

Library Management System

It is a simple web application consisting of two parts: an admin management section and a user interface for viewing the books in the library. The application is built using React.js and integrated with Firebase for data management and authentication.

Features

- **Adding New books (Admin)**
- **Deleting books (Admin)**
- **Updating the books (Admin)**
- **Membership Management (Admin)**
- **No Login Required to View the books (Users)**
- **Universal Search is implemented**
- **Pagination is implemented for arranging books**
- **Simple and Easy to use UI**
- **Authentication Context is Managed**

Usage

- To use the web application, follow these steps:
 1. Ensure that you have a compatible web browser installed on your device.
 2. If you are an admin user, log in to access the admin management section. Otherwise, proceed as a regular user.
 3. Explore the available books, use the search functionality, and navigate through different pages using pagination.
 4. If you are an admin user, utilize the provided functionalities to add, delete, update books, and manage user memberships.

Technology Used

- **React JS** : A JavaScript library for building user interfaces, used as the foundation of the web application.
- **Firebase Authentication** : A service provided by Firebase for user authentication, ensuring secure access to the application.
- **Firebase Firestore**: A cloud-based NoSQL database provided by Firebase, used for data management and storage of books and user information.
- **MUI (Material UI Framework)**: A popular React UI framework that provides pre-designed components and styling for a visually appealing and responsive user interface.
- **React Routers** : A library for handling routing in React applications, used for navigation between different pages and components within the application.

Setting up Project Locally

To set up the project locally, follow these steps:

1. Make sure you have Node.js and npm (Node Package Manager) installed on your computer. You can download them from the official Node.js website.
2. Clone the project repository from the version control system (e.g., GitHub) using Git or download the source code as a ZIP file from the repo [Library Management](#).
3. Open a terminal or command prompt and navigate to the project directory.
4. Run the following command to install the project dependencies:
5. `npm install`
6. Create a Firebase project and set up Firebase Authentication and Firestore services. Obtain the necessary configuration details, including the API keys and credentials.
7. Add those configuration API keys and credentials to the project in the page [Firebase.js](#)
8. Start the development server by running the following command:
9. `npm start`
10. The project should now be running locally. Open your web browser and visit `http://localhost:3000` to view the application.

You can now explore and interact with the project locally on your computer. Make any necessary changes or enhancements as per your requirements.

Hosting your application

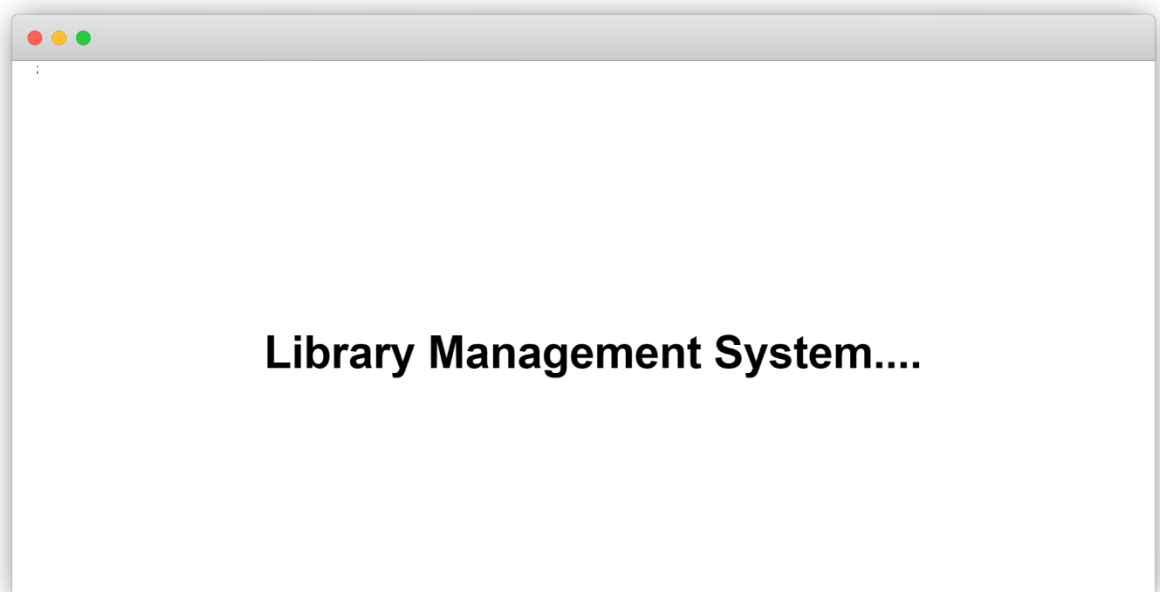
To host your project on Vercel from a GitHub repository, follow these steps:

1. Sign up for an account on Vercel (<https://vercel.com>) if you haven't already.
2. Make sure your project is pushed to a GitHub repository.
3. Login to Vercel and click on the "New Project" button.
4. Select your GitHub account and repository from the list.
5. Configure the project settings like the name, framework, and build settings. Make sure to set the root directory and the build command appropriate for your project.
6. Review the deployment settings and make any necessary changes.
7. Click on the "Deploy" button to start the deployment process.
8. Vercel will build and deploy your project automatically. Once the deployment is complete, you will receive a unique URL for your hosted project.
9. Visit the provided URL to see your project live on Vercel.

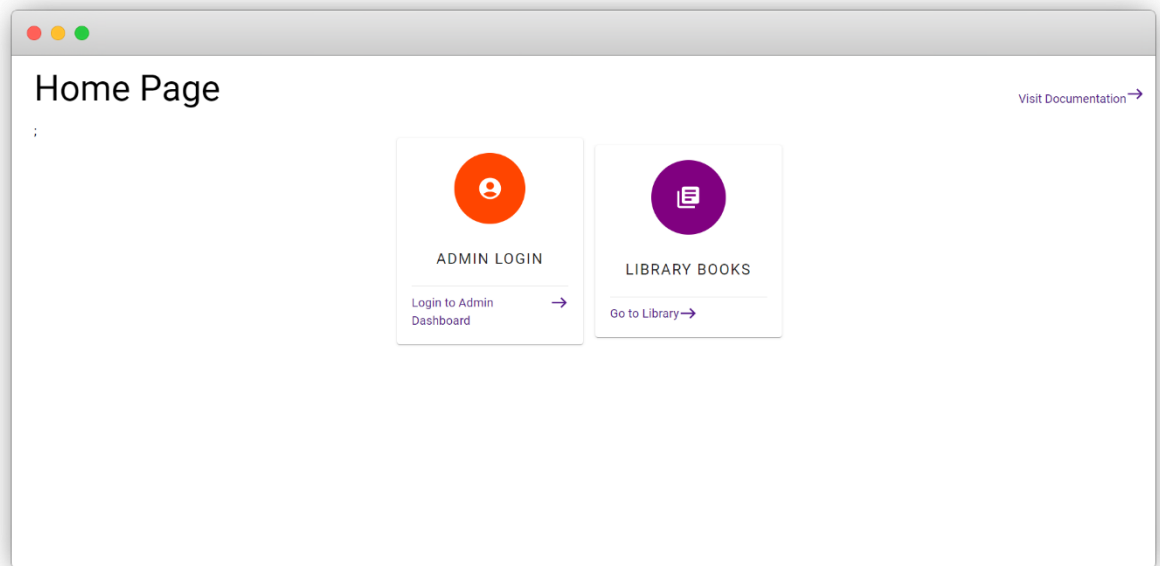
Vercel will automatically rebuild and deploy your project whenever you push new changes to the GitHub repository. You can also configure custom domain settings and other advanced options in the Vercel project settings.

Snapshots

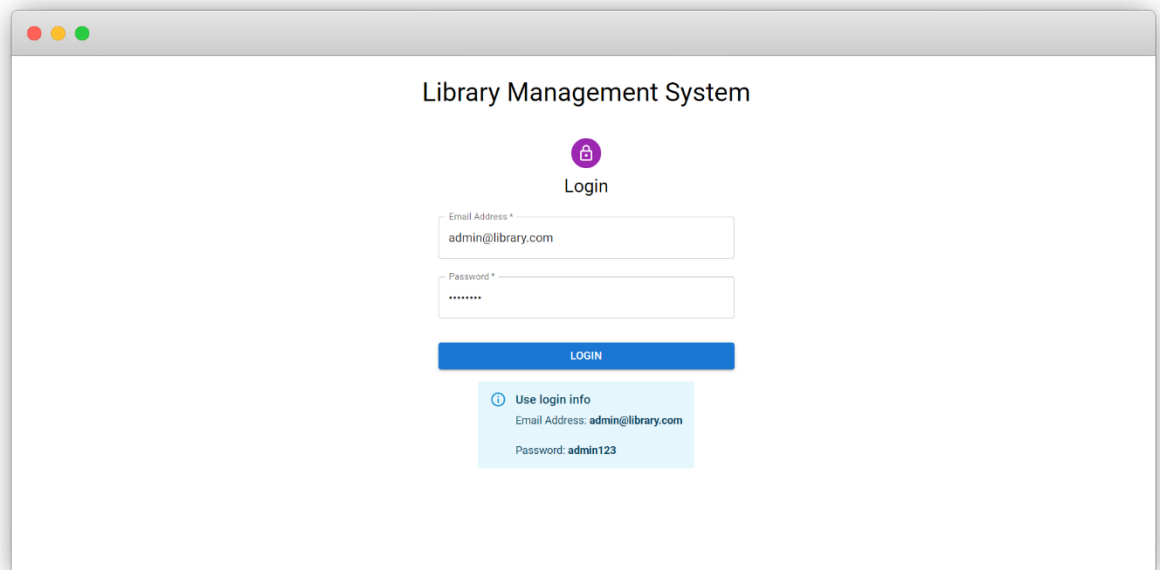
- **Splash Screen**



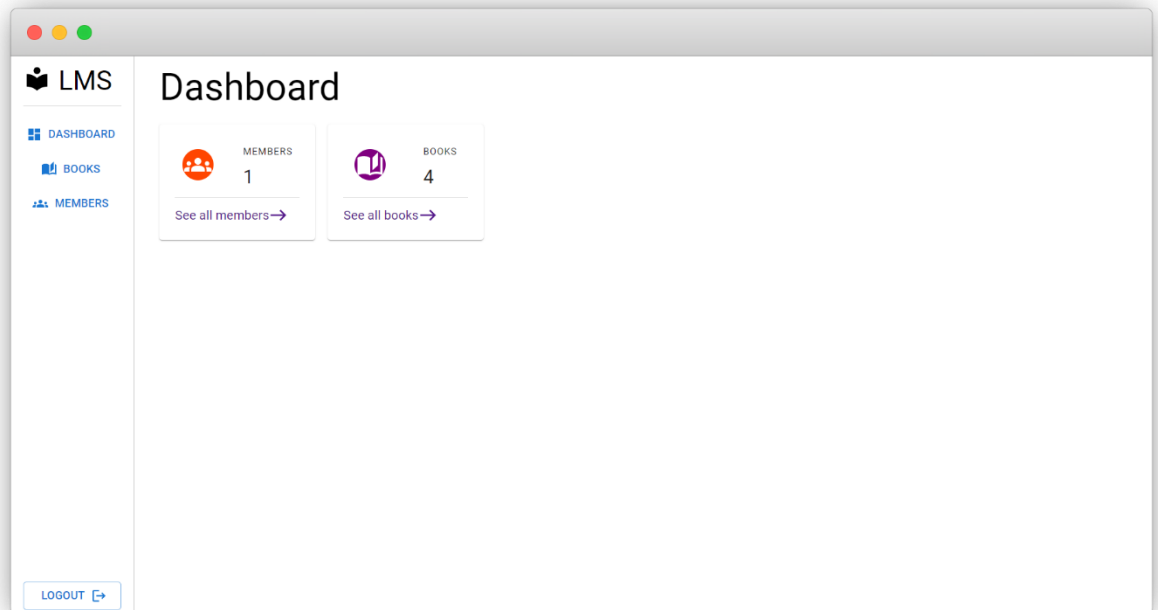
- **Home Screen**



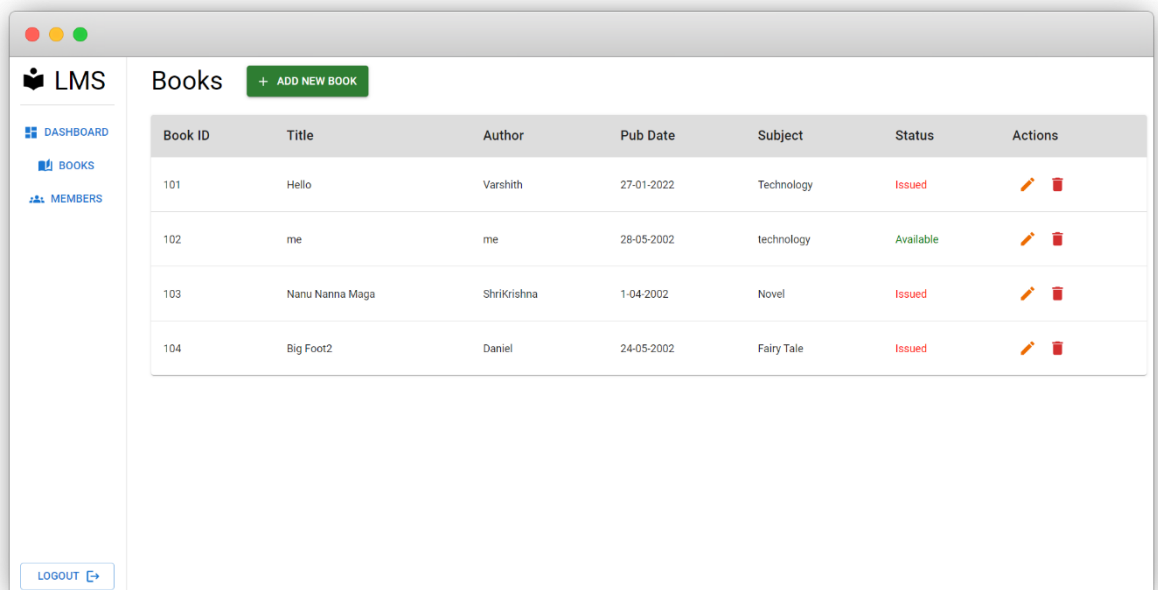
- **Admin Login**



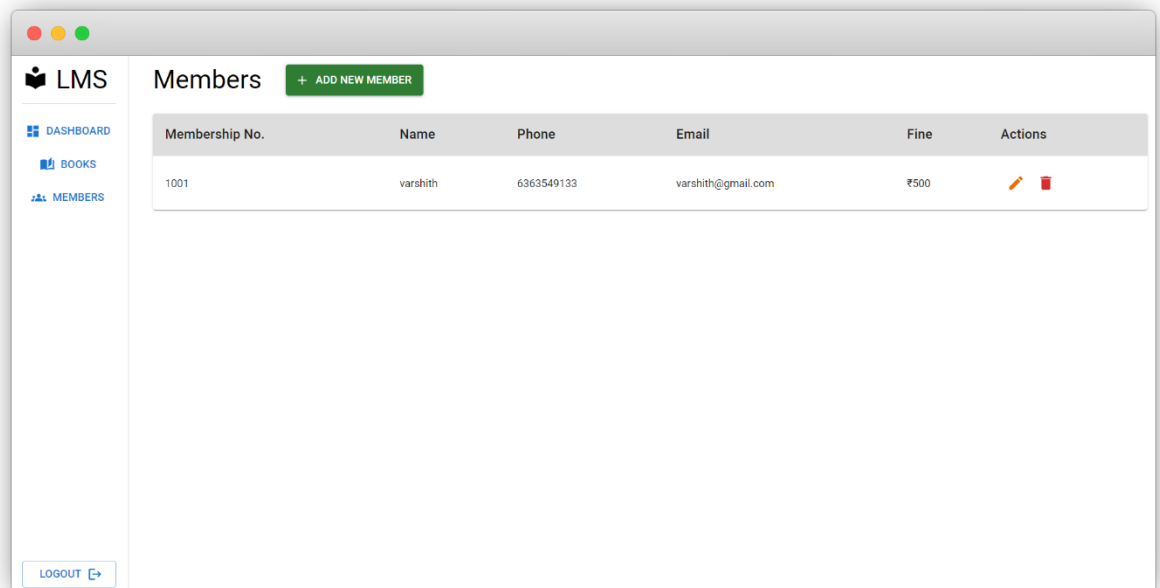
- Admin Dashboard





- Admin Books list



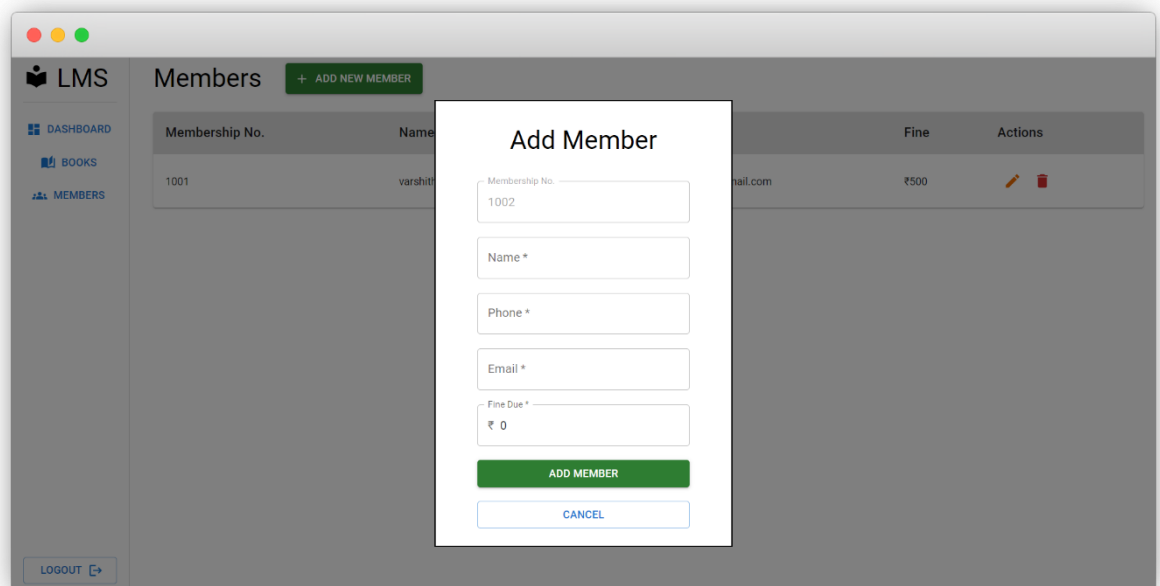
- Admin Members List



The screenshot shows a web application interface for managing members. On the left is a sidebar with the 'LMS' logo and navigation links for 'DASHBOARD', 'BOOKS', and 'MEMBERS'. The main content area is titled 'Members' and includes a green '+ ADD NEW MEMBER' button. Below this is a table with columns for Membership No., Name, Phone, Email, Fine, and Actions. A single member is listed with Membership No. 1001, Name varshith, Phone 6363549133, Email varshith@gmail.com, and a Fine of ₹500. The Actions column contains edit and delete icons. A 'LOGOUT' button is located at the bottom left of the sidebar.

Membership No.	Name	Phone	Email	Fine	Actions
1001	varshith	6363549133	varshith@gmail.com	₹500	 

- Admin Add Members Form



The screenshot shows the same 'Members' page as before, but with a modal form titled 'Add Member' open in the center. The modal contains input fields for 'Membership No.' (with the value 1002), 'Name *', 'Phone *', 'Email *', and 'Fine Due *' (with the value ₹ 0). At the bottom of the modal are two buttons: a green 'ADD MEMBER' button and a blue 'CANCEL' button. The background of the page is dimmed.

Membership No.

1002

Name *

Phone *

Email *

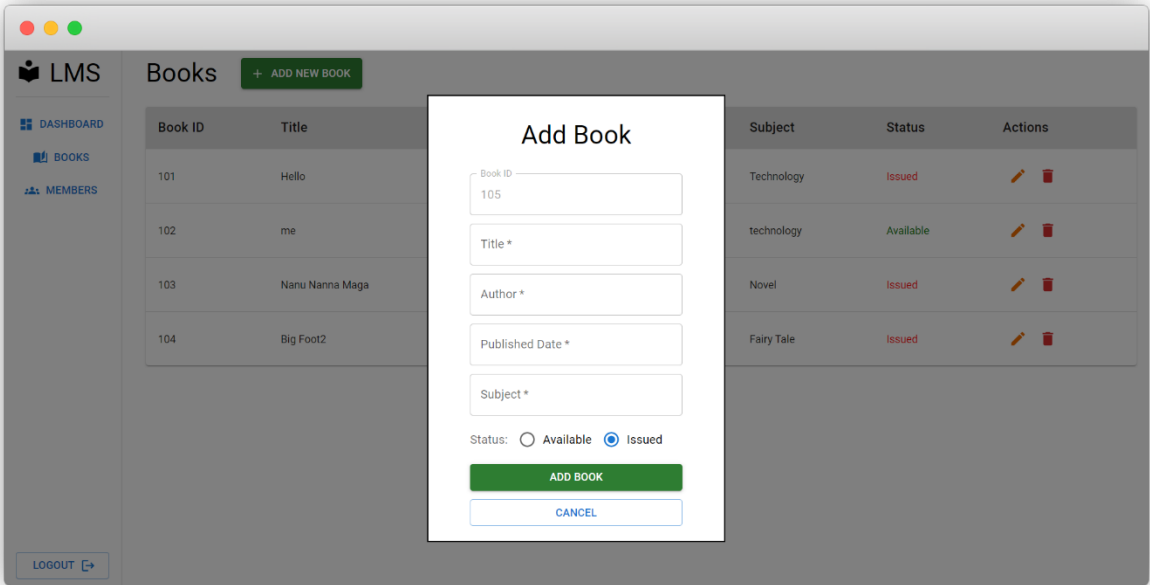
Fine Due *

₹ 0

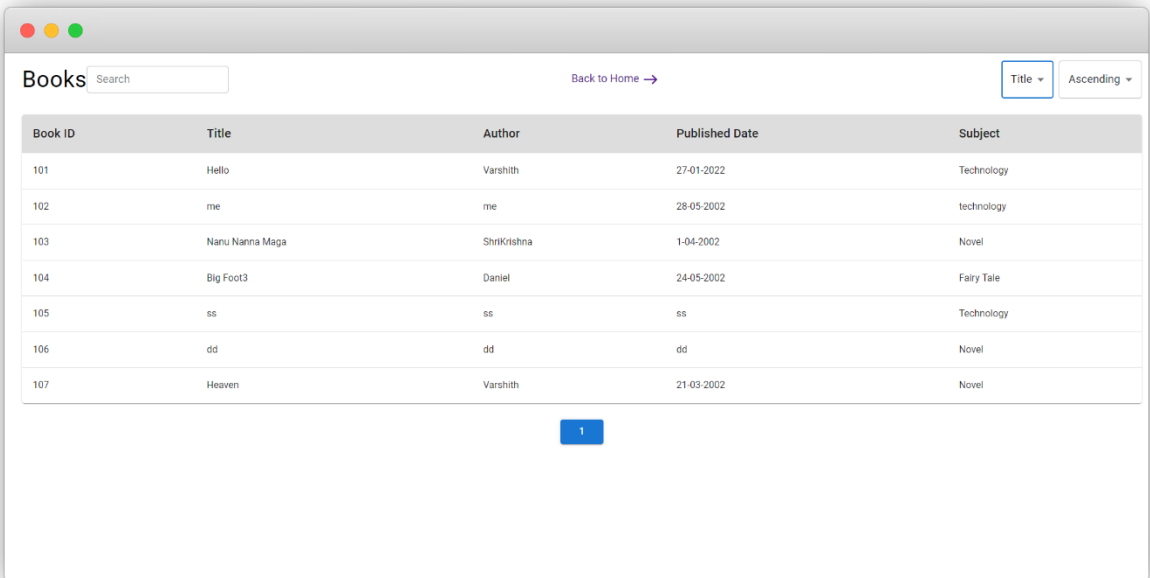
ADD MEMBER

CANCEL

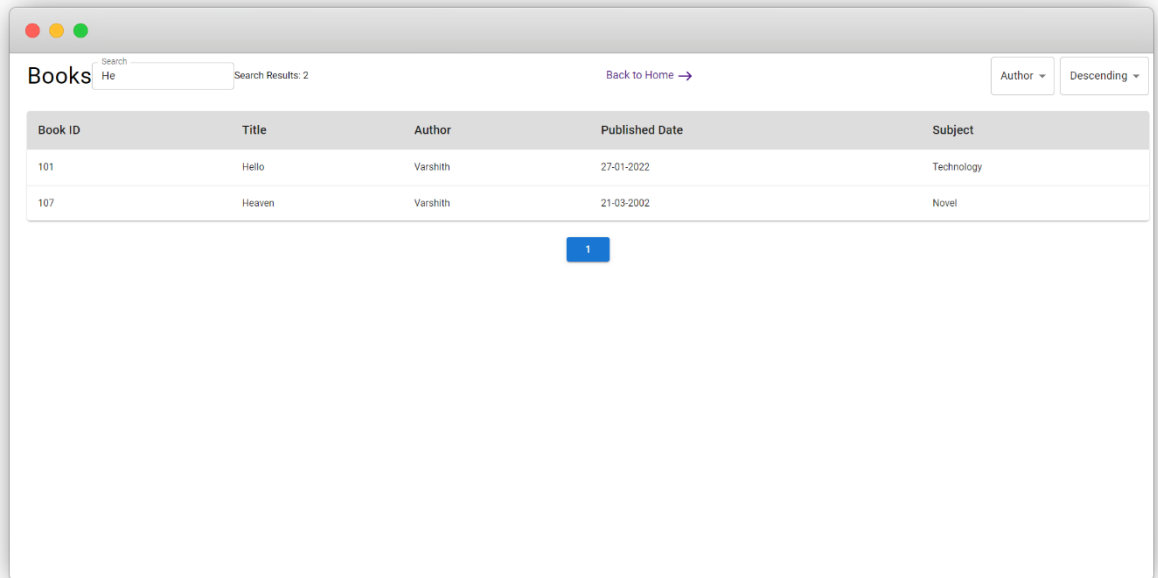
- **Admin Add or Edit Book**



- **User Book List Page**



- **User Book Search**



Code review

Firestore Connection

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  apiKey: REACT_APP_FIREBASE_API_KEY,
  authDomain: REACT_APP_AUTH_DOMAIN,
  projectId: REACT_APP_PROJECT_ID,
  storageBucket: REACT_APP_STORAGE_BUCKET,
  messagingSenderId: REACT_APP_MESSAGING_SENDER_ID,
  appId: REACT_APP_FIREBASE_APP_ID
};

const app = initializeApp(firebaseConfig);
export const db = getFirestore(app);
export const auth = getAuth();
```

Retrieving data from Firestore

```
async function firestoreData() {
  let tempBooks = [];
  try {
    const q2 = query(collection(db, "books"), orderBy("bookId"));
    const querySnapshot2 = await getDocs(q2);
    querySnapshot2.forEach((doc) => {
      tempBooks.push(doc.data());
    });
  }
}
```



```

    } catch (err) {
      console.log(err);
    }
  }
}

```

Adding a new Book to Firestore database

```

const [bookId, setBookId] = useState(0);
const [title, setTitle] = useState("");
const [author, setAuthor] = useState("");
const [pubDate, setPubDate] = useState("");
const [subject, setSubject] = useState("");
const [issued, setIssued] = useState("issued");
const AddNewBook = async (e) => {
  e.preventDefault();
  try {
    const bookRef = collection(db, "books");
    await addDoc(bookRef, {
      bookId,
      title,
      author,
      pubDate,
      subject,
      issued
    });
  }
}

```

Deleting Book from Firestore

```

const deleteBook = async (bookId) => {
  try {
    const querySnapshot = await getDocs(
      query(collection(db, "books"), where("bookId", "==", bookId))
    );

    if (!querySnapshot.empty) {
      querySnapshot.forEach((doc) => {
        deleteDoc(doc.ref);
      });
      console.log("Book deleted successfully!");
      window.location.reload();
    }
    else {
      console.log("Book not found!");
    }
  } catch (error) {
    console.error("Error deleting book:", error);
  }
};

const deleteHandler = (index) => {
  let confirmBool = window.confirm(`Are you sure you want to delete "${booksData[index].title}"?`);
  if (confirmBool) {
    console.log(index, booksData[index].title);
  }
}

```

```

        const bookId = booksData[index].bookId;
        console.log(bookId);
        deleteBook(bookId);
    }
};

```

Updating Book Data

```

async function editFormSubmit(e) {
    e.preventDefault();
    const form = e.target;
    const newTitle = form.title.value;
    const newAuthor = form.author.value;
    const newPubDate = form.PublishedDate.value;
    const newSubject = form.Subject.value;
    const newIssued = form.issued.value;
    try {
        const booksRef = collection(db, "books");
        const querySnapshot = await getDocs(booksRef);
        querySnapshot.forEach((doc) => {
            const bookData = doc.data();
            if (bookData.bookId === booksData[editIndex].bookId) {
                const bookRef = doc.ref;
                updateDoc(bookRef, {
                    title: newTitle,
                    author: newAuthor,
                    pubDate: newPubDate,
                    subject: newSubject,
                    issued: newIssued,
                });
                console.log("Book updated successfully!");
                window.location.reload(false);
            }
        });
    } catch (error) {
        console.error("Error updating book:", error);
    }
}

const editBtnHandler = (index) => {
    setFormType("edit");
    setEditIndex(index);
    setOpenForm(true);
    setTimeout(() => {
        const book = booksData[index];
        document.getElementById("bookId").value = book.bookId;
        document.getElementById("title").value = book.title;
        document.getElementById("author").value = book.author;
        document.getElementById("PublishedDate").value = book.pubDate;
        document.getElementById("Subject").value = book.subject;

        document.querySelector(`input[name="issued"][value="${book.issued}"]`).checked =
        true;

    }, 3000);
};

```

Future Enhancements

- More responsive Tables
- Connecting the Members to Books Database
- Member Login and Registration
- Image Cards for books