**Full Stack Development with MERN**

**Frontend Development Report**

|  |  |
| --- | --- |
| Date | 10-07-2024 |
| Team ID | SWTID1719939027 |
| Project Name | Book Nest |
| Maximum Marks | 10 |

**Project Title: Book Nest**

Date: 10-07-2024

Prepared by: T.V.Nageswara Rao

**Objective**

We have started this project with a goal to provide user and seller friendly web application to sell and buy books online at lowest possible price as publishers can directly register and act as sellers to sell them at a lower price than the available market price

The key aspect include dedicated login pages for buyers ,sellers and admin where data is rendered according to their role but all of them can access universal landing page from there they are divided based on the login details buyers page contains a Wishlist,Books,Orders routes and Seller page contains Addbooks page,Manage inventory page finally Admin login contains data of all available buyers and available sellers and the total list of books sold through the website

**Technologies Used**

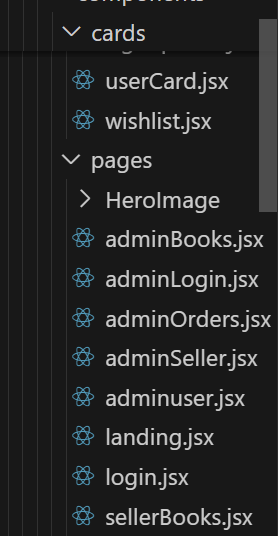
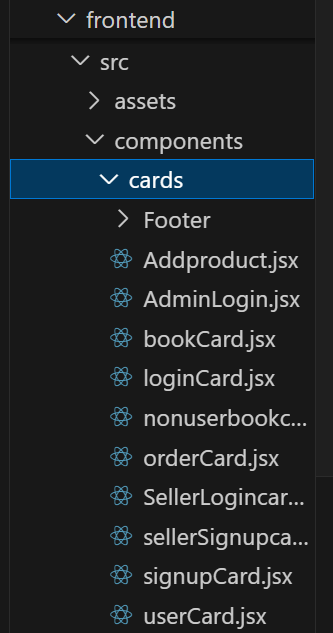
* **Frontend Framework:** React.js
* **State Management: None**
* **UI Framework/Libraries:**
  + Lucide-react
  + Tanstack/react-query
  + Tailwind
  + React
  + React-dom
  + React-hook-form
  + React-router-dom
  + @radix-ui/react-label
  + @radix-ui/react-slot

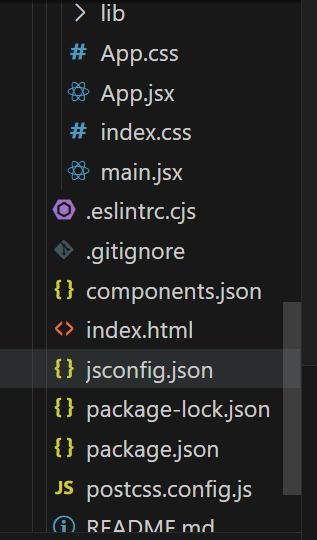
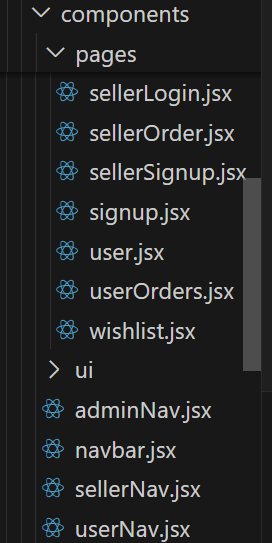
**API Libraries:**

* Axios,
* jwt-decode

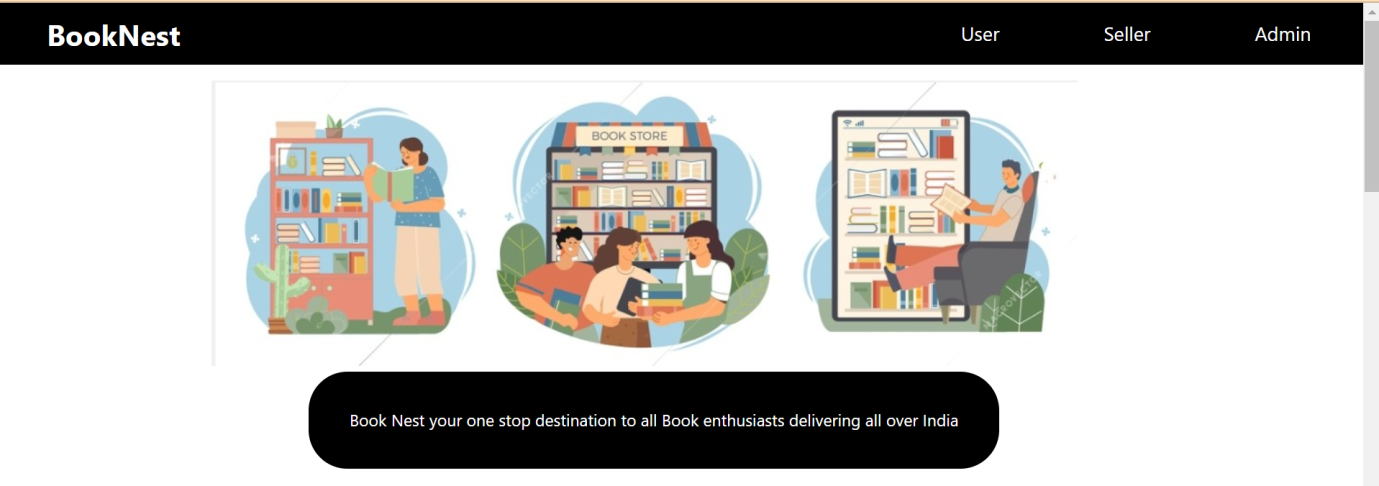
**Project Structure**

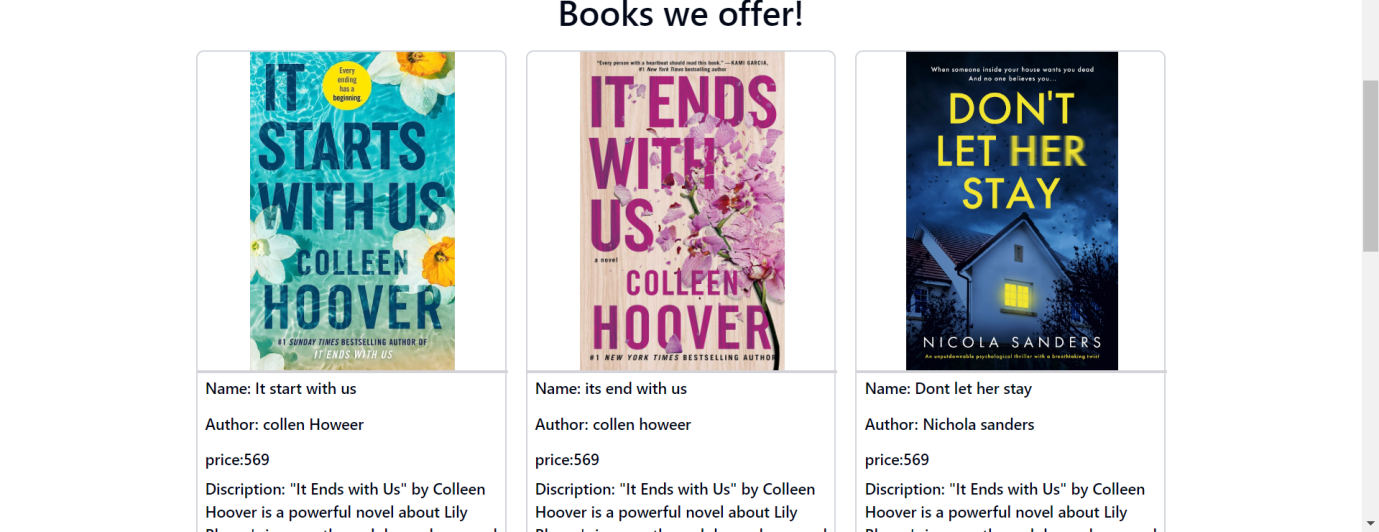
**File structuring in Frontend**

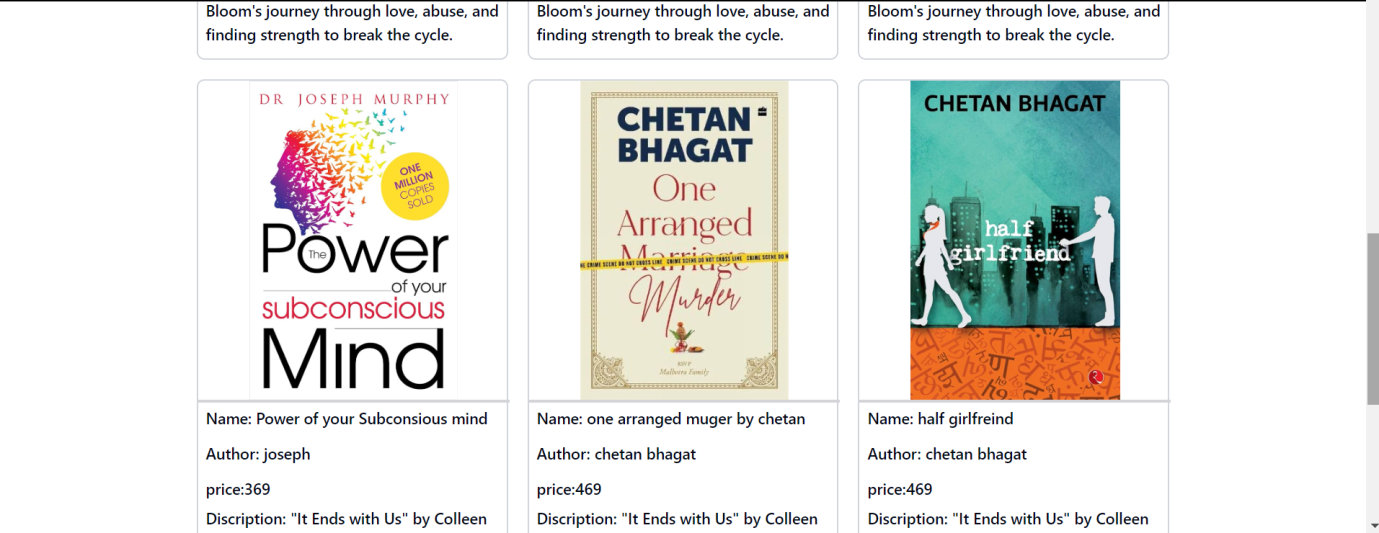
****

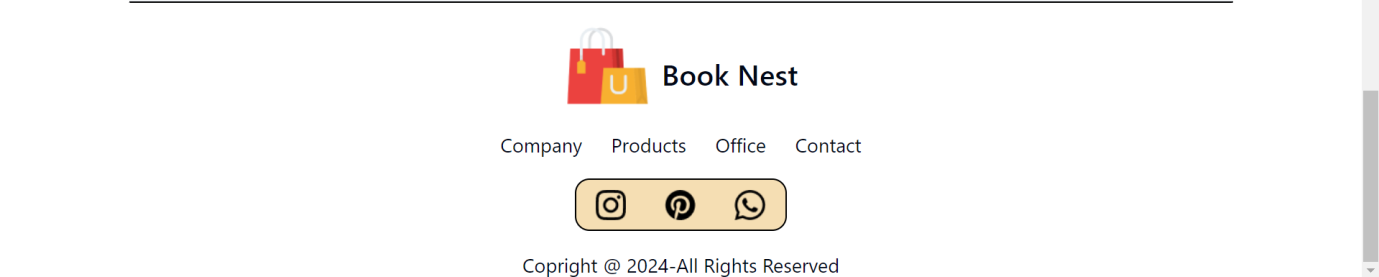
****

**Landing page**

****

****

****

****

**Key Components**

1. **App.js**

In this file all the routes are placed by importing the individual components and render the required component which is done by react-router-dom. Global state management operations are also performed in this page using Queryclient provider from Tanstack .

1. **/components**

All the components used are created in this folder for better file structuring for easy usage of required pieces in our project we have Navbar,Admin Navbar,Seller Navbar,User Navbar along with Footer,Addproductcard,Admin login,book card,Login card,nonuserCard,Order card,seller login card,sellersignupCard,Signupcard,userCard,Wishlist all these components are mounted at required positions in pages

1. **/pages**

We have several pages in our website created for various requirements and working they are AdminBooks,Admin login,Admin orders,AdminSeller,adminuser,landing,login,SellerBooks, SellerLogin, SellerOrder, SellerSignup,signup,user,userOrders,wishlist

**Routing**

Routing is managed using React Router. Here are the main routes:

Landing page routes

* **/user**- to log into user page
* **/seller** – to log into seller page and add or remove books in inventory
* **/admin**- to log into sellerprofile to manage books ,users and sellers

User page routes:

* **/wishlist**- to display books which are added to wishlist
* **/books**- to display all available books from sellers
* **/userOrders**-to check the details of books ordred

Seller page routes

* **/seller-** to add new books into inventory
* **/sellerroutes-** to view the data of books he sold

Admin page routes

* **/admin** –This takes us to the landing page of admin where he can see all books from all sellers available
* **/admin/users**- Admin can view all the users in the Book Nest
* **/admin/sellers-**Admin can view all the sellers in the Book Nest
* **/adminorders**-Admin can view log of all the books sold from any seller

**State Management (If Applicable)**

Not applicable­­­­­­­­­­­­

**Integration with Backend**

The frontend communicates with the backend APIs hosted on [backend URL]. Key endpoints include:

* **GET /api/data** - Retrieves data for display.
* **POST /api/user/login** - Handles user authentication.

The frontend of our book selling project communicates with the backend APIs hosted on [backend URL]. We use Axios, a promise-based HTTP client for the browser and Node.js, to handle these communications. Below are the key endpoints and how we integrate with them using Axios:

Key Endpoints

GET /api/data - Retrieves data for display(Retrieving data).

Ex-axios.get("http://localhost:3000/admin/seller")

POST /api/user/login - Handles user authentication(Login forms).

Ex-axios.post(http://localhost:3000/seller/books)

Using Axios for API Integration To integrate with the backend APIs, we use Axios to make HTTP requests. Here are examples of how we use Axios for the key endpoints

**User Interface (UI) Design**

* The UI design follows a [describe design principles].
* Implemented using [UI framework/library].

The design principles we followed are we want to show some part of the website(Landing page) for everyone who came into the website then asked to log into their specific roles of users,sellers,Admin .Also followed same clean and simple UI .When user logins he can view all the available books in each individual cards when seller logs in he can view all his books available and an option to add new books and a detailed log of all the books he sold and admin can view all the users and sellers available along with their booka and their orders

We have used React.js for the frontend part and Tailwind css for the styling

* @radix-ui/react-label: UI component library for accessible labels.
* @radix-ui/react-slot: UI component library for slot-based component composition.
* lucide-react: UI library for icons.
* react: UI library for building user interfaces.
* react-cookie: Library for handling cookies in React.
* react-dom: Library for working with the DOM in React applications.
* react-hook-form: Library for managing forms in React.
* react-router-dom: Library for routing in React applications.
* tailwind-merge: Utility library for merging Tailwind CSS classes.

**Third-Party Integrations (If any)**

1)Bcrypt:

Bcrypt is a cryptographic hash function used to store and hash passwords. It uses complex algorithms to make it harder for hackers to crack passwords.

2)JSON Web Tokens (JWTs):

They are a standardized way to securely send data between two parties. They contain information (claims) encoded in the JSON format. These claims help share specific details between the parties involved. At its core, a JWT is a mechanism for verifying the authenticity of some JSON data.

3)Jwtdecode:

jwt-decode JavaScript library that allows you to decode JSON Web Tokens (JWTs) without validating them. This can be useful for extracting information from a JWT, such as user details or token expiration time.

4)Universal-cookies:

Universal-cookies is a JavaScript library that allows you to manage cookies in both browser and server environments. It’s particularly useful for applications that need to handle cookies consistently across different platforms