MONGODB_URI=mongodb://localhost:27017/bookstore
- JWT_SECRET=your_jwt_secret
- Run the application:
- sh
- npm start

folder Structure

Client: React Frontend

The frontend is organized to facilitate modular and maintainable code:

```
client/
├── public/
│   ├── index.html
│   └── ...
├── src/
│   ├── assets/
│   │   └── images/          # Images used in the project
│   ├── components/
│   │   └── Header.js        # Example of a header component
│   │   └── Footer.js        # Example of a footer component
│   │   └── ...              # Other reusable components
│   ├── pages/
│   │   └── HomePage.js      # Home page
│   │   └── BookDetails.js   # Book details page
│   │   └── ...              # Other pages
│   ├── services/
│   │   └── api.js           # API calls and service functions
│   ├── App.js               # Main application component
│   ├── index.js             # Entry point of the application
│   ├── routes.js            # Routing setup for the application
│   └── styles/              # CSS/SCSS files
└── package.json
```

Server: Node.js Backend

The backend is structured to keep the server code organized and maintainable:

```
server/
├── config/
│   └── db.js                # Database connection configuration
├── controllers/
│   └── authController.js    # Handles authentication logic
│   └── bookController.js    # Handles book-related logic
│   └── ...                  # Other controllers
├── middleware/
```

```
│   └── authMiddleware.js    # Authentication middleware
│   └── ...                  # Other middleware
├── models/
│   └── User.js              # User model schema
│   └── Book.js              # Book model schema
│   └── ...                  # Other models
├── routes/
│   └── authRoutes.js        # Authentication routes
│   └── bookRoutes.js        # Book-related routes
│   └── ...                  # Other routes
├── utils/
│   └── helpers.js           # Utility functions
├── app.js                   # Express app setup
├── server.js                # Server entry point
└── package.json
```

Running the Application

Starting the Frontend

Navigate to the `client` directory and start the frontend server:

sh

```
cd client
npm start
```

Starting the Backend

Navigate to the `server` directory and start the backend server:

sh

```
cd server
npm start
```

7. API Documentation

Here is an overview of the API endpoints exposed by the backend of the Book Store project:

User Endpoints

1. **Register User**
   - **Method:** POST
   - **Endpoint:** /api/users/register
   - **Parameters:**

json
```
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

   - **Example Response:**

json
```
{
  "message": "User registered successfully"
}
```

2. **Login User**
   - **Method:** POST

- **Endpoint:** /api/users/login
- **Parameters:**

```
json
{
  "email": "string",
  "password": "string"
}
```

- **Example Response:**

```
json
{
  "token": "jwt_token"
}
```

3. **Get User Profile**
   - **Method:** GET
   - **Endpoint:** /api/users/profile
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Example Response:**

```
json
{
  "id": "user_id",
  "name": "string",
  "email": "string"
}
```

Book Endpoints

1. **Get All Books**
   - **Method:** GET
   - **Endpoint:** /api/books
   - **Example Response:**

```
json
[
  {
    "id": "book_id",
    "title": "string",
    "author": "string",
    "genre": "string",
    "price": "number",
    "stock": "number"
  }
]
```

2. **Get Book by ID**
   - **Method:** GET
   - **Endpoint:** /api/books/:id
   - **Example Response:**

```
json
{
  "id": "book_id",
  "title": "string",
  "author": "string",
  "genre": "string",
  "price": "number",
  "stock": "number"
}
```

3. **Create New Book**
   - **Method:** POST
   - **Endpoint:** /api/books
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Parameters:**

```
json
{
  "title": "string",
  "author": "string",
  "genre": "string",
  "price": "number",
  "stock": "number"
}
```
   - **Example Response:**

```
json
{
  "message": "Book created successfully"
}
```
4. **Update Book**
   - **Method:** PUT
   - **Endpoint:** /api/books/:id
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Parameters:**

```
json
{
  "title": "string",
  "author": "string",
  "genre": "string",
  "price": "number",
  "stock": "number"
}
```
   - **Example Response:**

```
json
{
  "message": "Book updated successfully"
}
```
5. **Delete Book**
   - **Method:** DELETE
   - **Endpoint:** /api/books/:id
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Example Response:**

```
json
{
  "message": "Book deleted successfully"
}
```
Order Endpoints

1. **Create Order**
   - **Method:** POST
   - **Endpoint:** /api/orders
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Parameters:**

```
json
{
  "books": [
    {
      "bookId": "string",
      "quantity": "number"
    }
  ]
}
```
   - **Example Response:**

```
json
{
  "message": "Order placed successfully"
}
```
2. **Get User Orders**
   - **Method:** GET
   - **Endpoint:** /api/orders
   - **Headers:** Authorization: Bearer <jwt_token>
   - **Example Response:**

```
json
[
  {
    "orderId": "order_id",
    "books": [
      {
        "bookId": "string",
        "quantity": "number"
      }
    ],
    "orderStatus": "string",
    "createdAt": "date"
  }
]
```

8. Authentication and Authorization

Authentication and **Authorization** in the Book Store project are handled using JSON Web Tokens (JWT).

Authentication Flow

1. **User Registration:**
    - When a user registers, their information (name, email, and password) is saved to the database. The password is hashed before storage for security.
2. **User Login:**
    - Upon login, the user provides their email and password. The backend verifies the credentials. If valid, a JWT is generated and sent back to the user.

JSON Web Tokens (JWT)

- **Token Generation:**
    - After a successful login, a JWT is generated using a secret key. The token includes encoded user information and expiration time.
- **Token Structure:**
    - A JWT consists of three parts: Header, Payload, and Signature. For example:
    - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
- **Token Verification:**
    - For protected routes, the token is required in the Authorization header. The backend verifies the token to ensure the user is authenticated.

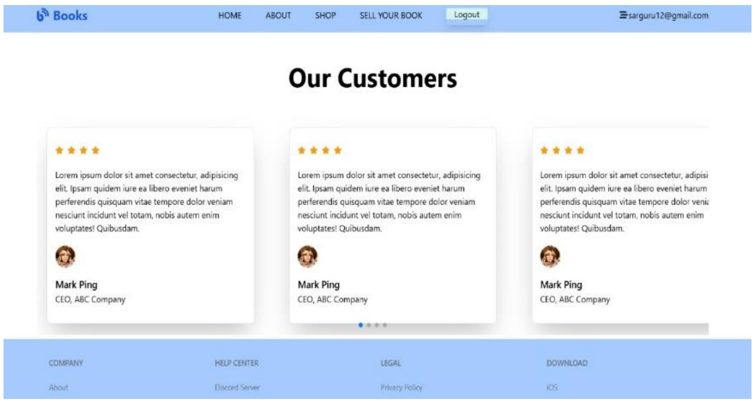
Middleware for Authorization

- **Authentication Middleware:**
    - Middleware verifies the presence and validity of the token. If valid, it allows access to protected routes; otherwise, it returns an error.

```
javascript
const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {
  const token = req.header('Authorization').split(' ')[1];
  if (!token) return res.status(401).send('Access Denied');

  try {
    const verified = jwt.verify(token, process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).send('Invalid Token');
  }
};
```



Our Customers

★★★★

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Ipsam quidem iure ea libero eveniet harum perferendis quisquam vitae tempore dolor veniam nesciunt incidunt vel totam, nobis autem enim voluptates! Quibusdam.

Mark Ping
CEO, ABC Company

★★★★

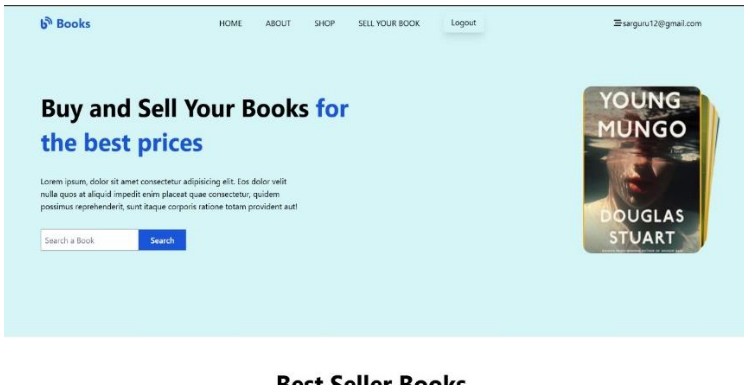
Lorem ipsum dolor sit amet consectetur, adipisicing elit. Ipsam quidem iure ea libero eveniet harum perferendis quisquam vitae tempore dolor veniam nesciunt incidunt vel totam, nobis autem enim voluptates! Quibusdam.

Mark Ping
CEO, ABC Company

★★★★

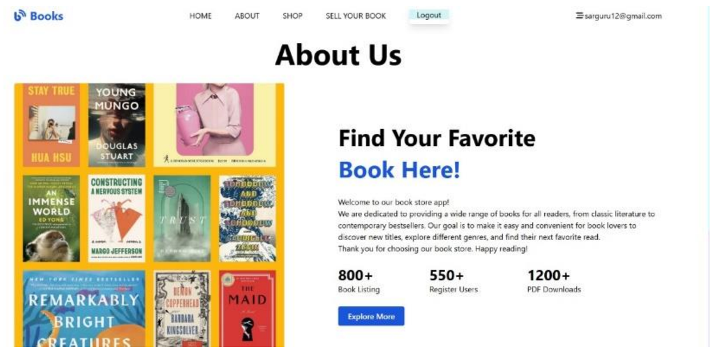
Lorem ipsum dolor sit amet consectetur, adipisi elit. Ipsam quidem iure ea libero eveniet harum perferendis quisquam vitae tempore dolor veni nesciunt incidunt vel totam, nobis autem enim voluptates! Quibusdam.

Mark Ping
CEO, ABC Company



Buy and Sell Your Books for the best prices

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Eos dolor velit nulla quos at aliquid impedit enim placeat quae consectetur, quidem possimus reprehenderit, sunt itaque corporis ratione totam provident aut!

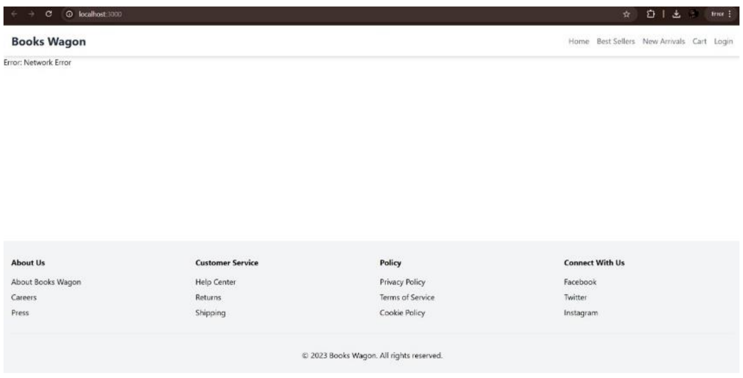


About Us

Find Your Favorite Book Here!

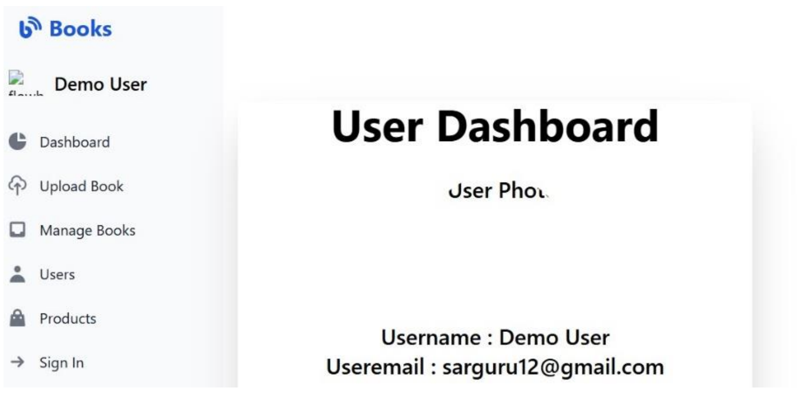
Welcome to our book store app!
We are dedicated to providing a wide range of books for all readers, from classic literature to contemporary bestsellers. Our goal is to make it easy and convenient for book lovers to discover new titles, explore different genres, and find their next favorite read.
Thank you for choosing our book store. Happy reading!

800+ Book Listing
550+ Register Users
1200+ PDF Downloads



Error: Network Error

© 2023 Books Wagon. All rights reserved.



Username : Demo User
Useremail : sarguru12@gmail.com

Testing

Testing Strategy:

1. **Unit Testing**:
   - Test individual components and functions to ensure they work as expected.
   - Use frameworks like Jest for JavaScript and React Testing Library for React components.
2. **Integration Testing**:
   - Test how different parts of the application work together.
   - Ensure that components interact correctly with the backend APIs.
3. **End-to-End (E2E) Testing**:
   - Simulate user interactions with the application.
   - Use tools like Cypress or Selenium to automate testing of user workflows.
4. **Manual Testing**:
   - Perform exploratory testing to catch any issues not covered by automated tests.
   - Test on different devices and browsers to ensure compatibility.

Testing Tools:

- **Jest**: For unit and integration tests.
- **React Testing Library**: For testing React components.
- **Cypress**: For end-to-end testing.
- **Postman**: For testing API endpoints manually.

11. Screenshots or Demo

Known Issues

1. **User Authentication Timeout**:
   - **Description**: Sometimes, user authentication tokens expire too quickly, causing users to log in again frequently.
   - **Workaround**: Extend token expiration time or implement refresh tokens.
2. **Search Functionality Lag**:
   - **Description**: The search feature may experience a slight delay in returning results, especially with a large database.
   - **Workaround**: Optimize search algorithms and consider indexing frequently searched fields.
3. **Mobile Responsiveness**:
   - **Description**: Certain pages may not render correctly on smaller screens.
   - **Workaround**: Improve CSS media queries and test on various mobile devices.
4. **Payment Gateway Integration**:
   - **Description**: Occasionally, payment processing fails due to timeout errors.
   - **Workaround**: Implement retry logic and provide clear error messages to users.
5. **Database Connection Issues**:
   - **Description**: Sporadic connection issues with MongoDB can lead to temporary unavailability.
   - **Workaround**: Implement robust retry and failover mechanisms.
6. **Review Submission**:
   - **Description**: Users occasionally face issues when submitting reviews, with some submissions not being saved.
   - **Workaround**: Ensure form validation and proper error handling on both client and server sides.

13. Future Enhancements

Potential future features or improvements for the Book Store project:

1. **Enhanced Recommendation System**:
   - Implement machine learning algorithms to provide more personalized book recommendations based on user behavior and preferences.
2. **Advanced Analytics Dashboard**:

    - Develop a comprehensive dashboard for admin users to monitor sales trends, user engagement, and inventory levels in real-time.

3. **Multilingual Support**:
    - Add support for multiple languages to cater to a broader audience and improve accessibility.
4. **Social Sharing Features**:
    - Enable users to share their favorite books and reviews on social media platforms directly from the website.
5. **Mobile Application**:
    - Develop a dedicated mobile application for iOS and Android to provide a seamless mobile experience.
6. **Subscription Model**:
    - Introduce a subscription service for premium users offering exclusive discounts, early access to new releases, and other perks.
7. **Enhanced Security Measures**:
    - Implement advanced security features such as two-factor authentication (2FA) and biometric login.
8. **Integration with E-Book Platforms**:
    - Allow users to purchase and download e-books, and integrate with popular e-book readers.
9. **User-Generated Content**:
    - Enable users to create and share book lists, reviews, and recommendations within the community.
10. **Gamification**:
    - Introduce gamification elements such as badges, achievements, and leaderboards to enhance user engagement.