Full Stack Development with MERN Project Documentation format

TITLE: BOOK STORE

TEAM MEMBERS:

Y.Jayaraman
D.flington
M.Yokeshwar
K.Pratheep

PROJECT DESCRIPTION:

Welcome to the literary haven of the digital age-introducing our revolutionary Book-Store Application, a masterpiece crafted with precision using the powerful MERN (MongoDB, Express.js, React, Node.js) Stack. Immerse yourself in a world where the love for reading converges seamlessly with cutting-edge technology, redefining the way bibliophiles explore, discover, and indulge in their literary pursuits.

Tailored for the modern book enthusiast, our MERN-based Book-Store Application seamlessly blends robust functionality with an intuitive user interface. From the joy of discovering new releases to the nostalgia of revisiting timeless classics, our platform promises an immersive reading experience customized to cater to your literary preferences.

Fueling the backbone of our application is MongoDB, ensuring a scalable and efficient database infrastructure that facilitates swift access to an extensive collection of literary works. Express.js, with its streamlined web application framework, establishes a responsive and efficient server, while Node.js ensures high-performance, non-blocking I/O operations-resulting in a seamless and enjoyable user experience.

At the heart of our Book-Store Application lies React, a dynamic and feature-rich JavaScript library. Dive into a visually enchanting and interactive interface where every click, search, and book selection feels like a literary journey. Whether you're exploring on a desktop, tablet, or smartphone, our responsive design ensures a consistent and delightful experience across all devices.

Say farewell to the constraints of traditional bookstores and embrace a new era of possibilities with our MERN Stack Book-Store Application. Join us as we transform how you connect with literature, making the discovery of your next favorite read an effortless and enriching experience. Get ready to turn the digital pages of a new chapter in reading, where every book is just a click away, and the literary world is at your fingertips. It's time to open the door to a future where the love for books meets the convenience of modern technology.

SCENARIO BASED CASE STUDY:

Sarah is an avid reader with a passion for exploring new genres and authors. However, her busy schedule often leaves her with limited time to visit physical bookstores. Sarah is looking for a solution that allows her to discover and purchase books conveniently, without compromising her reading preferences or the joy of browsing through a bookstore.

User Registration and Authentication: Allow users to register accounts securely, log in, and authenticate their identity to access the book store platform.

Book Listings: Display a comprehensive list of available books with details such as title, author, genre, description, price, and availability status.

Book Selection: Provide users with options to select their preferred books based on factors like genre, author, ratings, and popularity.

Purchase Process: Allow users to add books to their cart, specify quantities, and complete purchases securely. Upon successful completion, an order is generated, and the inventory is updated accordingly.

Order Confirmation: Provide users with a confirmation page or notification containing details of their order, including book information, total price, and order ID.

Order History: Allow users to view their past and current orders, providing options to track shipments, review purchased books, and rate their shopping experience.

<u>**TECNICAL ARCHITECTURE:**</u>

User Interface: The user interface will serve as the platform where customers can browse books, search for specific titles or authors, read book descriptions, and make purchases. It should be intuitive and user-friendly, enabling easy navigation and exploration of available books.

Web Server: The web server hosts the user interface of the book store app, serving dynamic web pages to users and ensuring a seamless browsing and shopping experience.

API Gateway: Similar to the original architecture, the API gateway will serve as the central entry point for client requests, directing them to the relevant services within the system. It will handle requests such as fetching book information, processing orders, and managing user accounts.

Authentication Service: The authentication service manages user authentication and authorization, ensuring secure access to the book store app and protecting sensitive user information during the browsing and purchasing process.

Database: The database stores persistent data related to books, including information such as titles, authors, genres, descriptions, prices, and availability. It also stores user profiles, purchase history, and other essential entities crucial to the book store app.

View Books: This feature allows users to browse through the available books. They can explore different categories and genres to discover books of interest.

Category Selection: Users can select specific categories or genres to filter and refine their book browsing experience, making it easier to find books tailored to their preferences.

Inventory Management Service: This service manages information about available books, including their availability, stock levels, and ratings. It ensures efficient management of the book inventory and seamless integration with the browsing and purchasing process.

Order Management Service: This service facilitates the ordering process, allowing users to add books to their cart, specify quantities, and complete purchases securely. It also handles order tracking and status updates in real-time.

<u>**BACKEND DEVELOPMENT:**</u>

User Authentication:

Create routes and middleware for user registration, login, and logout.
Set up authentication middleware to protect routes that require user authentication.

Define API Routes:

Create separate route files for different API functionalities such as users orders, and authentication.
Define the necessary routes for listing products, handling user registration and login,managing orders, etc.
Implement route handlers using Express.js to handle requests and interact with the database.

Implement Data Models:

Define Mongoose schemas for the different data entities like products, users, and orders.
Create corresponding Mongoose models to interact with the MongoDB database.
Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

User Authentication:

Create routes and middleware for user registration, login, and logout.
Set up authentication middleware to protect routes that require user authentication.

Error Handling:

Implement error handling middleware to catch and handle any errors that occur during the API requests.
Return appropriate error responses with relevant error messages and HTTP status codes.

FRONTEND DEVELOPMENT:

1. Setup React Application:

• Create React application.

• Configure Routing.

• Install required libraries.

2. Design UI components:

• Create Components.

• Implement layout and styling.

• Add navigation.

3. Implement frontend logic:

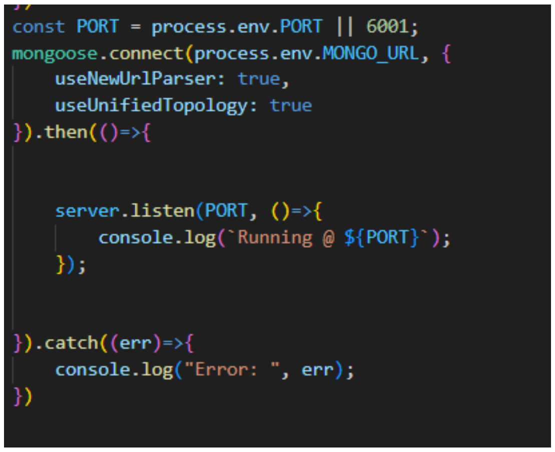
• Integration with API endpoints.

• Implement data binding.

DATABASE:

Connect database to backend:

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.

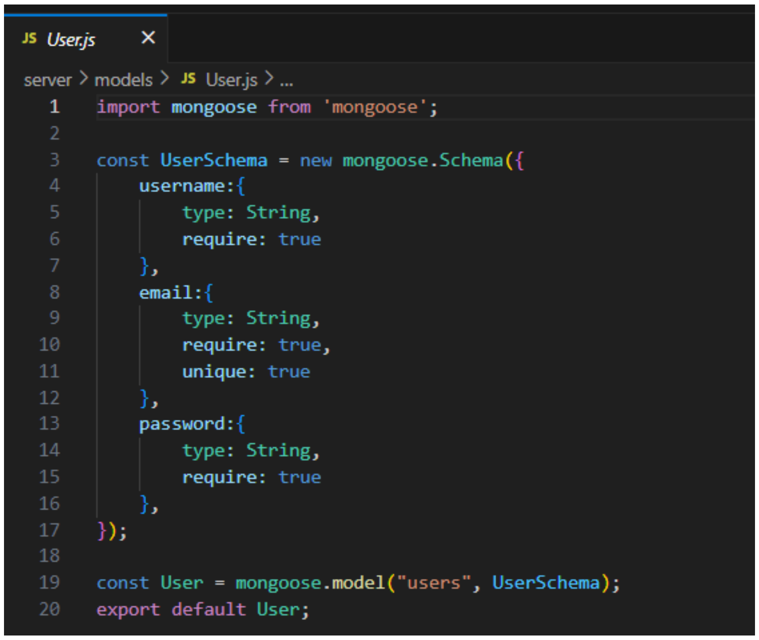


```
const PORT = process.env.PORT || 6001;
mongoose.connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{


    server.listen(PORT, ()=>{
        console.log(`Running @ ${PORT}`);
    });


}).catch((err)=>{
    console.log("Error: ", err);
})
```

Configure Schema:

Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas.

The Schemas for this application look alike to the one provided below.



```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
    username:{
        type: String,
        require: true
    },
    email:{
        type: String,
        require: true,
        unique: true
    },
    password:{
        type: String,
        require: true
    },
});

const User = mongoose.model("users", UserSchema);
export default User;
```

<u>GITHUB LINK:-</u>

IMPLEMENTATION:

https://github.com/FLINGTON/Book-Store/blob/main/implementation.pdf

FRONTEND:

https://github.com/FLINGTON/Book-Store/blob/main/Frontend.zip

BACKEND:

https://github.com/FLINGTON/Book-Store/blob/main/Backend.zip

DEMO LINK:-

DRIVE LINK:

https://drive.google.com/file/d/192tYmqc3uqHNQmq23c4W-ScVyhSz9EUX/view

GITHUB:

https://github.com/FLINGTON/Book-Store/blob/main/Demo%20video-book%20store.mp4