MIT World Peace University

Full Stack Development
Third Year B. Tech, Semester 5

Developing a FSD Application using MERN Stack.
Novel Tea Library

Lab Assignment 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

September 22, 2023

Contents

1 Aim

Develop a full stack web application using MERN stack to perform CRUD operations.

2 Objectives

- To develop full-stack web projects using the MERN stack.
- To learn database connectivity using fetch api.
- To perform insert, update, delete and search operations on database.

3 Problem Statement

Student can create a React form or use existing/ implemented HTML form for Library Management System with the fields mentioned: Book name, ISBN No, Book title, Author name, Publisher name and perform following operations

1. Insert Book details -Book name, ISBN No, Book title, Author name, Publisher name
2. Delete the Book records based on ISBN No
3. Update the Book details based on ISBN No- Example students can update wrong entered book details based on searching the record with ISBN No.
4. Display the Updated Book details or View the Book Details records in tabular format.

4 Theory

4.1 The MERN Stack

Definition 1 *The MERN stack is a JavaScript stack designed to streamline the development process. MERN comprises four open-source components: MongoDB, Express, React, and Node.js. These components together create a comprehensive framework for developers. The MERN stack is known for its ability to simplify and enhance the development experience.*

Figure 1: The MERN Stack

4.2 Features

1. **Full-Stack JavaScript:** MERN is entirely built on JavaScript, making it a full-stack solution. This allows developers to use a single programming language for both the front-end (React) and back-end (Node.js).

2. **Modularity:** Each component of the MERN stack (MongoDB, Express, React, and Node.js) is modular and can be replaced or extended with other libraries or frameworks to suit specific project requirements.

3. **Reusability:** React components can be reused across the application, improving code maintainability and reducing development time.

4. **Real-Time Updates:** MERN applications can easily incorporate real-time features using technologies like WebSockets or libraries like Socket.io.

5. **Scalability:** MERN applications can be scaled horizontally and vertically to handle increased user loads and growing data requirements.

4.3 Advantages

1. **Rapid Development:** MERN's unified JavaScript ecosystem allows for quicker development, as developers can work seamlessly across the entire stack.

2. **Community and Libraries:** The MERN stack benefits from a large and active developer community, along with numerous libraries and packages available through npm (Node Package Manager).

3. **Isomorphic Applications:** MERN allows for isomorphic or universal JavaScript applications, where code can run on both the server and client. This enhances SEO and improves initial page load times.

4. **Single-Page Applications (SPAs):** React's component-based architecture facilitates the development of SPAs, resulting in a smooth and interactive user experience.

5. **Flexibility:** Developers have the flexibility to choose from a wide range of libraries, tools, and plugins to tailor the stack to their specific project requirements.

4.4 History and Significance

1. **Origin:** The MERN stack emerged as a response to the growing popularity of JavaScript as a server-side language. It brings together key technologies, with React introduced by Facebook, Node.js by Ryan Dahl, Express by TJ Holowaychuk, and MongoDB as a NoSQL database.

2. **Significance:** MERN has become a prominent stack for building web applications and APIs due to its versatility and efficiency. It is widely used for creating both small-scale projects and large-scale applications.

3. **Continued Development:** The MERN stack continues to evolve with updates to individual components, the introduction of new libraries, and the development of best practices. It remains a popular choice in the web development community.

4.5 Components of the Mern Stack

4.5.1 MongoDB

MongoDB is a NoSQL database that stores data in JSON-like documents. It is an open-source, cross-platform, document-oriented database written in C++. MongoDB is a NoSQL database that stores data in JSON-like documents. It is an open-source, cross-platform, document-oriented database written in C++.

Figure 2: MongoDB

Figure 3: Sample from MongoDB

4.5.2 Express JS

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Figure 4: Express JS

4.5.3 React JS

React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

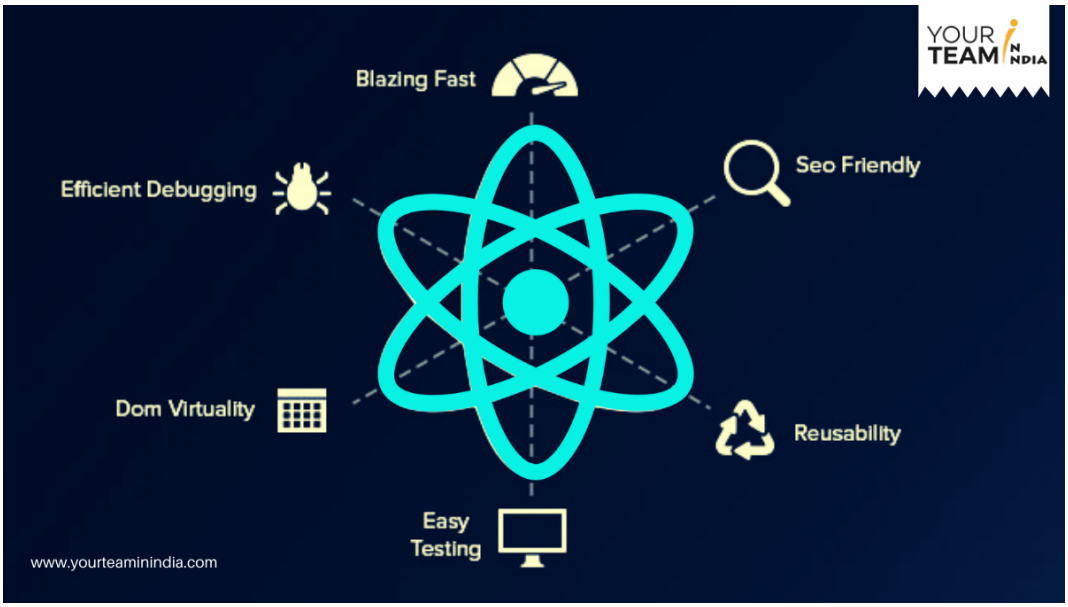

Figure 5: React Js Features

4.5.4 Node JS

Node is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Figure 6: Node JS

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Brave Browser (Chromium v117.0.5938.88.)

6 Input and Output

1. A Library Interface called 'Novel Tea' Library was set up on the front end usign React. Its Back end was then written in PHP, and the database was set up using MongoDB.
2. The Front end was hosted on another locolhost port, while the php server was hosted using httpd, and default php server.
3. The Front end was then connected to the backend using the simple FETCH API or Axios calls, and the data was sent to the backend using JSON. The responses were also sent from the backend to the Frontend using JSON.
4. The Features of the Library include, and therefore extend to the backend:
   (a) Adding a Book to the Library
   (b) Removing a Book from the Library
   (c) Updating a Book in the Library
   (d) Searching for a Book in the Library
   (e) Getting all the Books from the library to display.
5. These features are demonstrated in the screenshots below.

7 Screenshots

7.1 React Frontend

Figure 7: The Home Page of the Novel Tea Library

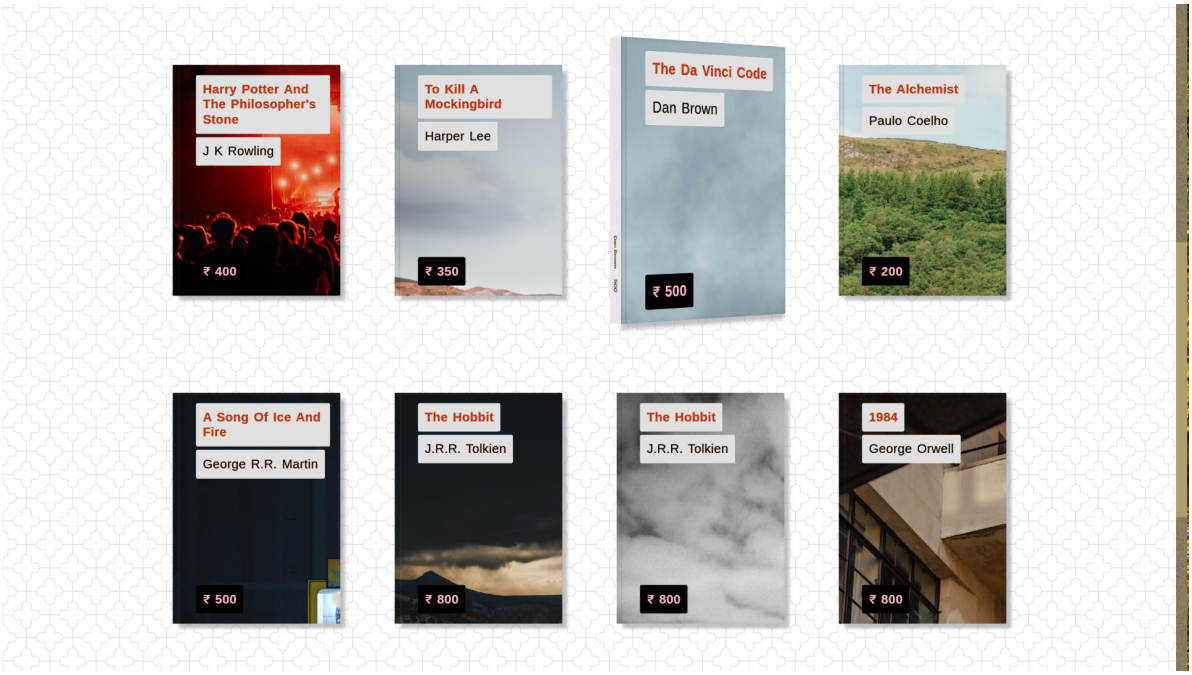

Figure 8: The Books retreived from /getbooks.php and shown on the Frontend

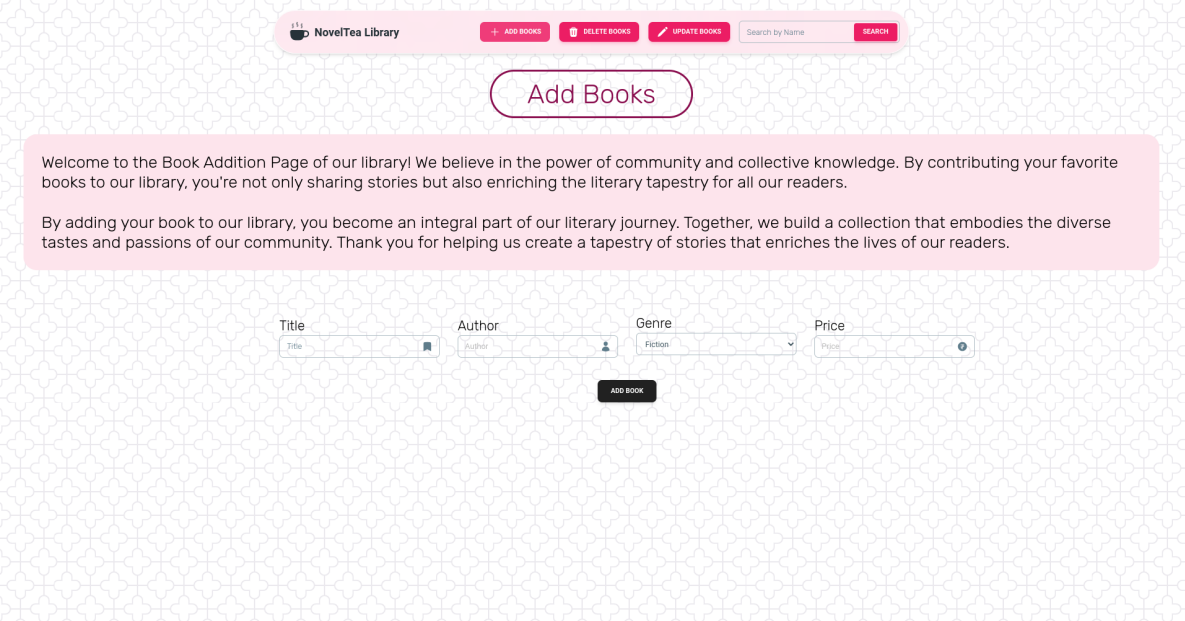

Figure 9: The Add Page

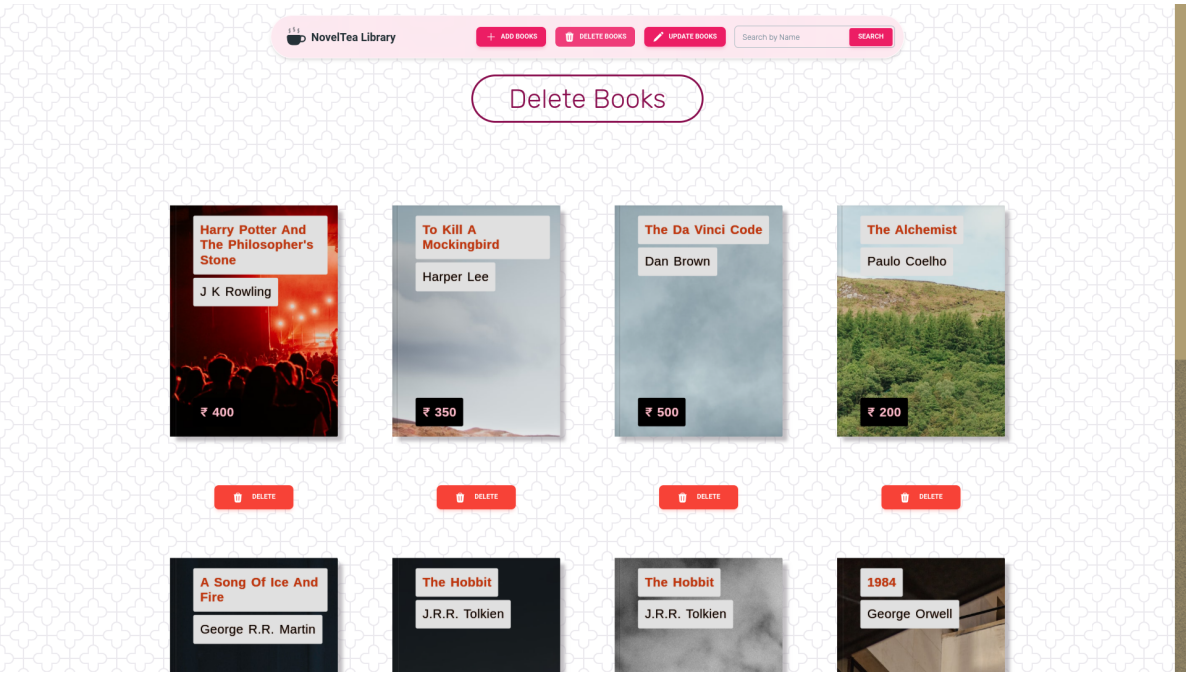

Figure 10: The Delete Page

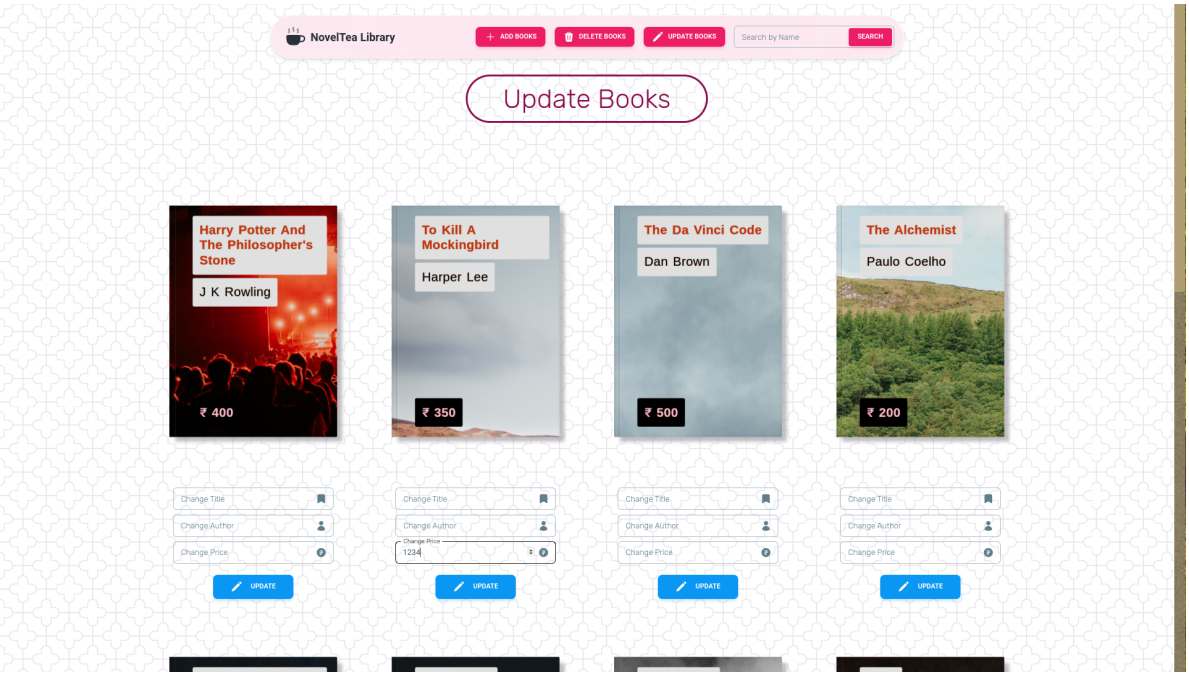

Figure 11: The Update Page, where any field can be updated

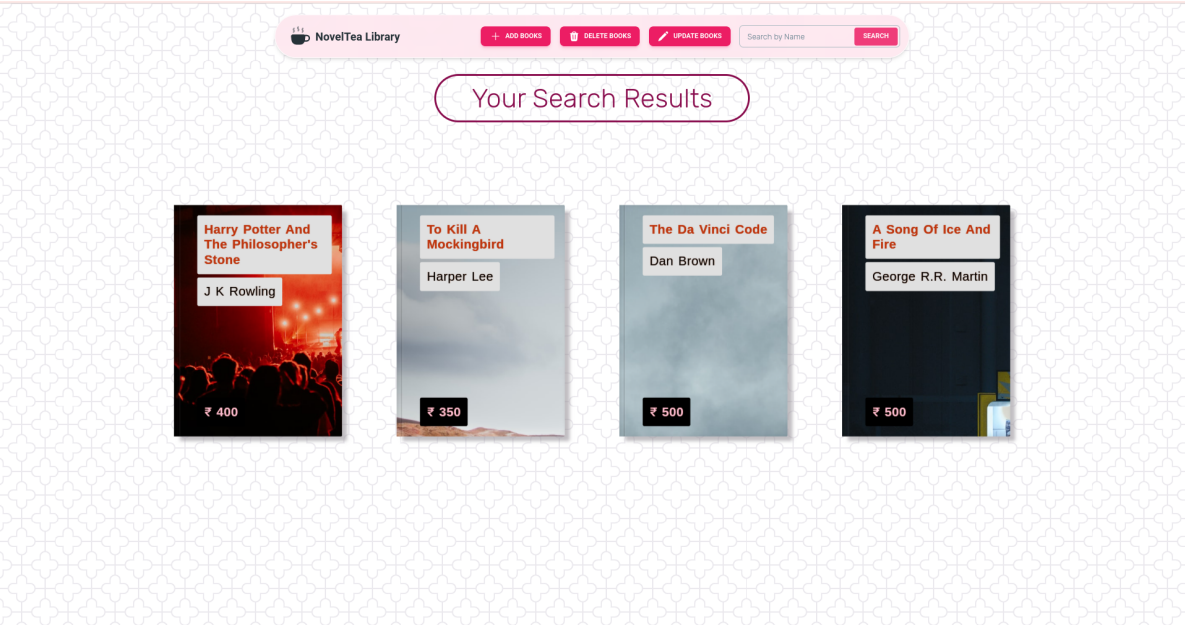

Figure 12: The Search Page showing results of search term "H"

7.2 Node and Express Backend

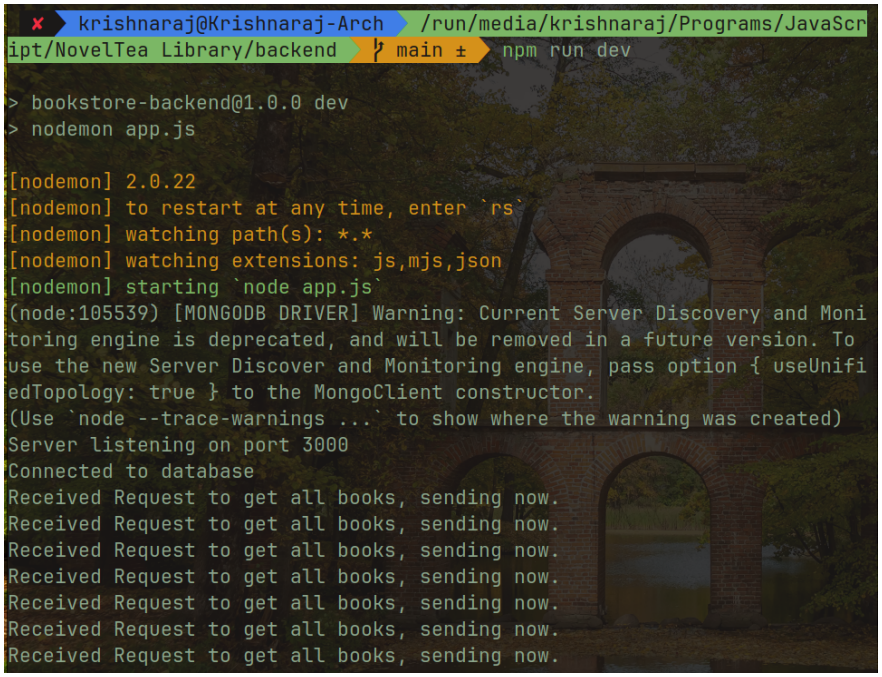

Figure 13: The Node and Express Backend server running. It is listening on port 3000.

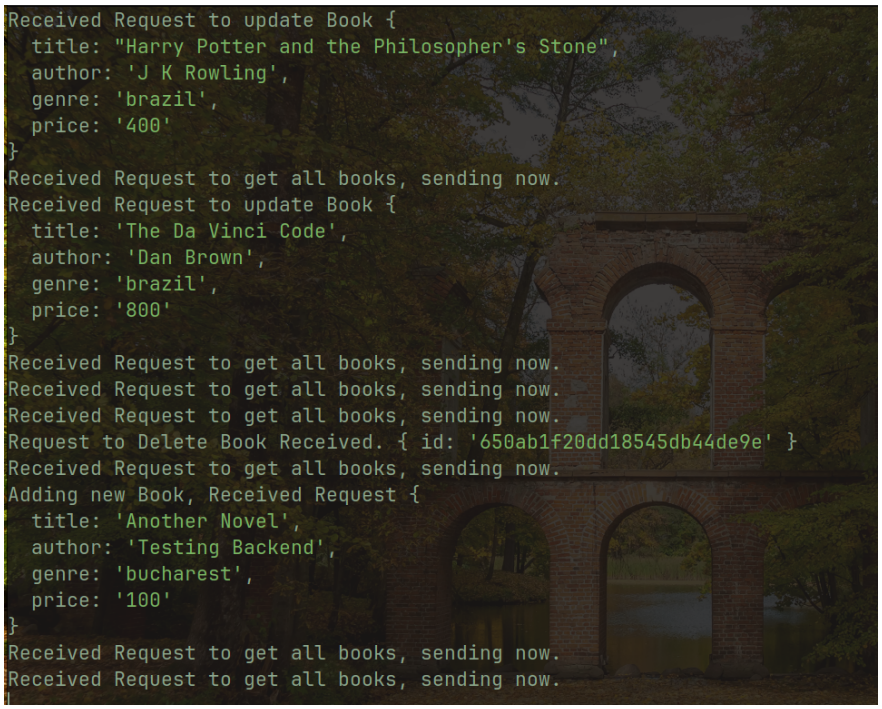

Figure 14: The Node and Express Backend server running and serving request.

7.3 MongoDB Database

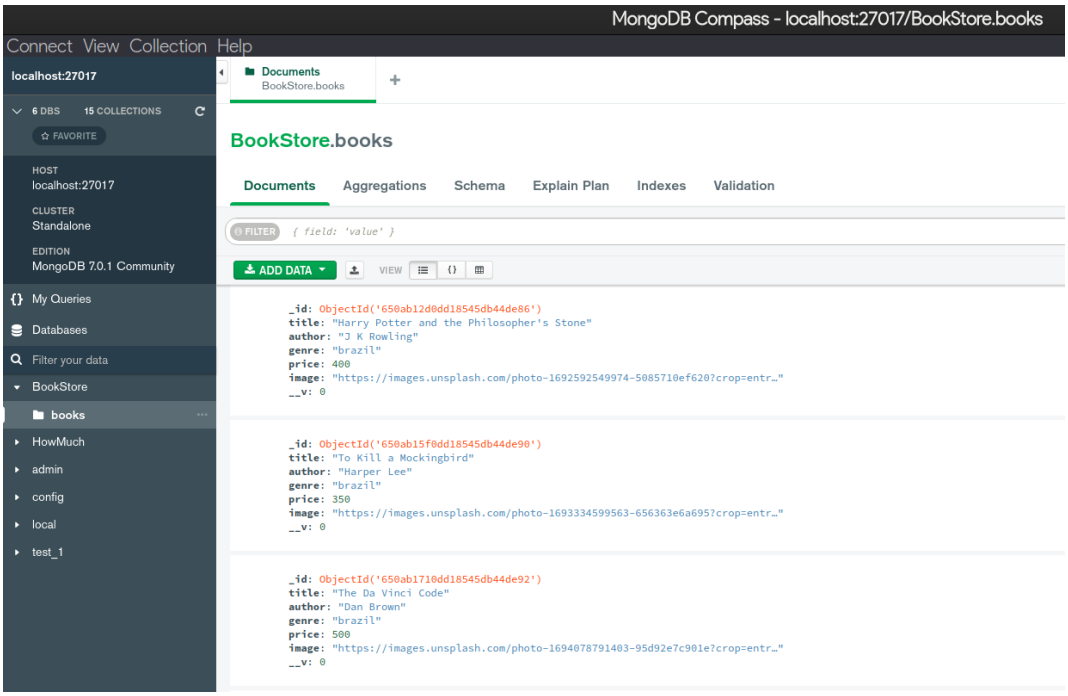

Figure 15: The MongoDB Compass showing the database.

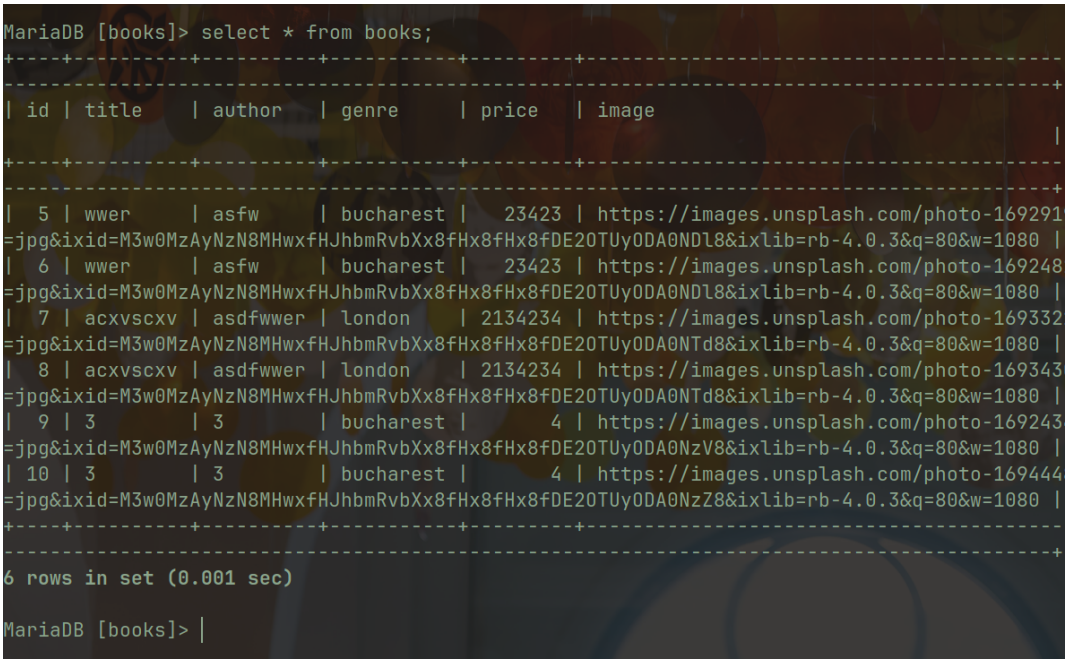

Figure 16: The MongoDB Database

8 Code

```
{
  "name": "bookstore-backend",
  "version": "1.0.0",
  "description": "Backend for the Bookstore project",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "dependencies": {
    "axios": "^1.5.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongoose": "^5.13.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.12"
  }
}
```

Listing 1: package.json

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const cors = require("cors");
const setApiRoutes = require("./routes/api");

const app = express();

// Set up middleware
app.use(bodyParser.json());
app.use(cors());

// Set up API routes
app.use(setApiRoutes);

// Connect to database
mongoose
  .connect("mongodb://localhost/BookStore", { useNewUrlParser: true })
  .then(() => {
    console.log("Connected to database");
  })
  .catch((error) => {
    console.error("Error connecting to database:", error);
  });

// Start server
const port = process.env.PORT || 3000;
app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Listing 2: app.js managing the app.

```
const express = require("express");
const router = express.Router();
const BooksController = require("../controllers/books");

```

```
const booksController = new BooksController();

router.get("/books", booksController.getAllBooks);
router.get("/", (req, res) => {
  res.send("Hello World!");
});
router.get("/books/:id", booksController.getBookById);
router.post("/books", booksController.createBook);
router.put("/books/:id", booksController.updateBook);
router.delete("/books/:id", booksController.deleteBook);

module.exports = router;
```

Listing 3: api.js defining the routes.

```
const mongoose = require("mongoose");

const bookSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  author: {
    type: String,
    required: true,
  },
  genre: {
    type: String,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
  image: {
    type: String,
  },
});

const Book = mongoose.model("Book", bookSchema);

module.exports = Book;
```

Listing 4: Book.js interacting with MongoDB

```
const Book = require("../models/Book");
const axios = require("axios");

async function getRandomImageUrl() {
  try {
    const response = await axios.get("https://api.unsplash.com/photos/random", {
      headers: {
        Authorization: "Client-ID UBAXKPq5WgOxU9z7XjLZrt8B0hayG2dv8gh_vyGQg-0",
      },
    });

    return response.data.urls.regular;
  } catch (error) {
    console.error(error);
```

```
    return null;
  }
}
class BooksController {
  async getAllBooks(req, res) {
    console.log("Received Request to get all books, sending now. ");
    try {
      const books = await Book.find();
      res.status(200).json(books);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }

  async getBookById(req, res) {
    try {
      const book = await Book.findById(req.params.id);
      if (!book) {
        return res.status(404).json({ message: "Book not found" });
      }
      res.status(200).json(book);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }

  async createBook(req, res) {
    console.log("Adding new Book, Received Request", req.query);
    const image = await getRandomImageUrl();
    const book = new Book({
      title: req.query.title,
      author: req.query.author,
      genre: req.query.genre,
      price: req.query.price,
      image: image,
    });

    try {
      const newBook = await book.save();
      res.status(201).json(newBook);
    } catch (error) {
      res.status(400).json({ message: error.message });
    }
  }

  async updateBook(req, res) {
    console.log("Received Request to update Book", req.query);
    try {
      const book = await Book.findById(req.params.id);
      if (!book) {
        return res.status(404).json({ message: "Book not found" });
      }

      book.title = req.query.title || book.title;
      book.author = req.query.author || book.author;
      book.genre = req.query.genre || book.genre;
      book.price = req.query.price || book.price;

      const updatedBook = await book.save();
```

```
74        res.status(200).json(updatedBook);
75      } catch (error) {
76        res.status(500).json({ message: error.message });
77      }
78    }
79
80    async deleteBook(req, res) {
81      console.log("Request to Delete Book Received.", req.params);
82      try {
83        const book = await Book.findById(req.params.id);
84        if (!book) {
85          return res.status(404).json({ message: "Book not found" });
86        }
87
88        await book.remove();
89        res.status(200).json({ message: "Book deleted successfully" });
90      } catch (error) {
91        res.status(500).json({ message: error.message });
92      }
93    }
94  }
95
96  module.exports = BooksController;
```

Listing 5: books.js serving routes.

9 Conclusion

Thus, we have successfully created a MERN Stack Application, and performed CRUD operations on it. The database is hosted on MongoDB Atlas. The frontend is hosted at here and the backend is hosted at locally. on its Github repo.

10 FAQ

1. *What makes MERN stack the fastest growing tech stack?*

   (a) **MERN is a full-stack JavaScript solution:** MERN includes four powerful technologies that work together to build dynamic web applications. These technologies are MongoDB, Express, React, and Node.js. All of these technologies are based on JavaScript, which makes it easier for developers to work on the entire stack within the same language. This also makes it easier to switch between front-end and back-end development.

   (b) **MERN is open-source:** All of the technologies in the MERN stack are open-source, which means that they are free to use and can be modified by developers. This allows developers to customize the stack to suit their specific project requirements.

   (c) **MERN is flexible and extensible:** MERN is modular and can be extended with other libraries and frameworks to suit specific project requirements. Developers can also choose from a wide range of tools and packages to customize the stack.

   (d) **MERN is easy to learn:** MERN is based on JavaScript, which is one of the most popular programming languages. This makes it easier for developers to learn the stack and build applications using the MERN stack.

   (e) **MERN is supported by a large community:** MERN has a large and active developer community, which provides support and resources for developers. This makes it easier for developers to learn the stack and build applications using the MERN stack.

   (f) **MERN is scalable:** MERN applications can be scaled horizontally and vertically to handle increased user loads and growing data requirements.

   (g) **MERN is cross-platform:** MERN applications can be deployed on any platform, including Windows, Linux, and macOS.

   (h) **MERN is secure:** MERN applications are secure by default, as they are built on top of secure technologies like MongoDB and Node.js.