Documentation for Book Store using MongoDB

Table of Contents

1. Introduction

2. System Requirements

3. Prerequisite

4. Architecture

5. ER Diagram

6. Project Structure

7. Application Flow

8. Project Setup and Configuration

9. Project Implementation and Execution

10. Conclusion

1. Introduction

In this project, we develop a Book Store application using MongoDB as the database backend. MongoDB is a NoSQL database known for its scalability and flexibility in handling large volumes of unstructured data. This application provides a platform for users to view and purchase books, along with an administrative interface to manage books and orders. By using MongoDB, we can easily handle diverse book data (e.g., title, author, price) and user activity, all in a schemaless format.

1. System Requirements

To run the Book Store application successfully, the following system requirements must be met

Hardware

Processor: 2 GHz or higher

RAM: 4 GB minimum (8 GB recommended)

Storage: 1 GB of free space for application and database storage

Software:

Operating System: Windows, macOS, or Linux

MongoDB: Version 4.x or higher

Node.js: Version 14.x or higher

npm: Version 6.x or higher

Web Browser: Latest version of Chrome, Firefox, or Safari

Development Tools:

IDE/Editor: Visual Studio Code or any preferred code editor

Postman for API testing

1. Prerequisite

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

MongoDB Setup: Ensure that MongoDB is installed and running on your local machine or on a cloud service (e.g., MongoDB Atlas).

Node.js and npm: Install Node.js and npm (node package manager) for setting up the application server.

Version Control: Git should be installed for version control and collaboration.

Knowledge of JavaScript/Node.js: The application will use JavaScript with Node.js for the backend.

1. Architecture

The Book Store application follows a ClientServer architecture with a RESTful API connecting the frontend (user interface) to the backend (server and database).

Key components of the architecture:

Frontend (Clientside):

A webbased interface for users to browse, purchase, and review books.

Developed using HTML, CSS, and JavaScript.

Backend (Serverside):

Node.js server hosting the RESTful API.

Express.js to manage routes and API endpoints.

MongoDB to store and retrieve bookrelated data, user profiles, and orders.

Database (MongoDB):

MongoDB collections for books, users, and orders.

Data stored in JSONlike documents (BSON format).

Flow:

1. User Interaction: The user interacts with the frontend through a web browser.

2. API Requests: The frontend sends HTTP requests (GET, POST, PUT, DELETE) to the backend.

3. Database Operations: The backend interacts with MongoDB to perform CRUD (Create, Read, Update, Delete) operations.

4. Response to User: The backend sends a response back to the frontend for rendering.

1. ER Diagram

The EntityRelationship (ER) Diagram for the Book Store application includes the following entities:

Book:

Attributes: \_bookID\_, \_title\_, \_author\_, \_genre\_, \_price\_, \_description\_, \_stock\_

User:

Attributes: \_userID\_, \_name\_, \_email\_, \_password\_, \_address\_, \_phone\_

Order:

Attributes: \_orderID\_, \_userID\_, \_bookID\_, \_orderDate\_, \_quantity\_, \_totalPrice\_, \_status\_

Relationship:

A user can place multiple orders (onetomany).

An order can contain multiple books (manytomany).

Below is a simple ER Diagram representation:

```

++ ++ ++

| User | | Order | | Book |

++ ++ ++

| userID (PK)| < | orderID | > | bookID (PK)|

| name | | userID (FK)| | title |

| email | | bookID (FK)| | author |

| phone | | quantity | | genre |

| address | | totalPrice| | price |

++ ++ ++

```

1. Project Structure

The directory structure for the Book Store project is as follows:

```

bookstore/

│

├── backend/

│ ├── config/ MongoDB configuration and environment variables

│ ├── controllers/ API route handlers

│ ├── models/ MongoDB schemas (Book, User, Order)

│ ├── routes/ API routes (user routes, book routes)

│ ├── app.js Entry point of the server (Express app)

│ └── server.js Server configuration

│

├── frontend/

│ ├── index.html Landing page (home)

│ ├── styles/ CSS files for styling

│ ├── scripts/ JavaScript files for frontend logic

│ └── assets/ Images and other assets

│

├── .gitignore Git ignore file

├── package.json Project metadata and dependencies

└── README.md Project documentation

```

1. Application Flow

1. User Visits Homepage: The user accesses the homepage to browse available books.

2. Login/Register: If not logged in, the user can either log in or register.

3. Browse Books: The user can browse books by category, search by title/author, or filter by price.

4. Add to Cart: The user adds books to the shopping cart.

5. Checkout: The user proceeds to checkout and places an order.

6. Payment (Optional): Implement a simple payment interface (e.g., using a dummy payment gateway).

7. Order Confirmation: The user receives an order confirmation and can track order status.

1. Project Setup and Configuration

To set up the Book Store project locally:

1. Clone the Repository:

```bash

git clone https://github.com/yourusername/bookstore.git

cd bookstore

```

2. Install Dependencies:

Navigate to both `backend` and `frontend` directories and install dependencies.

```bash

cd backend

npm install

```

3. MongoDB Setup:

Set up MongoDB locally or use MongoDB Atlas for cloud hosting.

Configure the database connection in `backend/config/db.js`.

4. Start the Server:

To start the backend server:

```bash

cd backend

npm start

```

5. Run Frontend:

Open `index.html` in a browser or set up a local web server (e.g., using `liveserver`).

1. Project Implementation and Execution

The backend is built with Node.js and Express.js. Each route handles specific API requests such as retrieving books, adding books to the cart, and placing orders.

MongoDB is used to store and query data about books, users, and orders.

The frontend is a simple HTML/CSS/JavaScript interface that interacts with the backend API to display books and handle the user flow (e.g., adding books to cart, user login).

1. Conclusion

The Book Store application demonstrates how a simple ecommerce platform can be built using MongoDB for storing and retrieving data. This architecture allows for flexibility and scalability as the system grows. By using MongoDB's documentbased structure, we can efficiently manage book data, user profiles, and orders. This project highlights the power of NoSQL databases for realworld applications that require high performance and scalability.