

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	1
2 Objectives	1
3 Problem Statement	1
4 Theory	1
4.1 The MERN Stack	1
4.2 Features	2
4.3 Advantages	2
4.4 History and Significance	2
4.5 Components of the Mern Stack	3
4.5.1 MongoDB	3
4.5.2 Express JS	4
4.5.3 React JS	4
4.5.4 Node JS	5
5 Platform	5
6 Input and Output	5
7 Screenshots	6
7.1 React Frontend	6
7.2 Node and Express Backend	9
7.3 MongoDB Database	10
8 Code	10
9 Conclusion	14
10 FAQ	15

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.*

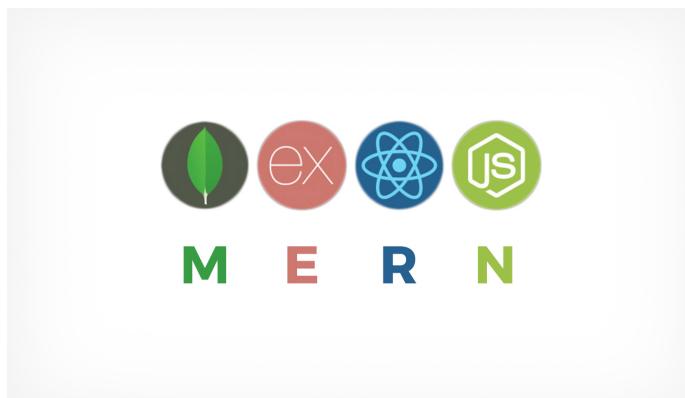


Figure 1: The MERN Stack

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.

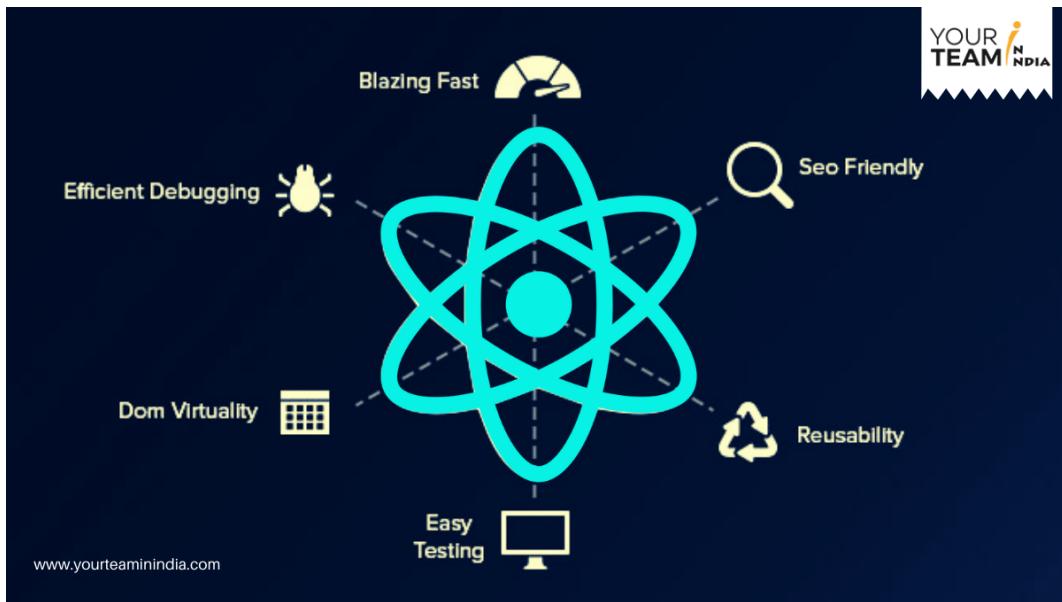


Figure 5: React Js Features

4.5.4 Node JS

Node.js 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.js 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 using 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 localhost 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

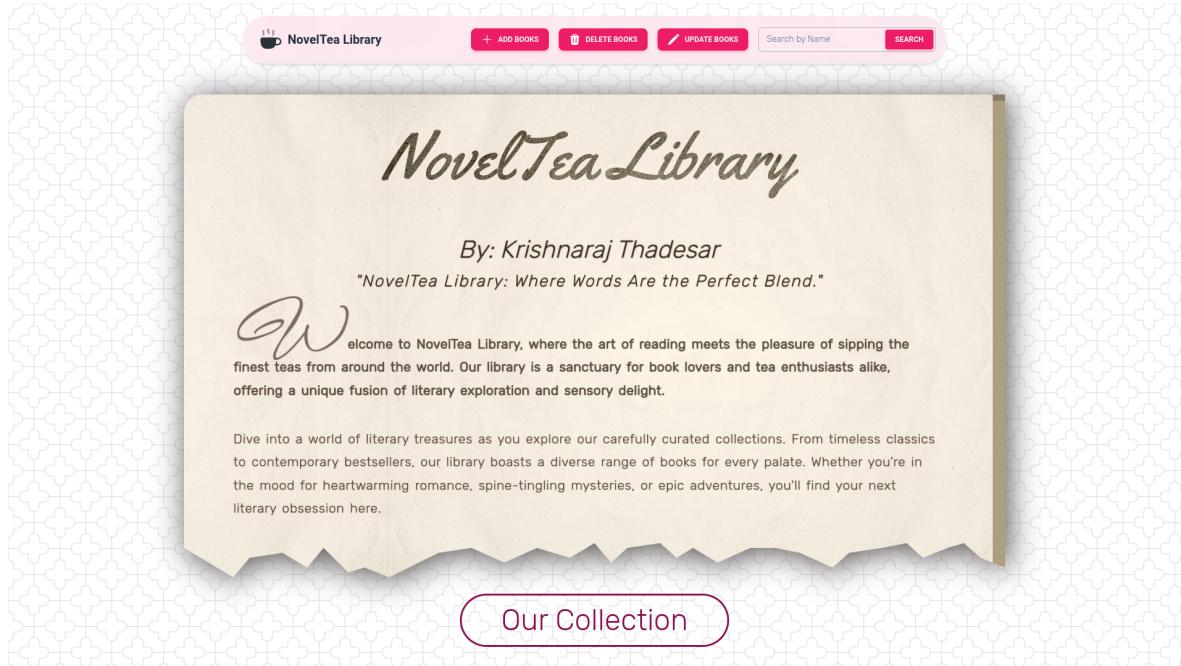


Figure 7: The Home Page of the Novel Tea Library

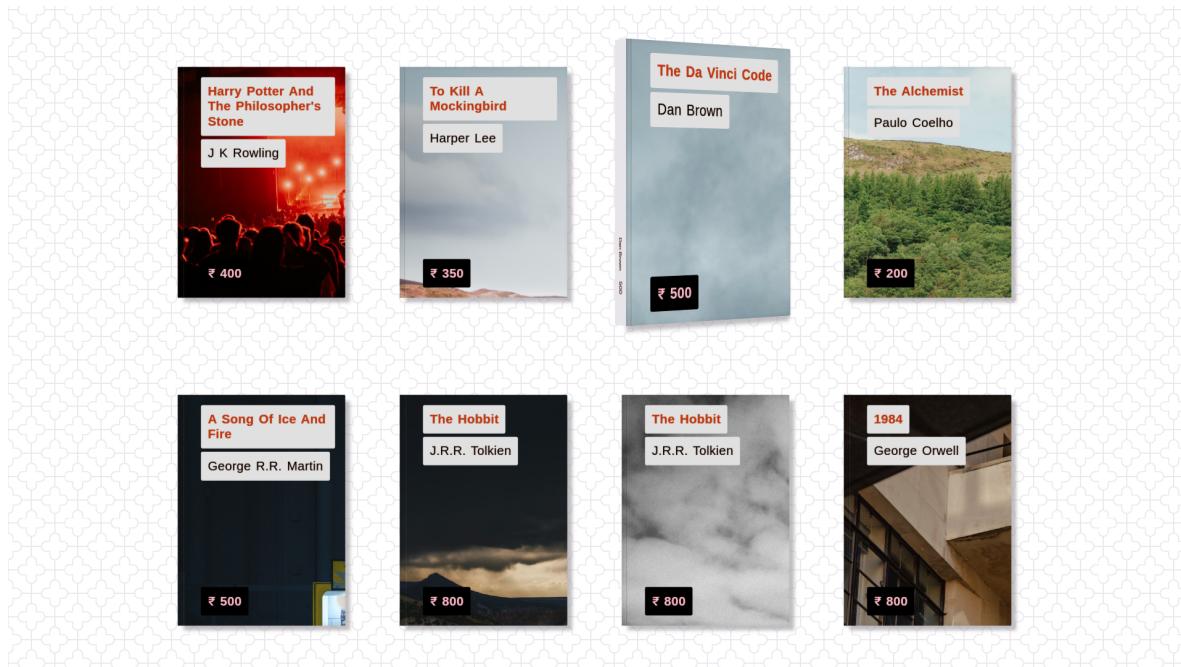


Figure 8: The Books retrieved from /getbooks.php and shown on the Frontend

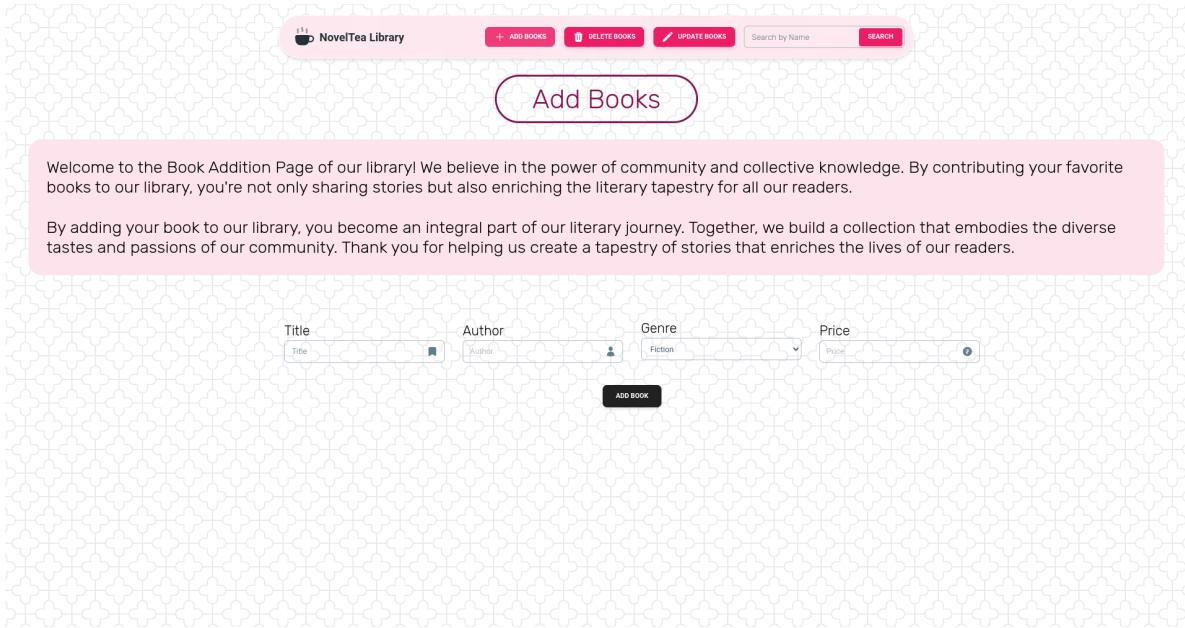


Figure 9: The Add Page

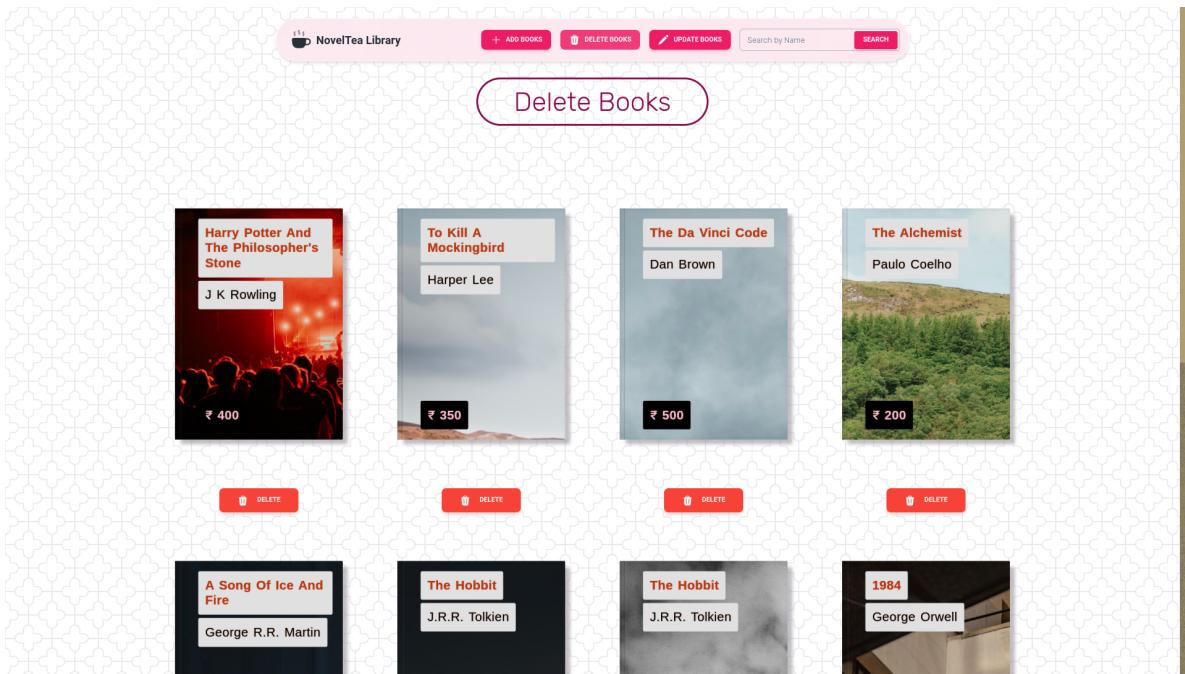


Figure 10: The Delete Page

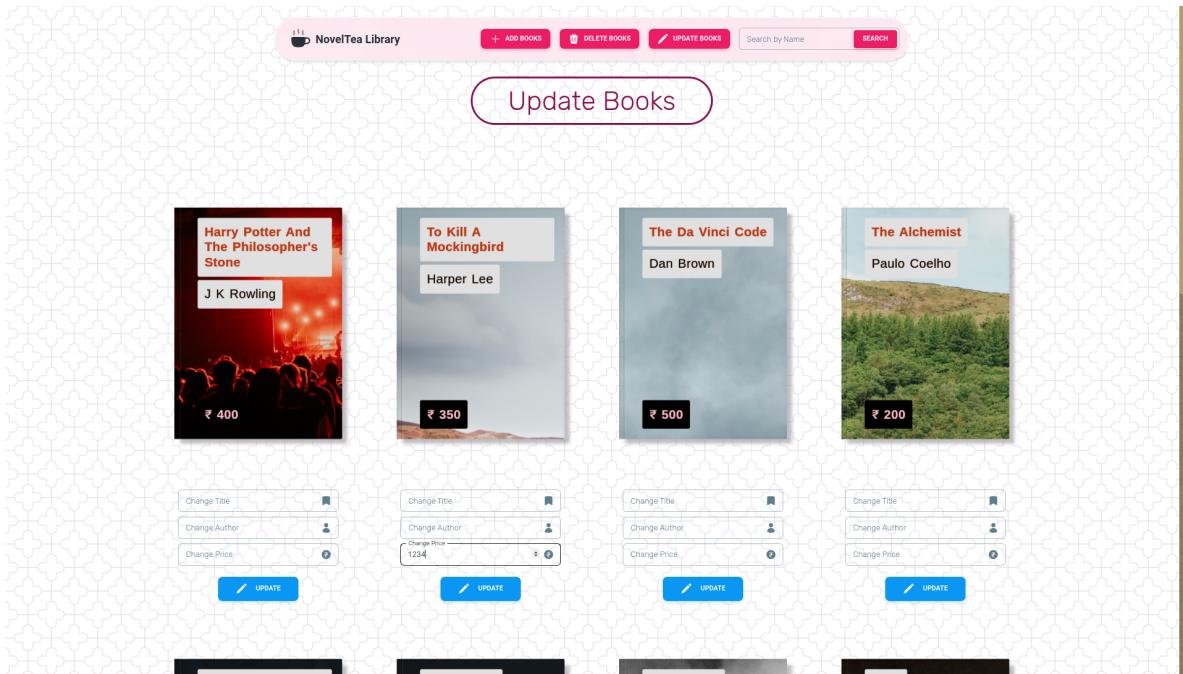


Figure 11: The Update Page, where any field can be updated

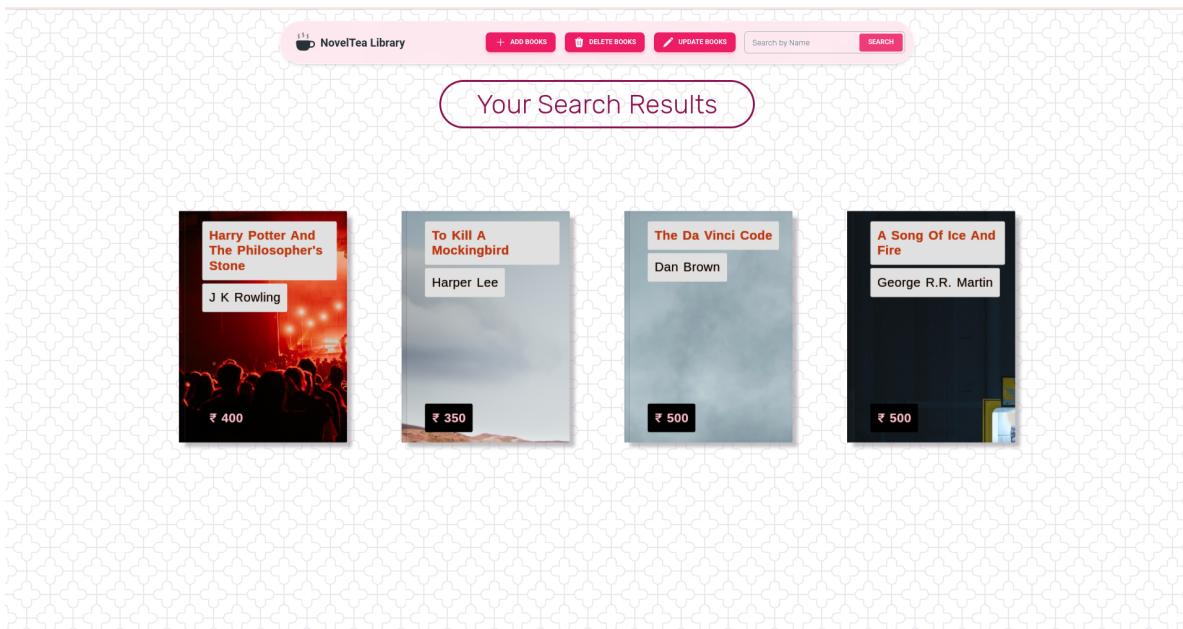
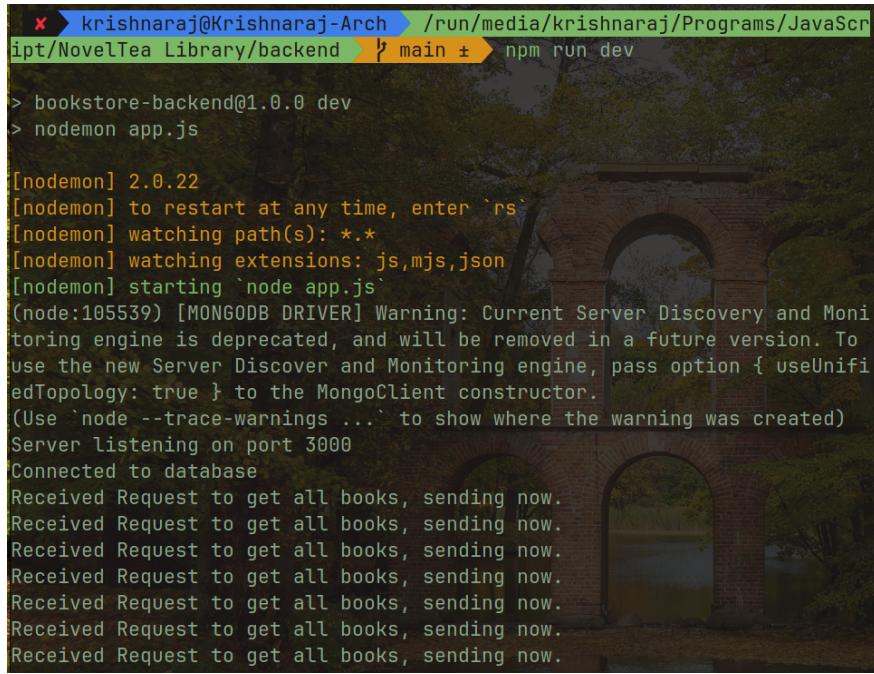


Figure 12: The Search Page showing results of search term "H"

7.2 Node and Express Backend

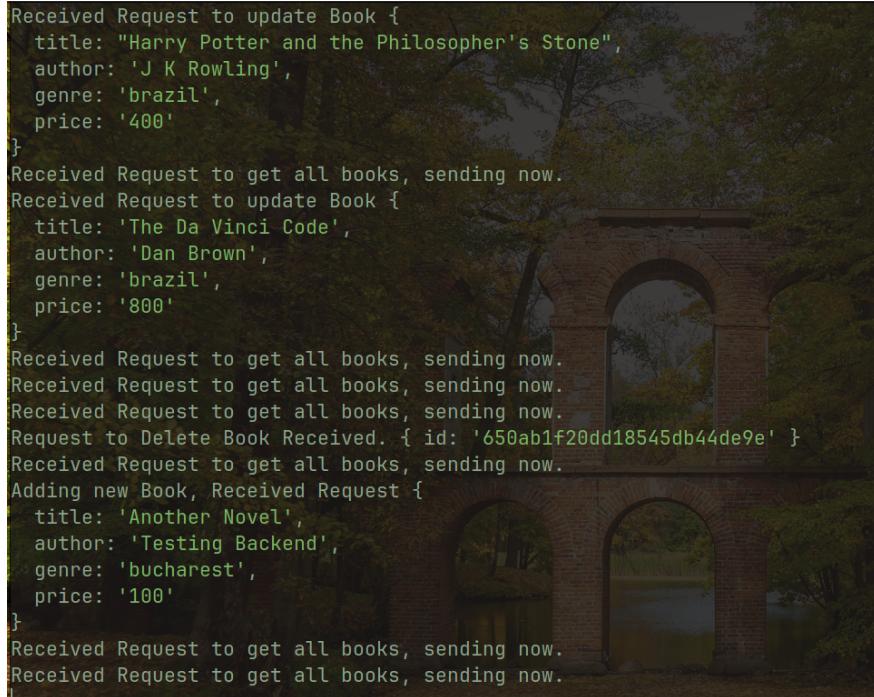


```
krishnaraj@Krishnaraj-Arch:~/run/media/krishnaraj/Programs/JavaScn/ipt/NovelTea Library/backend$ main ± npm run dev

> bookstore-backend@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
(node:105539) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(Use `node --trace-warnings ...` to show where the warning was created)
Server listening on port 3000
Connected to database
Received Request to get all books, sending now.
```

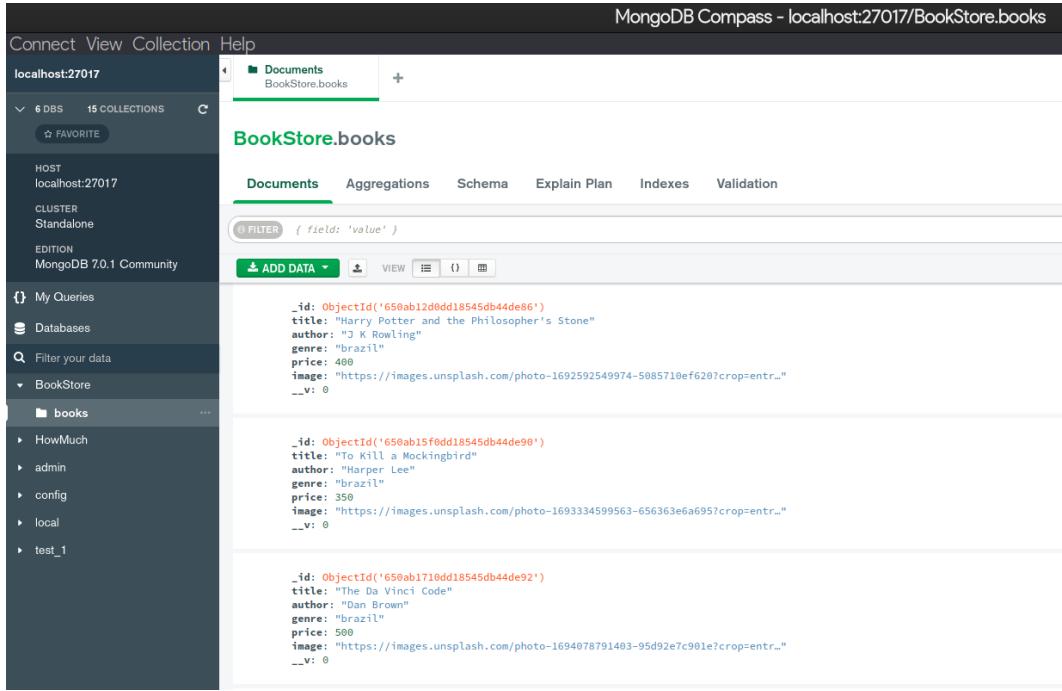
Figure 13: The Node and Express Backend server running. It is listening on port 3000.



```
Received Request to update Book {
  title: "Harry Potter and the Philosopher's Stone",
  author: 'J K Rowling',
  genre: 'brazil',
  price: '400'
}
Received Request to get all books, sending now.
Received Request to update Book {
  title: 'The Da Vinci Code',
  author: 'Dan Brown',
  genre: 'brazil',
  price: '800'
}
Received Request to get all books, sending now.
Received Request to get all books, sending now.
Received Request to get all books, sending now.
Request to Delete Book Received. { id: '650ab1f20dd18545db44de9e' }
Received Request to get all books, sending now.
Adding new Book, Received Request {
  title: 'Another Novel',
  author: 'Testing Backend',
  genre: 'bucharest',
  price: '100'
}
Received Request to get all books, sending now.
Received Request to get all books, sending now.
```

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

7.3 MongoDB Database



The screenshot shows the MongoDB Compass interface connected to localhost:27017. The left sidebar lists databases and collections, with 'BookStore' selected. The main area displays the 'Documents' tab for the 'BookStore.books' collection. Three documents are listed, each representing a book with fields like _id, title, author, genre, price, and image.

```

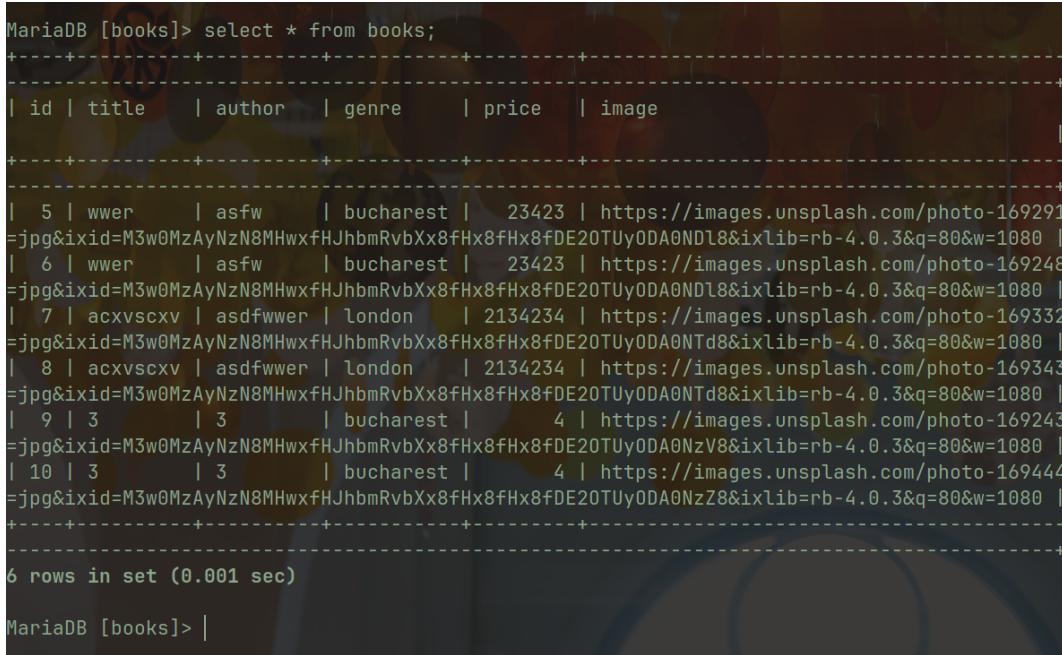
_id: ObjectId("650ab12d0dd18545db44de86")
title: "Harry Potter and the Philosopher's Stone"
author: "J K Rowling"
genre: "brazil"
price: 400
image: "https://images.unsplash.com/photo-1692592549974-5085710ef620?crop=entr..."
__v: 0

_id: ObjectId("650ab15f0dd18545db44de90")
title: "To Kill a Mockingbird"
author: "Harper Lee"
genre: "brazil"
price: 350
image: "https://images.unsplash.com/photo-1693334599563-656363e6a695?crop=entr..."
__v: 0

_id: ObjectId("650ab1710dd18545db44de92")
title: "The Da Vinci Code"
author: "Dan Brown"
genre: "brazil"
price: 500
image: "https://images.unsplash.com/photo-1694078791403-95d92e7c901e?crop=entr..."
__v: 0

```

Figure 15: The MongoDB Compass showing the database.



```

MariaDB [books]> select * from books;
+----+-----+-----+-----+-----+-----+
| id | title | author | genre | price | image
+----+-----+-----+-----+-----+-----+
| 5 | wwer | asfw | bucharest | 23423 | https://images.unsplash.com/photo-169291
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NDl8&ixlib=rb-4.0.3&q=80&w=1080 |
| 6 | wwer | asfw | bucharest | 23423 | https://images.unsplash.com/photo-169248
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NDl8&ixlib=rb-4.0.3&q=80&w=1080 |
| 7 | acxvscvx | asdfwwer | london | 2134234 | https://images.unsplash.com/photo-169332
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NTd8&ixlib=rb-4.0.3&q=80&w=1080 |
| 8 | acxvscvx | asdfwwer | london | 2134234 | https://images.unsplash.com/photo-169343
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NTd8&ixlib=rb-4.0.3&q=80&w=1080 |
| 9 | 3 | 3 | bucharest | 4 | https://images.unsplash.com/photo-169243
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NzV8&ixlib=rb-4.0.3&q=80&w=1080 |
| 10 | 3 | 3 | bucharest | 4 | https://images.unsplash.com/photo-169444
=jpg&ixid=M3w0MzAyNzN8MHwxJHJhbRvbXx8fHx8fHx8fDE20TUyODA0NzZ8&ixlib=rb-4.0.3&q=80&w=1080 |
+----+-----+-----+-----+-----+-----+
6 rows in set (0.001 sec)

MariaDB [books]> |

```

Figure 16: The MongoDB Database

8 Code

```
1 {
2   "name": "bookstore-backend",
3   "version": "1.0.0",
4   "description": "Backend for the Bookstore project",
5   "main": "app.js",
6   "scripts": {
7     "start": "node app.js",
8     "dev": "nodemon app.js"
9   },
10  "dependencies": {
11    "axios": "^1.5.0",
12    "cors": "^2.8.5",
13    "express": "^4.17.1",
14    "mongoose": "^5.13.2"
15  },
16  "devDependencies": {
17    "nodemon": "^2.0.12"
18  }
19 }
```

Listing 1: package.json

```
1 const express = require("express");
2 const bodyParser = require("body-parser");
3 const mongoose = require("mongoose");
4 const cors = require("cors");
5 const setApiRoutes = require("./routes/api");
6
7 const app = express();
8
9 // Set up middleware
10 app.use(bodyParser.json());
11 app.use(cors());
12
13 // Set up API routes
14 app.use(setApiRoutes);
15
16 // Connect to database
17 mongoose
18   .connect("mongodb://localhost/BookStore", { useNewUrlParser: true })
19   .then(() => {
20     console.log("Connected to database");
21   })
22   .catch((error) => {
23     console.error("Error connecting to database:", error);
24   });
25
26 // Start server
27 const port = process.env.PORT || 3000;
28 app.listen(port, () => {
29   console.log(`Server listening on port ${port}`);
30 });


```

Listing 2: app.js managing the app.

```
1 const express = require("express");
2 const router = express.Router();
3 const BooksController = require("../controllers/books");
4
```

```
5 const booksController = new BooksController();
6
7 router.get("/books", booksController.getAllBooks);
8 router.get("/", (req, res) => {
9   res.send("Hello World!");
10 });
11 router.get("/books/:id", booksController.getBookById);
12 router.post("/books", booksController.createBook);
13 router.put("/books/:id", booksController.updateBook);
14 router.delete("/books/:id", booksController.deleteBook);
15
16 module.exports = router;
```

Listing 3: api.js defining the routes.

```
1 const mongoose = require("mongoose");
2
3 const bookSchema = new mongoose.Schema({
4   title: {
5     type: String,
6     required: true,
7   },
8   author: {
9     type: String,
10    required: true,
11   },
12   genre: {
13     type: String,
14     required: true,
15   },
16   price: {
17     type: Number,
18     required: true,
19   },
20   image: {
21     type: String,
22   },
23 });
24
25 const Book = mongoose.model("Book", bookSchema);
26
27 module.exports = Book;
```

Listing 4: Book.js interacting with MongoDB

```
1 const Book = require("../models/Book");
2 const axios = require("axios");
3
4 async function getRandomImageUrl() {
5   try {
6     const response = await axios.get("https://api.unsplash.com/photos/random", {
7       headers: {
8         Authorization: "Client-ID UBAXKPq5Wg0xU9z7XjLZrt8B0hayG2dv8gh_vyGQg-0",
9       },
10     });
11
12     return response.data.urls.regular;
13   } catch (error) {
14     console.error(error);
```

```
15     return null;
16 }
17 }
18 class BooksController {
19   async getAllBooks(req, res) {
20     console.log("Received Request to get all books, sending now. ");
21     try {
22       const books = await Book.find();
23       res.status(200).json(books);
24     } catch (error) {
25       res.status(500).json({ message: error.message });
26     }
27   }
28
29   async getBookById(req, res) {
30     try {
31       const book = await Book.findById(req.params.id);
32       if (!book) {
33         return res.status(404).json({ message: "Book not found" });
34       }
35       res.status(200).json(book);
36     } catch (error) {
37       res.status(500).json({ message: error.message });
38     }
39   }
40
41   async createBook(req, res) {
42     console.log("Adding new Book, Received Request", req.query);
43     const image = await getRandomImageUrl();
44     const book = new Book({
45       title: req.query.title,
46       author: req.query.author,
47       genre: req.query.genre,
48       price: req.query.price,
49       image: image,
50     });
51
52     try {
53       const newBook = await book.save();
54       res.status(201).json(newBook);
55     } catch (error) {
56       res.status(400).json({ message: error.message });
57     }
58   }
59
60   async updateBook(req, res) {
61     console.log("Received Request to update Book", req.query);
62     try {
63       const book = await Book.findById(req.params.id);
64       if (!book) {
65         return res.status(404).json({ message: "Book not found" });
66       }
67
68       book.title = req.query.title || book.title;
69       book.author = req.query.author || book.author;
70       book.genre = req.query.genre || book.genre;
71       book.price = req.query.price || book.price;
72
73       const updatedBook = await book.save();
74     }
75   }
76 }
```

```
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.