BookNest: Where Stories Nestle

Team ID : LTVIP2025TMID53966
Team Members :
Vangara Bharath Kumar
Vaddi Gayathridevi
Veeramallu Deekshitha Maithreyi
Vakkalagaddda Likitha

INTRODUCTION

BookNest is a cutting-edge digital bookstore application tailored for the modern bibliophile. Built using the MERN Stack (MongoDB, Express.js, React, Node.js), BookNest offers a seamless and immersive literary browsing and shopping experience. Whether you're discovering new releases or revisiting classics, BookNest redefines how readers connect with literature in the digital age.

Key Highlights:

Immersive and responsive UI

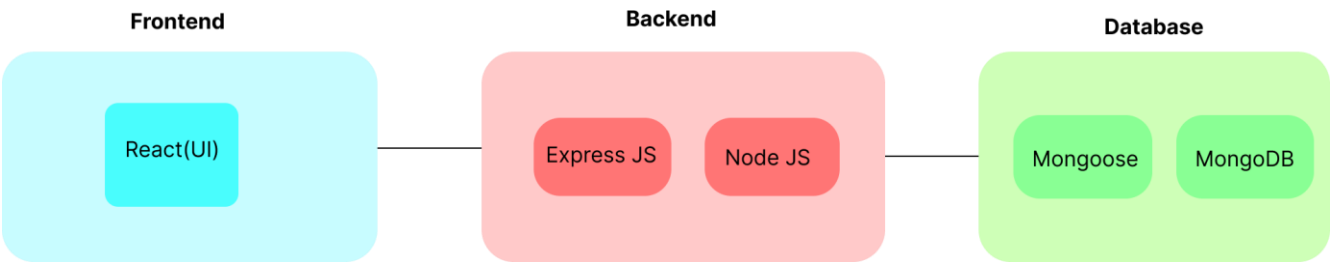
Personalized recommendations

Scalable and efficient backend

Secure and intuitive purchase experience.

PROJECT OVERVIEW

TECHNICAL ARCHITECTURE:



In this architecture diagram:

Frontend (React): User-facing interface including user authentication, book listings, cart, profile, and admin dashboard.

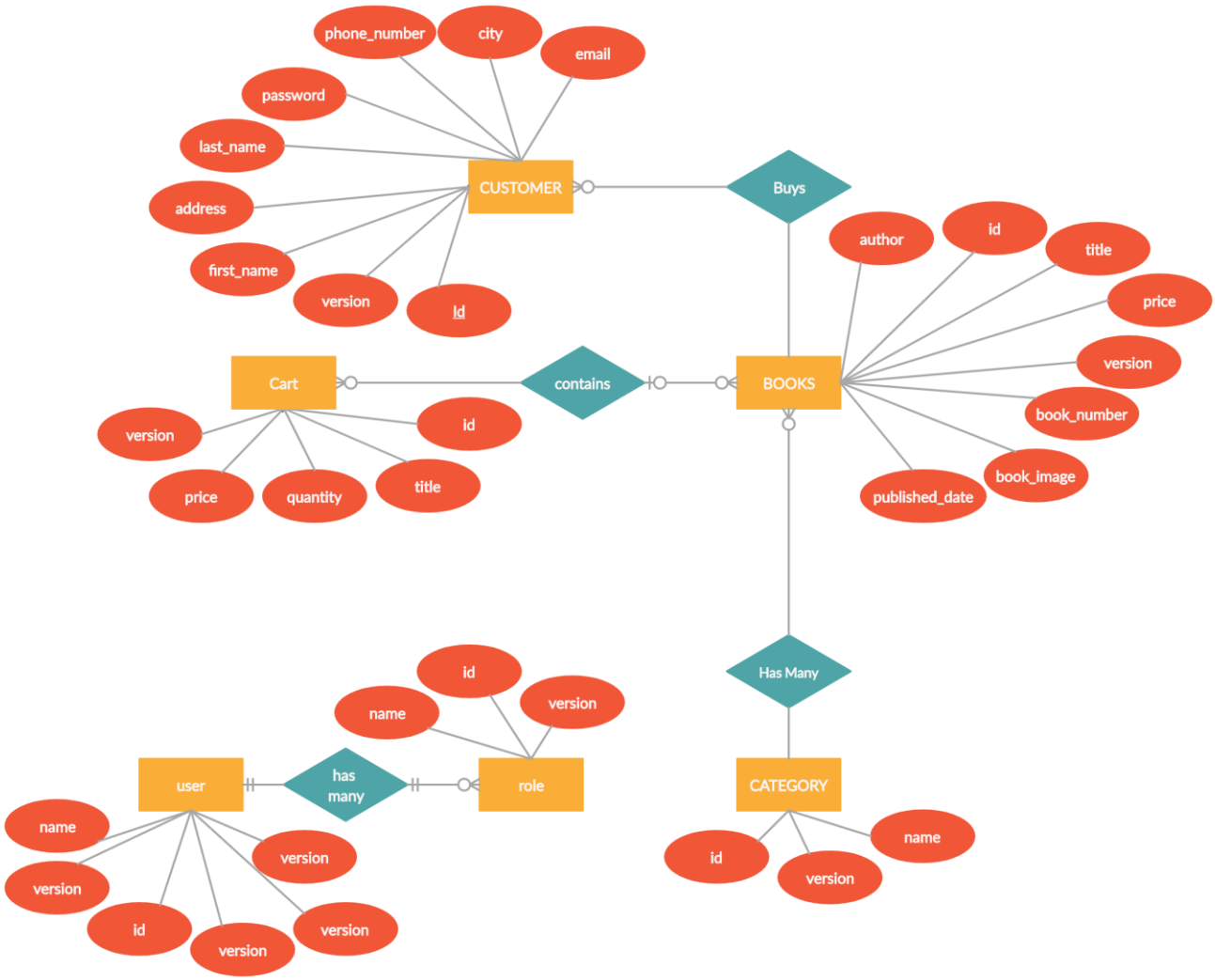
Backend (Node.js + Express.js): RESTful APIs for users, books, orders, admin controls.

Database (MongoDB): Collections for Users, Books, Cart, Orders, and Admin data.

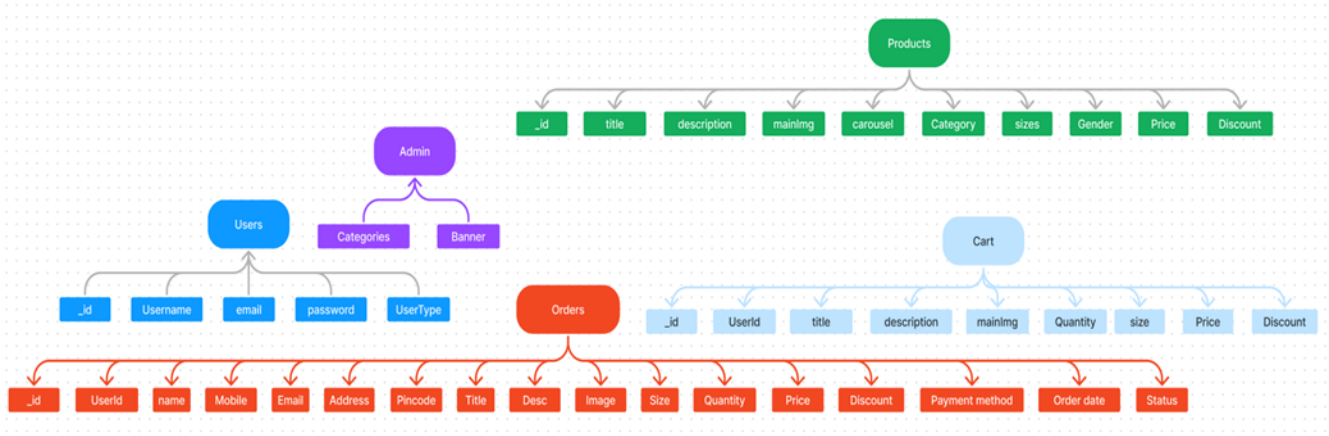
API Gateway: Central entry point for client requests, routing to appropriate services.

Authentication Service: Handles registration, login, JWT-based sessions, and role-based access.

ER DIAGRAM:



- The Database section represents the database that stores collections for Users, Admin, Cart, Orders and products.



The BookNest ER diagram illustrates the relationships between users, books, carts, and orders, ensuring seamless tracking of user interactions and transaction flows across the application.

User: User registration and profile

Admin: Manages books, orders, and categories

Books: Title, author, genre, price, description, stock

Cart: Tracks user-selected books before purchase

Orders: Stores completed transactions with details

PURPOSE:
BookNest is designed to provide an online bookstore where users can:

1.Easily register and authenticate

2.Discover and browse books by genre, author, or rating

3.Add books to cart and complete secure purchases

4.Track orders and review purchase history

5.Admins can manage books, users, and orders

FEATURES:

1.Comprehensive Book Catalog: Explore detailed listings with title, author, price, genre, reviews, and availability.

2.Personalized Discovery: Filter and search books by user preferences.

3.Secure Checkout: Streamlined order placement with secure payment integrations.

4.Order Confirmation: Post-purchase details including order ID, book list, and shipping info.

5.**User Dashboard**: View current and past orders, rate purchases, and track shipments.

6.Admin Dashboard: Add/edit/remove books, view sales data, manage users and orders.

PREREQUISITES:

To develop a full-stack e-commerce app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm:

Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: https://www.mongodb.com/try/download/community
- Installation instructions: https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing,middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command:
  npm install express

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

https://reactjs.org/docs/create-a-new-react-app.html

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling,and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: https://gitscm.com/downloads

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits

your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

To Connect the Database with Node JS go through the below provided link: •

Link: https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/

To run the existing BookNest App project downloaded from github:
Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the BookNest app.
- Execute the following command to clone the repository:

Git clone: https://github.com/yaswanthpuritipati/ShopEZ-e-commerce-MERN

Install Dependencies:

- Navigate into the cloned repository directory:
  cd SBookNest-MERN

- Install the required dependencies by running the following command: **npm install**

Start the Development Server:

- To start the development server, execute the following command:
  npm run dev or npm run start

- The e-commerce app will be accessible at http://localhost:4000 by default. You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to http://localhost:4000.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

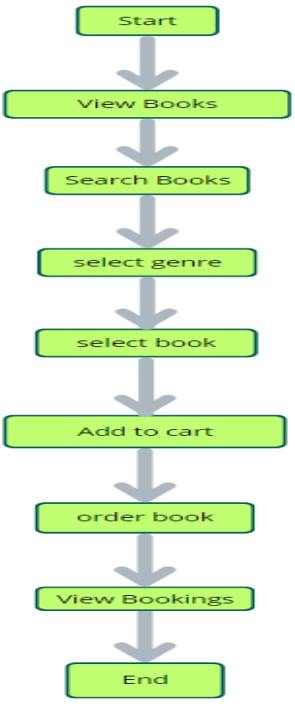
You have successfully installed and set up the BookNest app on your local machine. You can now

proceed with further customization, development, and testing as needed.

USER & ADMIN FLOW:

1. User Flow:

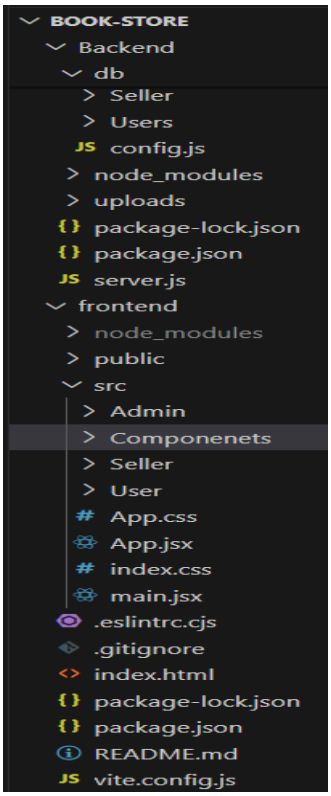
- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available books in the platform.
- Users can add the books they wish to their carts and order.
- They can then proceed by entering address and payment details.
- After ordering, they can check them in the profile section.



2. Admin Flow:

- Admins start by logging in with their credentials.
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, books, orders, etc.,

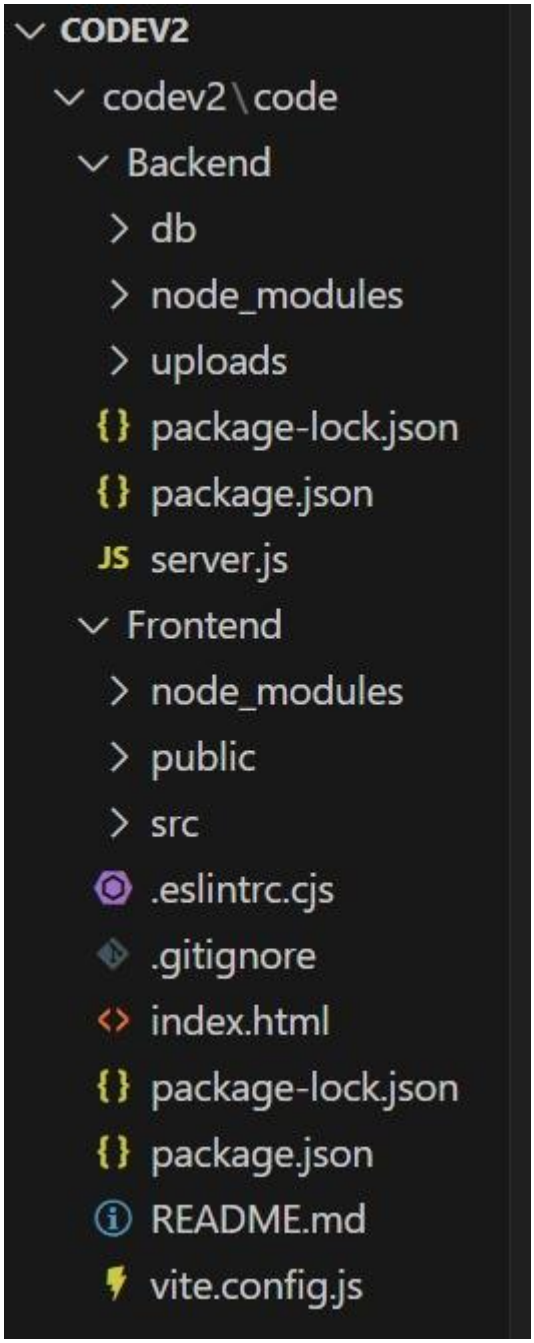
PROJECTSTRUCTURE:



This structure assumes a React app and follows a modular approach. Here's a brief explanation of the main directories and files:

• src/components: Contains components related to the application such as, register, login, home, etc.,

• src/pages has the files for all the pages in the application.



PROJECT SETUP AND CONFIGURATION:

Install required tools and software:

• Node.js.

Reference Article: https://www.geeksforgeeks.org/installation-of-node-js-on-windows/

• Git.

Reference Article: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

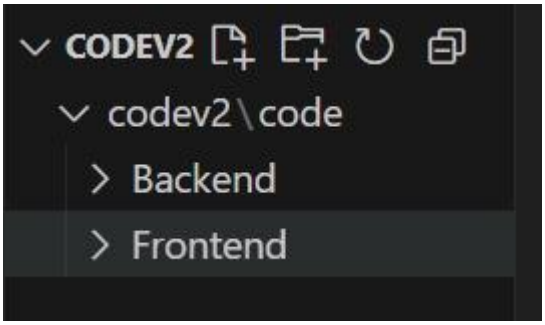
Create project folders and files:

• Client folders.

• Server folders

Referral Video Link:
https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QIjTpnYQ/view?usp=sharing

Referral Image:



DATABASE DEVELOPMENT:

Create database in cloud video link:- https://drive.google.com/file/d/1CQiI5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view

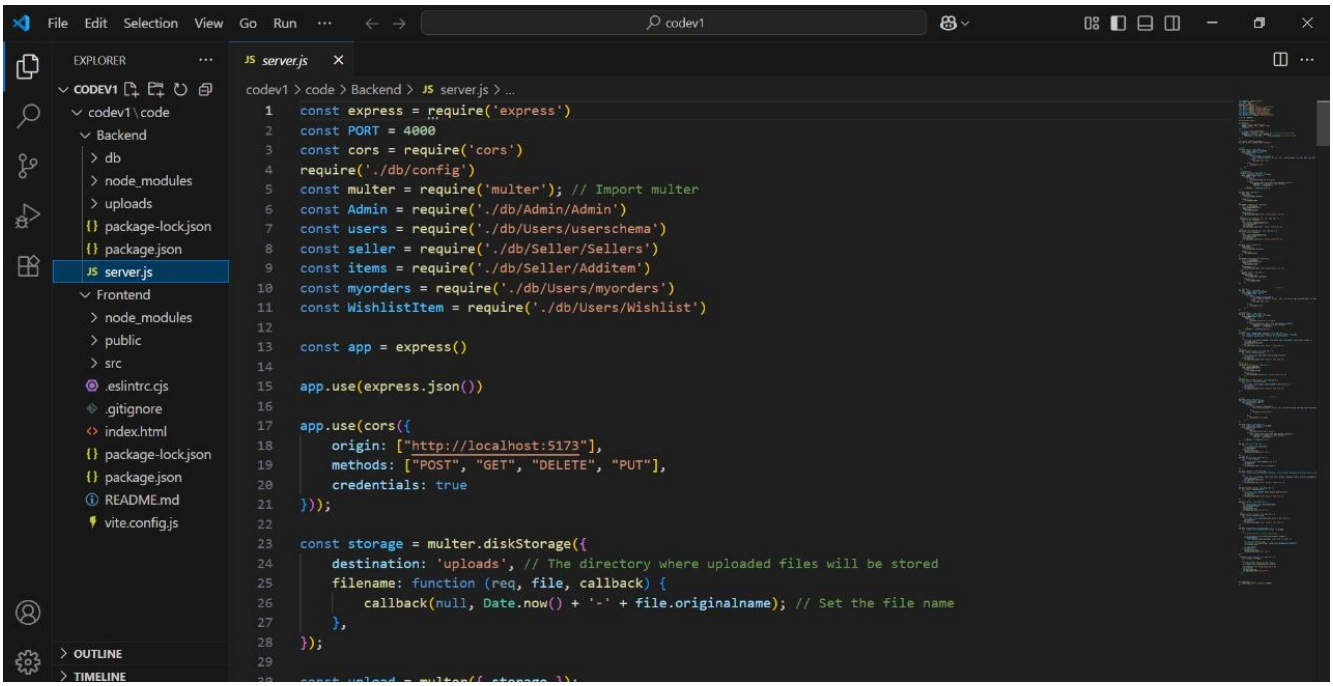
• Install Mongoose.

• Create database connection.

Reference Video of connect node with mongoDB database: https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

Reference Article: https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/

Reference Image:



Model use-case:

1. User Model:

- Model: ‘User’

- The User schema represents the user data and includes fields such as username, email, and password.
- It is used to store user information for registration and authentication purposes.
- The email field is marked as unique to ensure that each user has a unique email address

2. Book Model:

- Model: 'Book'
- The Book schema represents the data of all the books in the platform.
- It is used to store information about the book details, which will later be useful for ordering .

3. Order Model:

- Model: 'Order'
- The Orders schema represents the orders data and includes fields such as userId, book Id, book name, quantity, size, order date, etc.,
- It is used to store information about the orders made by users.
- The user Id field is a reference to the user who made the order.

Code Explanation:

```
app.post('/items', upload.single('itemImage'), async (req, res) => {

    try {
        const item = new items({ itemImage, title, author, genre, description, price, userId, userName });
        await item.save();
        res.status(201).json(item);
    } catch (err) {
        res.status(400).json({ error: 'Failed to create item' });
    }
});
//getbooks
app.get('/getitem/:userId', async (req, res) => {
    const userId = req.params.userId;
    try {
        const tasks = await items.find({ userId }).sort('position');
        res.json(tasks);
    } catch (err) {
        res.status(500).json({ error: 'Failed to fetch tasks' });
    }
});
//delete book
app.delete('/itemdelete/:id', (req, res) => {
    const { id } = req.params;
    items.findByIdAndDelete(id)
        .then(() => {
            res.sendStatus(200);
        })
        .catch((error) => {
            res.status(500).json({ error: 'Internal server error' });
```

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate, useParams } from 'react-router-dom';
import '../Seller/List.css';
import Unavbar from './Unavbar';

function OrderItem() {
  const [item, setItem] = useState({});
  const [formData, setFormData] = useState({
    flatno: '',
    city: '',
    pincode: '',
    state: '',
  });
  const fee= 99;
  const { id } = useParams();
  const navigate = useNavigate();

  useEffect(() => {
    axios.get(`http://localhost:4000/item/${id}`)
      .then((resp) => {
        setItem(resp.data);
      })
      .catch((error) => {
        console.log("Failed to fetch item data:", error);
      });
  }, [id]);
```

BACKEND DEVELOPMENT:

Setup express server:

• Create index.js file.

• Create an express server on your desired port number.

• Define API's

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

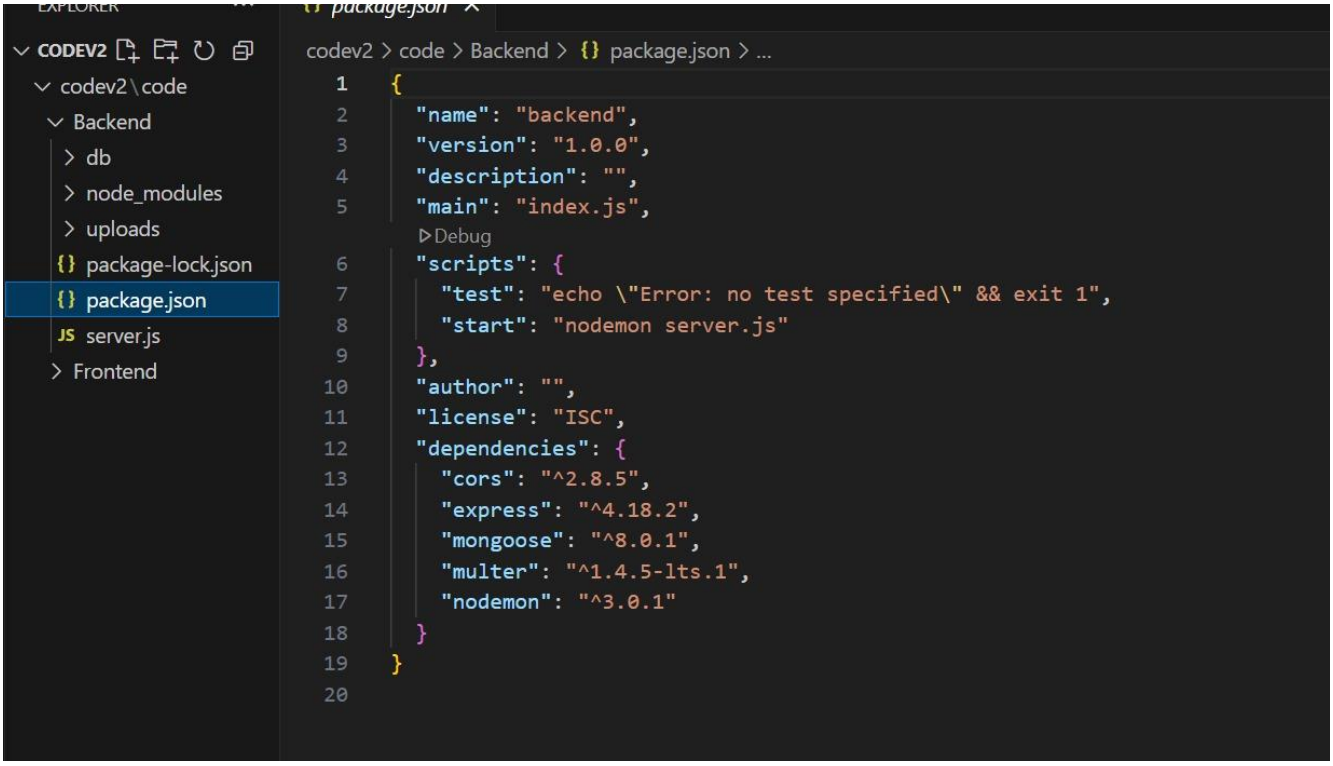
Set Up Project Structure:

• Create a new directory for your project and set up a package.json file using the npm init command.

• Install necessary dependencies such as Express.js, Mongoose, and other required packages.

Reference Video: https://drive.google.com/file/d/19df7NU-gQK3DO6wr7ooAfJYIQwnemZoF/view?usp=sharing

Reference Images:



2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing books, handling user registration and login,managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the

database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like books, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new book listings, including fetching book data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding books, managing user orders, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

WEB DEVELOPMENT:

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.Design UI components:

- Create Components.
- Implement layout and styling.

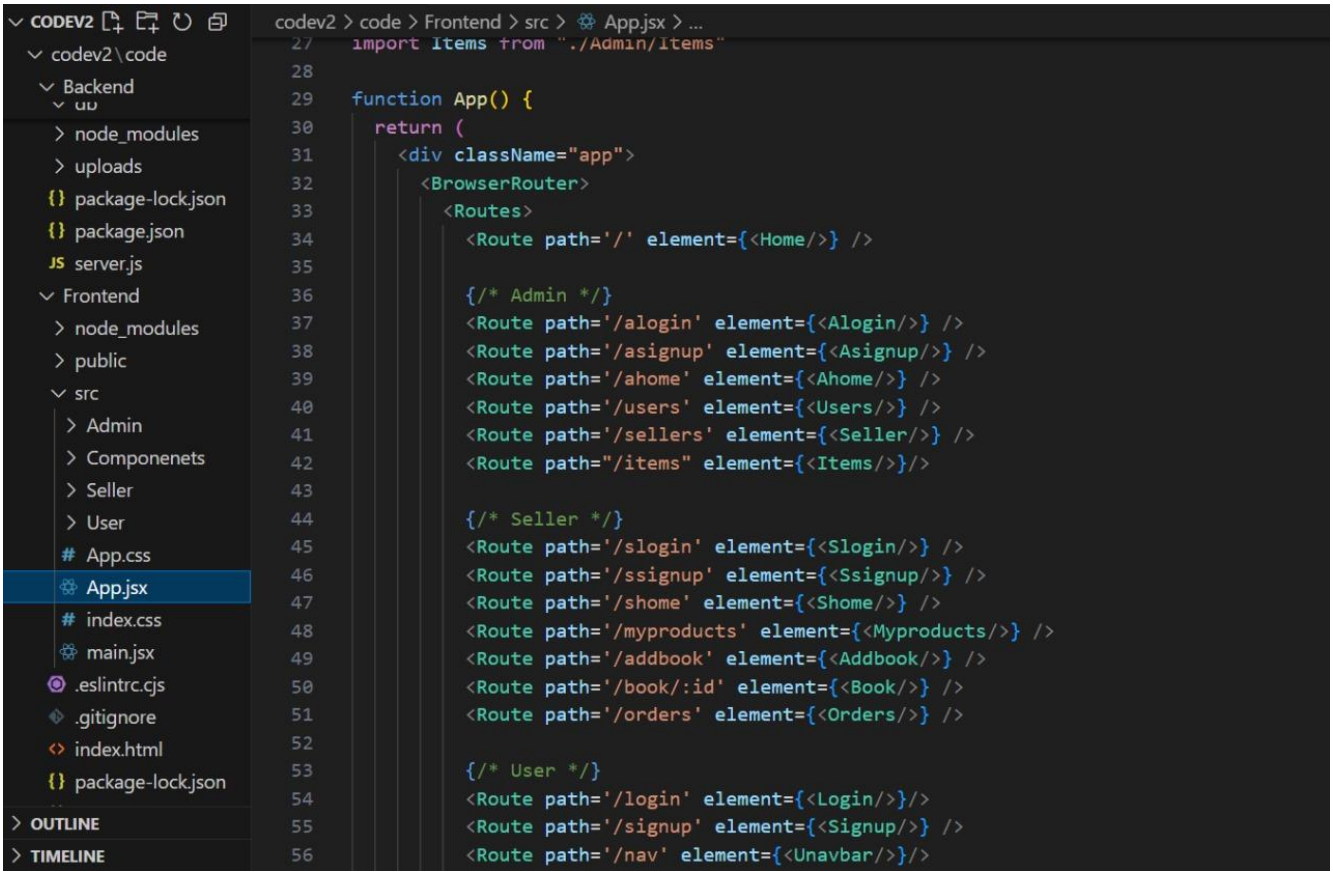
• Add navigation.

3.Implement frontend logic:
• Integration with API endpoints.
• Implement data binding.

Reference Video Link:
https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing

Reference Article Link:
https://www.w3schools.com/react/react_getstarted.asp

Reference Image:



PROJECT IMPLEMENTATION & EXECUTION:

User Authentication:

· **Backend**

Now, here we define the functions to handle http requests from the client for authentication.

auth.js

server > routes > auth.js > router.post('/register') callback > error

```
// Login
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ error: 'Invalid credentials' });
    const match = await bcrypt.compare(password, user.password);
    if (!match) return res.status(400).json({ error: 'Invalid credentials' });
    const token = jwt.sign({ id: user._id, isAdmin: user.isAdmin }, JWT_SECRET, { expiresIn: '7d' });
    res.json({ token, user: { id: user._id, name: user.name, email: user.email, isAdmin: user.isAdmin } });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;
```

Frontend

Login:

CODEV2

- codev2\code
  - Backend
  - Frontend
    - node_modules
    - public
    - src
      - Admin
      - Componenets
      - Seller
      - User
        - Login.jsx
        - Myorders.jsx
        - OrderItem.jsx
        - Products.jsx
        - Signup.jsx
        - uhome.css
        - Uhome.jsx
        - Uitem.jsx
        - Unavbar.jsx
        - Wishlist.jsx
      - App.css
      - App.jsx
      - index.css
      - main.isx
- OUTLINE

codev2 > code > Frontend > src > User > Login.jsx > Login > handleSubmit > then() callback

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import Home from '../Componenets/Home';

const Login = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  axios.defaults.withCredentials = true;

  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);

    let payload = { email, password };
    axios
      .post("http://localhost:4000/login", payload)
      .then((res) => {
        console.log("login: " + res.data.Status);
        if (res.data.Status === "Success") {
          console.log(res.data.user);
          localStorage.setItem('user', JSON.stringify(res.data.user));
          navigate('/uhome');
          alert("Login successful");
        } else {
          alert("Wrong credentials");
```

Register:

logout:

```
const logout = () => {
  localStorage.removeItem('user');
  localStorage.removeItem('token');
  setUser(null);
};
```

All books (User):

In the home page, we'll fetch all the books available in the platform along with the filters.

Fetching books:

```
CODEV2
codev2\code
  backend
  Frontend
    node_modules
    public
    src
      Admin
      Componenets
      Seller
      User
        Login.jsx
        Myorders.jsx
        OrderItem.jsx
        Products.jsx
        Signup.jsx
        uhome.css
        Uhome.jsx
        Uitem.jsx
        Unavbar.jsx
        Wishlist.jsx
      App.css
      App.jsx
      index.css
      main.jsx
OUTLINE
TIMELINE
```

codev2 > code > Frontend > src > User > Products.jsx > ...

```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import Unavbar from './Unavbar';
import { Link } from 'react-router-dom';

function Products() {
  const [items, setItems] = useState([]);
  const [wishlist, setWishlist] = useState([]);
  const [loading, setLoading] = useState(true);
  const [searchTerm, setSearchTerm] = useState('');
  const [selectedGenre, setSelectedGenre] = useState('all');

  useEffect(() => {
    // Fetch all items
    axios
      .get(`http://localhost:4000/item`)
      .then((response) => {
        const taskData = response.data;
        setItems(taskData);
        setLoading(false);
      })
      .catch((error) => {
        console.error('Error fetching tasks: ', error);
        setLoading(false);
      });

    // Fetch wishlist items
    const user = JSON.parse(localStorage.getItem('user'));
    if(user){
      axios.get(`http://localhost:4000/wishlist/${user.id}`)
```

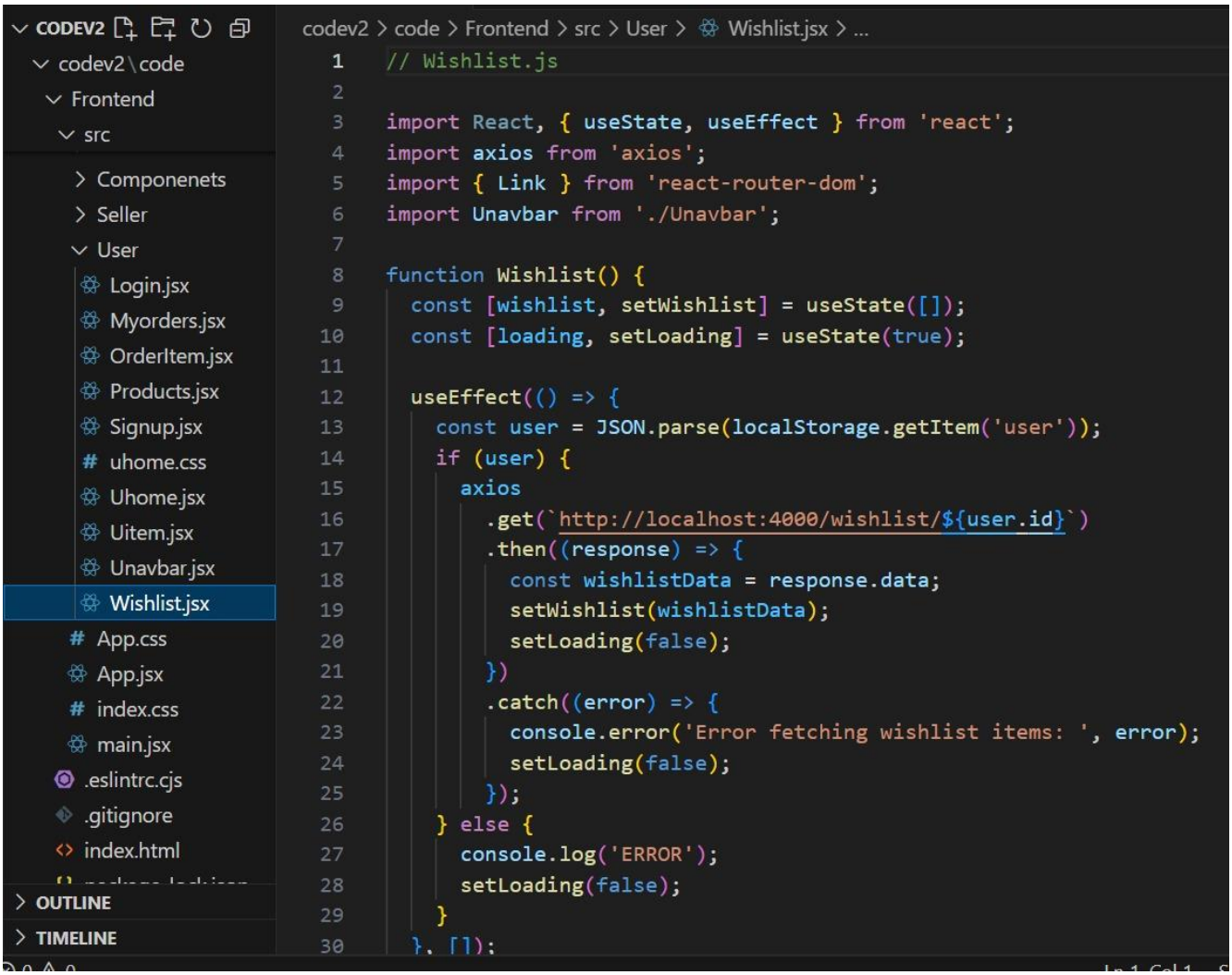
In the backend, we fetch all the products and then filter them on the client side.

```
CODEV2
codev2\code
  backend
  Frontend
    node_modules
    public
    src
      Admin
      Componenets
      Seller
      User
        Login.jsx
        Myorders.jsx
        OrderItem.jsx
        Products.jsx
        Signup.jsx
        uhome.css
        Uhome.jsx
        Uitem.jsx
        Unavbar.jsx
        Wishlist.jsx
      App.css
      App.jsx
      index.css
      main.jsx
OUTLINE
TIMELINE
```

codev2 > code > Frontend > src > User > Uitem.jsx > ...

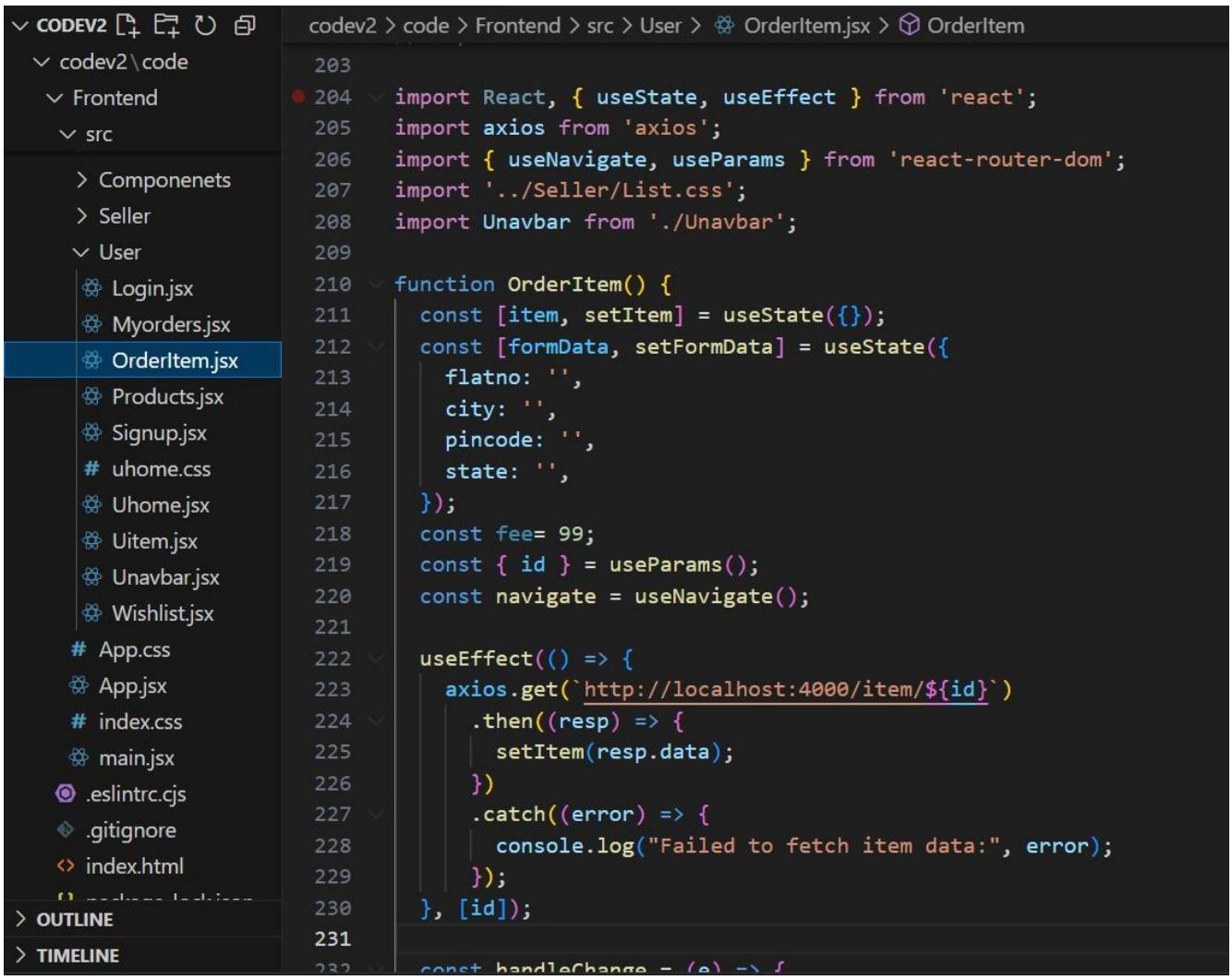
```jsx
import axios from 'axios';
import React, { useEffect, useState } from 'react';
import { Link, useParams } from 'react-router-dom';
import Unavbar from './Unavbar';

const Uitem = () => {
    const [item, setItem] = useState(null);
    const [loading, setLoading] = useState(true);
    const [wishlist, setWishlist] = useState([]);
    const [isInWishlist, setIsInWishlist] = useState(false);

    const { id } = useParams();

    useEffect(() => {
        // Fetch book details
        axios.get(`http://localhost:4000/item/${id}`)
            .then((resp) => {
                console.log(resp);
                setItem(resp.data);
                setLoading(false);
            })
            .catch(() => {
                console.log("Did not get data");
                setLoading(false);
            });

        // Fetch wishlist to check if book is saved
        const user = JSON.parse(localStorage.getItem('user'));
        if (user) {
            axios.get(`http://localhost:4000/wishlist/${user.id}`)
```

Wishlist:



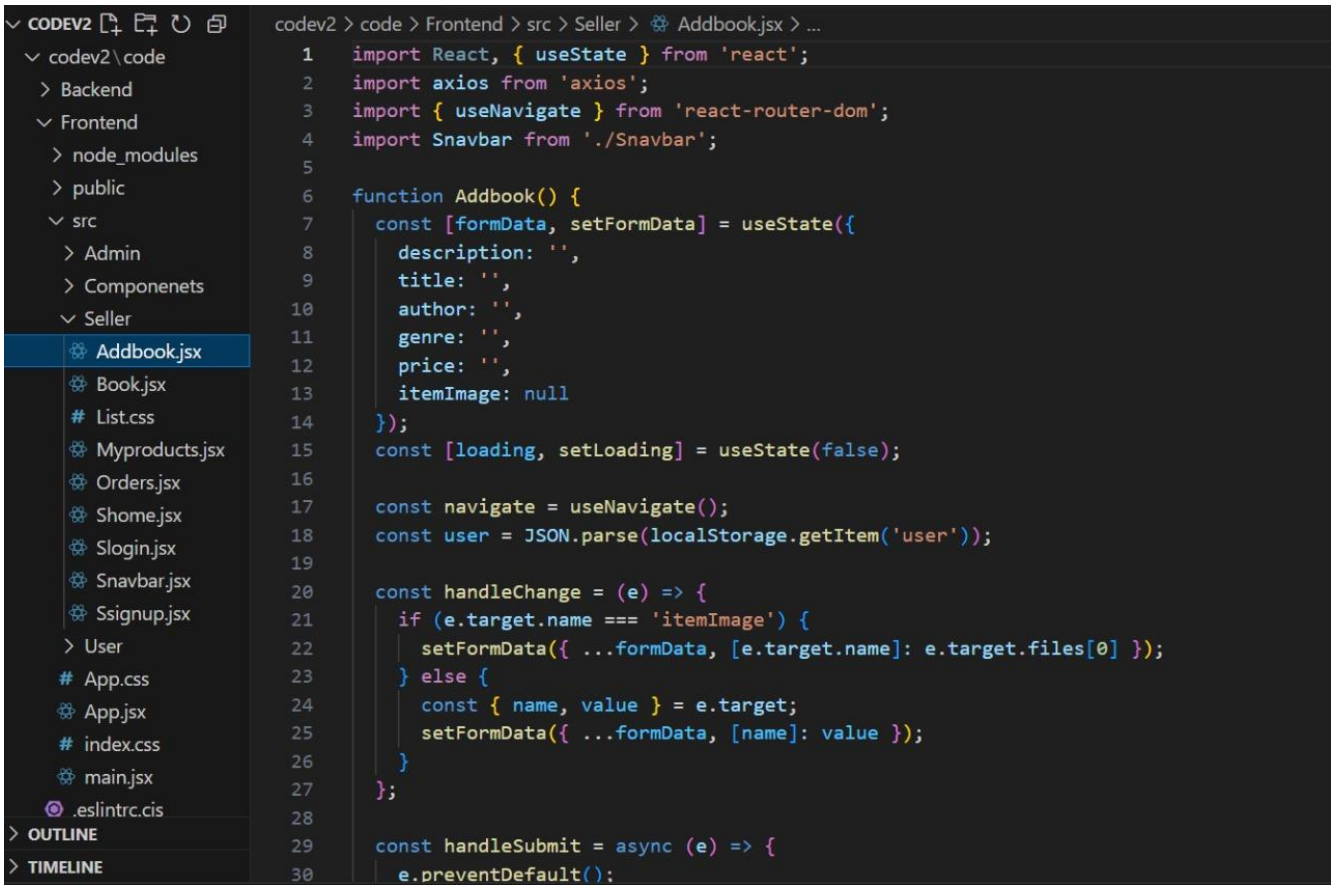
Order Item:

Here, we can add the product to the cart or can buy directly.

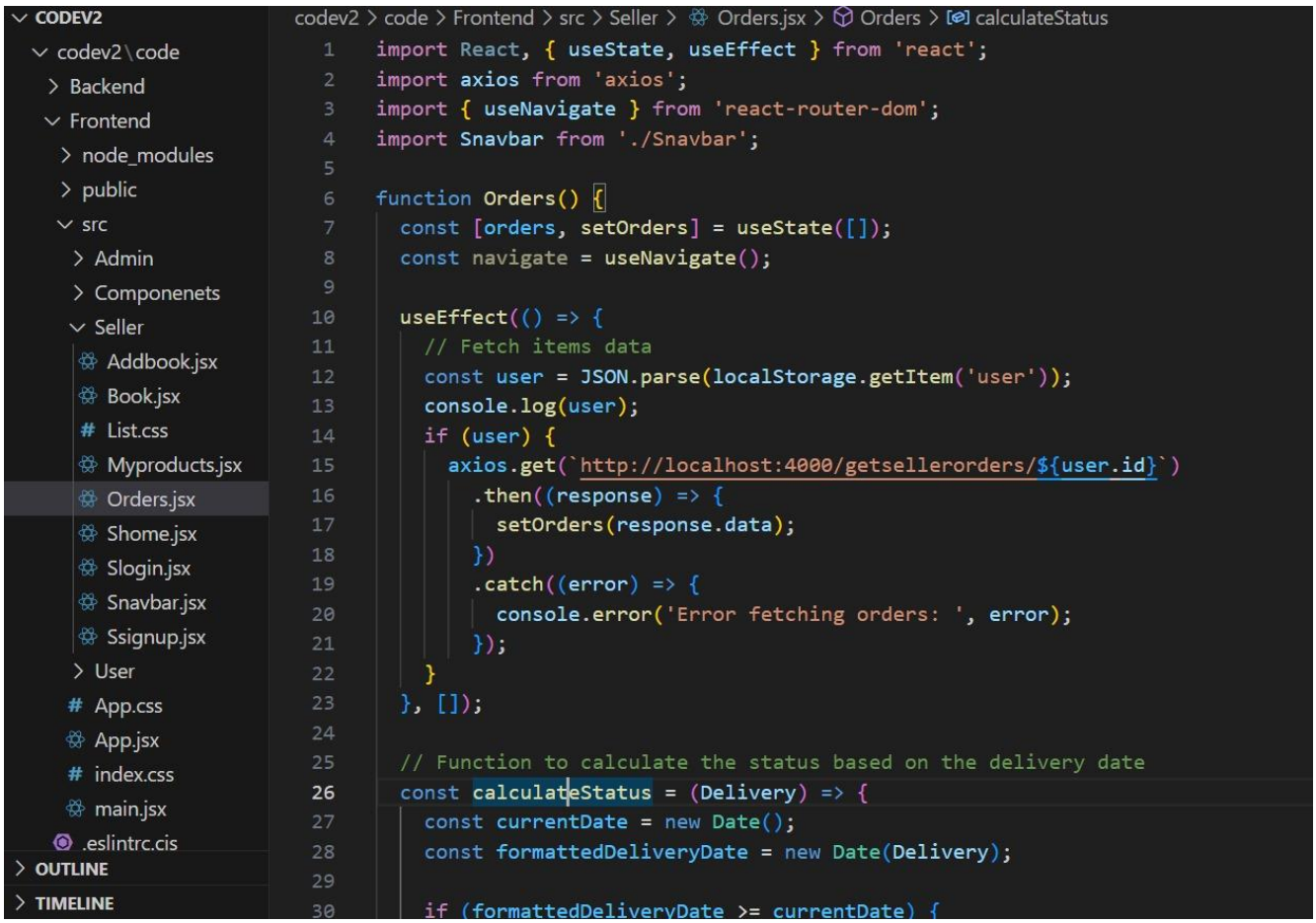


· Backend**:** In the backend, if we want to buy, then with the address and payment method, we process buying. If we need to add the product to the cart, then we add the product details along with the user Id to the cart collection.

Add Book:



My Orders:



Book:

```
codev2 > code > Frontend > src > Seller > Book.jsx > ...
```

```jsx
import axios from 'axios';
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import Vnavbar from './Snavbar';

const Book = () => {
  const [item, setItem] = useState(null); // Initialize item as null

  const { id } = useParams();

  useEffect(() => {
    axios.get(`http://localhost:4000/item/${id}`)
      console.log(resp);
      setItem(resp.data); // Set item to the fetched data (an object, not an array)
    })
    .catch(() => {
      console.log("Did not get data");
  }, [id]); // include 'id' as a dependency to re-fetch data when the ID changes

  return (
    <div>
      <Vnavbar />
      <br />
      {item && (
```

Add Item:

```
codev2 > code > Backend > db > Seller > Additem.js > ...
```

```jsx
const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
      },
      author: {
        type: String,
        required: true,
      },
      genre: {
        type: String,
        required: true,
      },
    // itemtype:String,
    itemImage:String,
    description:String,
    price:String,
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    userName:String,
})

module.exports =mongoose.model('books',bookSchema)
```

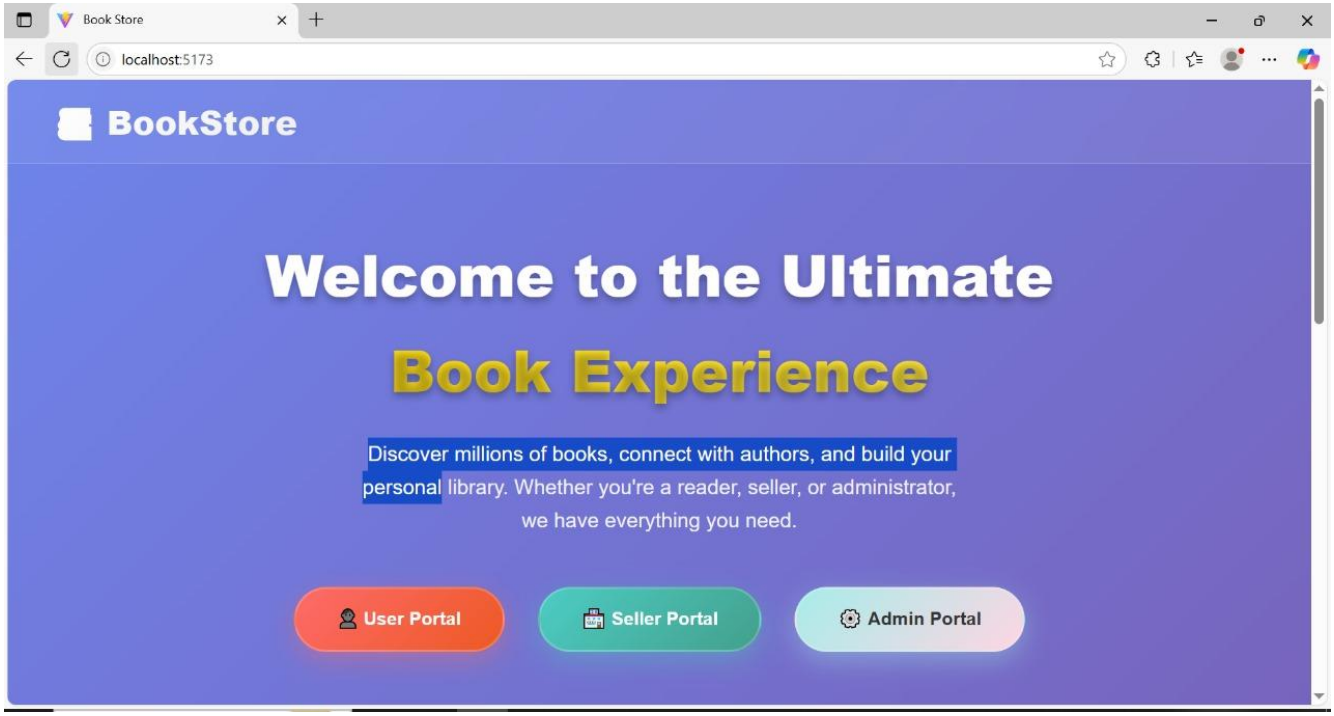
My orders:

```
CODEV2
codev2\code
Backend
db
Admin
Admin.js
Seller
Additem.js
Sellers.js
Users
myorders.js
userschema.js
Wishlist.js
config.js
node_modules
uploads
package-lock.json
package.json
server.js
Frontend
node_modules
public
src
Admin
Componenets
OUTLINE
TIMELINE
```

```
codev2 > code > Backend > db > Users > myorders.js > ...
const mongoose = require('mongoose');

const bookschema = new mongoose.Schema({
    flatno:String,
    pincode:String,
    city:String,
    state:String,
    totalamount:String,
    seller:String,
    sellerId:String,
    booktitle:String,
    bookauthor:String,
    bookgenre:String,
    itemImage:String,
    description:String,
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    userName:String,
    BookingDate: {
        type: String, // Store dates as strings
        default: () => new Date().toLocaleDateString('hi-IN') // Set the default value t
    },
    Delivery: {
      type: String, // Store dates as strings
      default: () => {
        // Set the default value to the current date + 7 days in "MM/DD/YYYY" format
        const currentDate = new Date();
        currentDate.setDate(currentDate.getDate() + 7); // Add 7 days
        const day = currentDate.getDate();
        const month = currentDate.getMonth() + 1; // Month is zero-based, so add 1
        const year = currentDate.getFullYear();
```

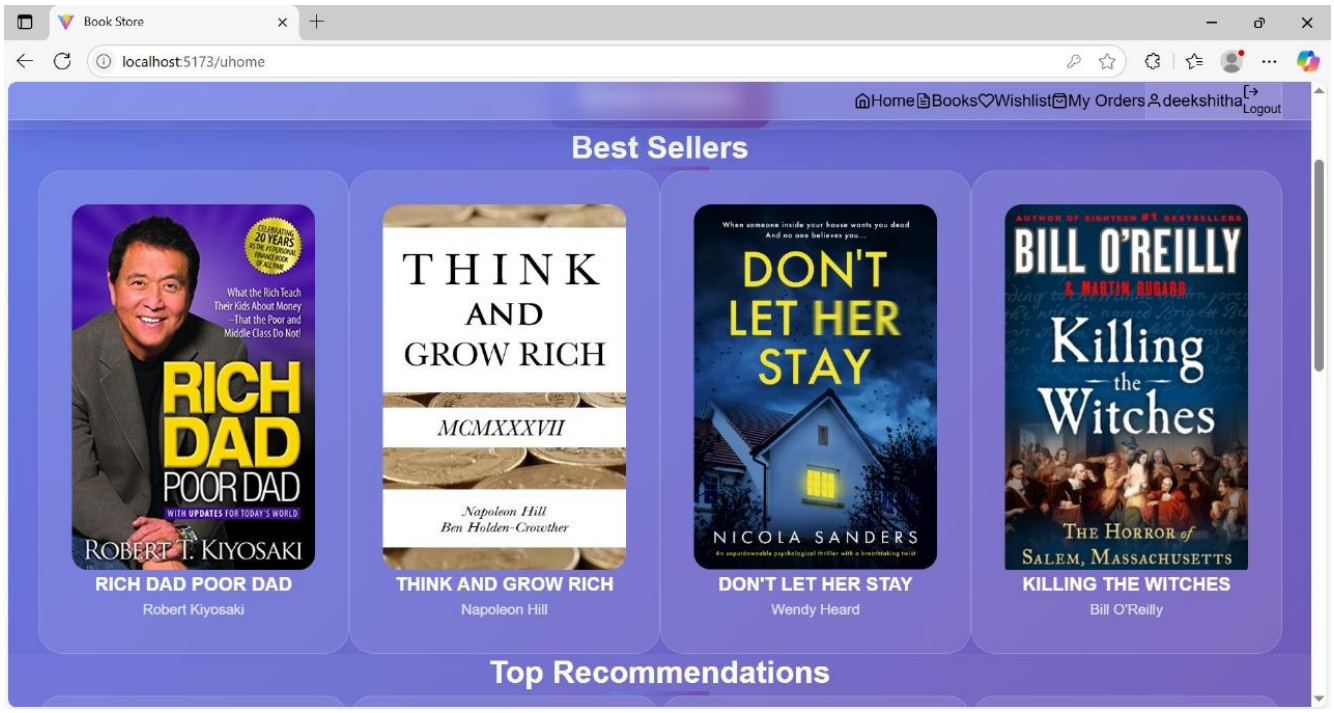
Demo UI images:

- **Landing page**

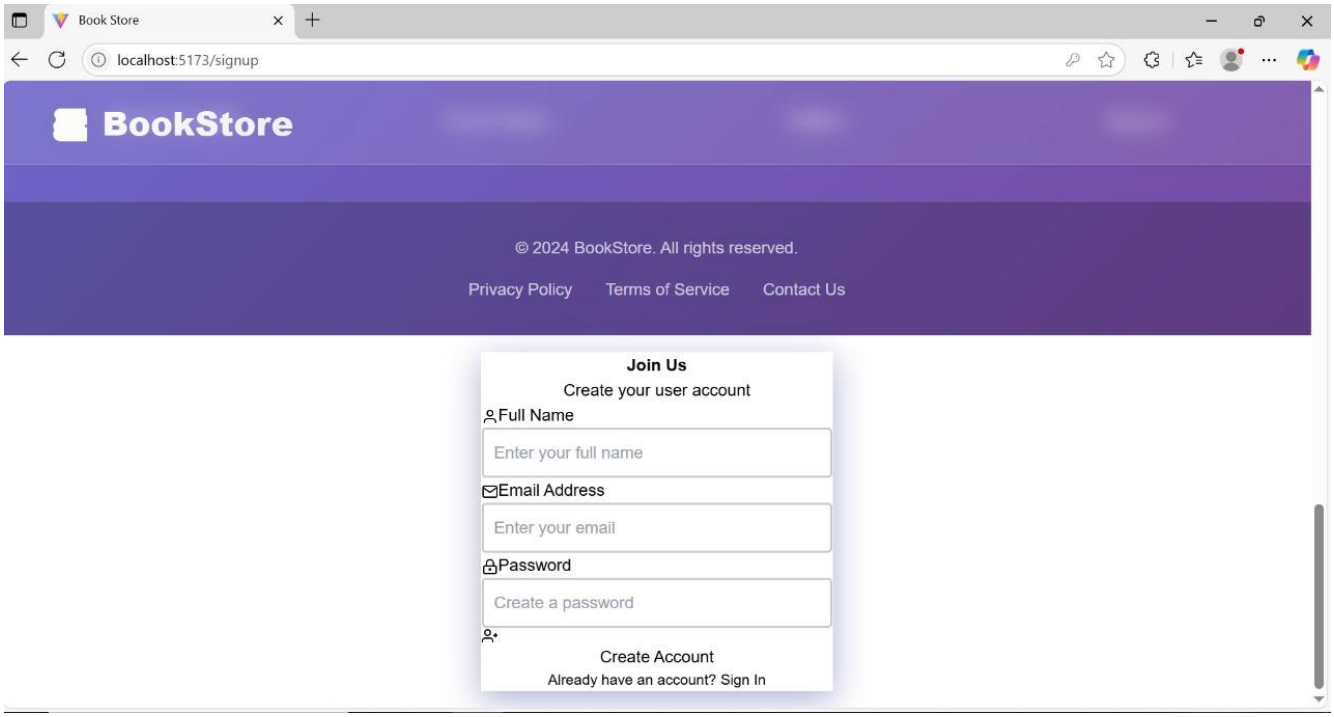


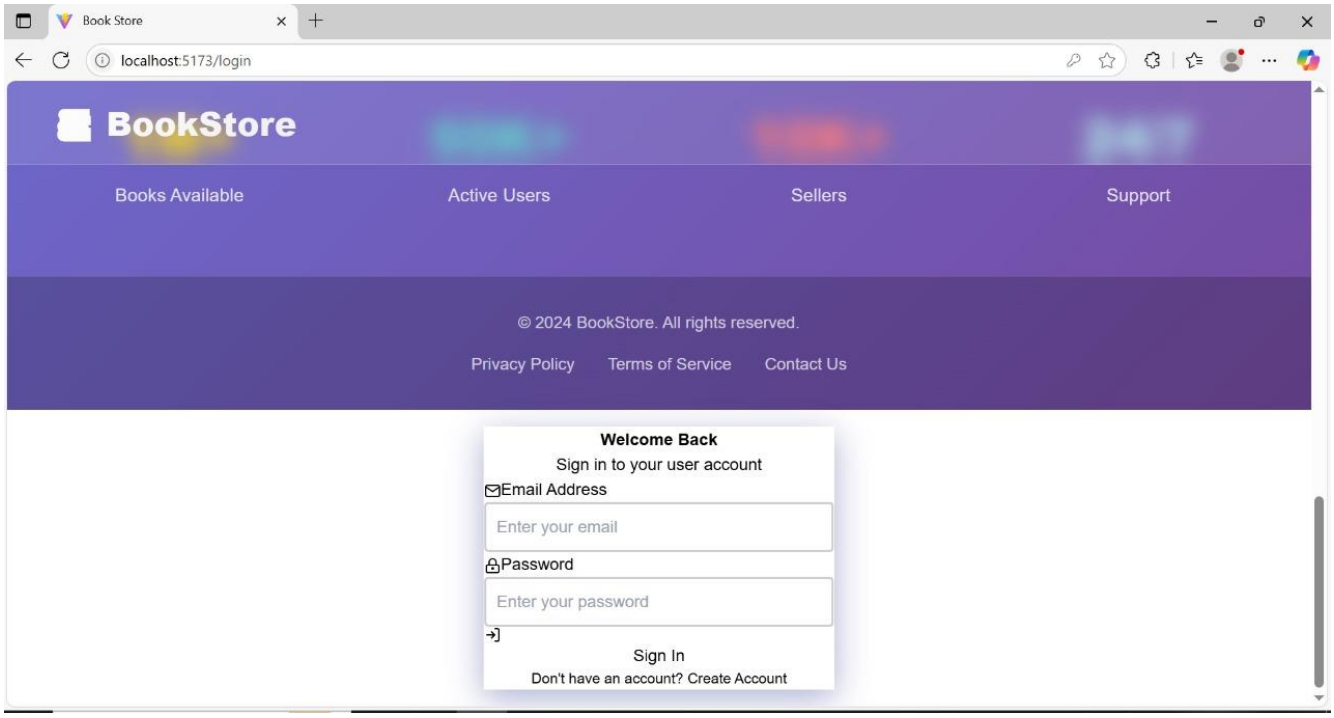
.

books



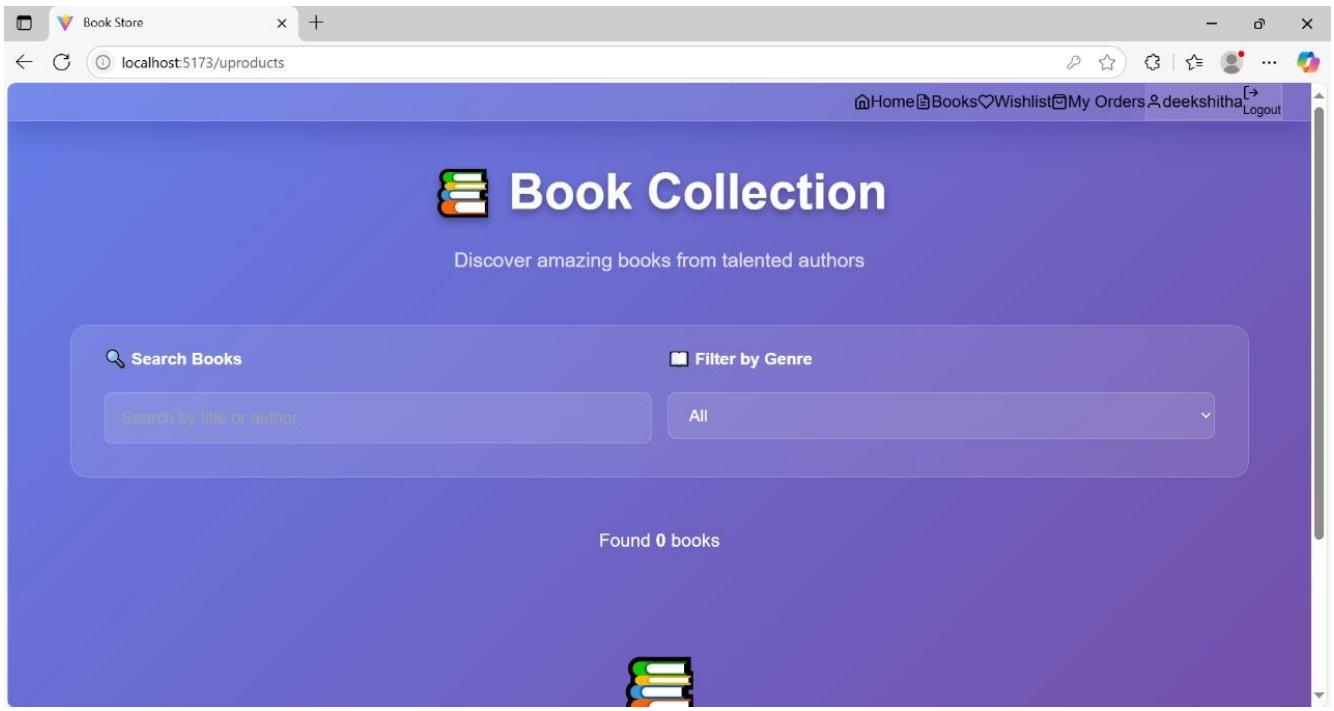
Authentication



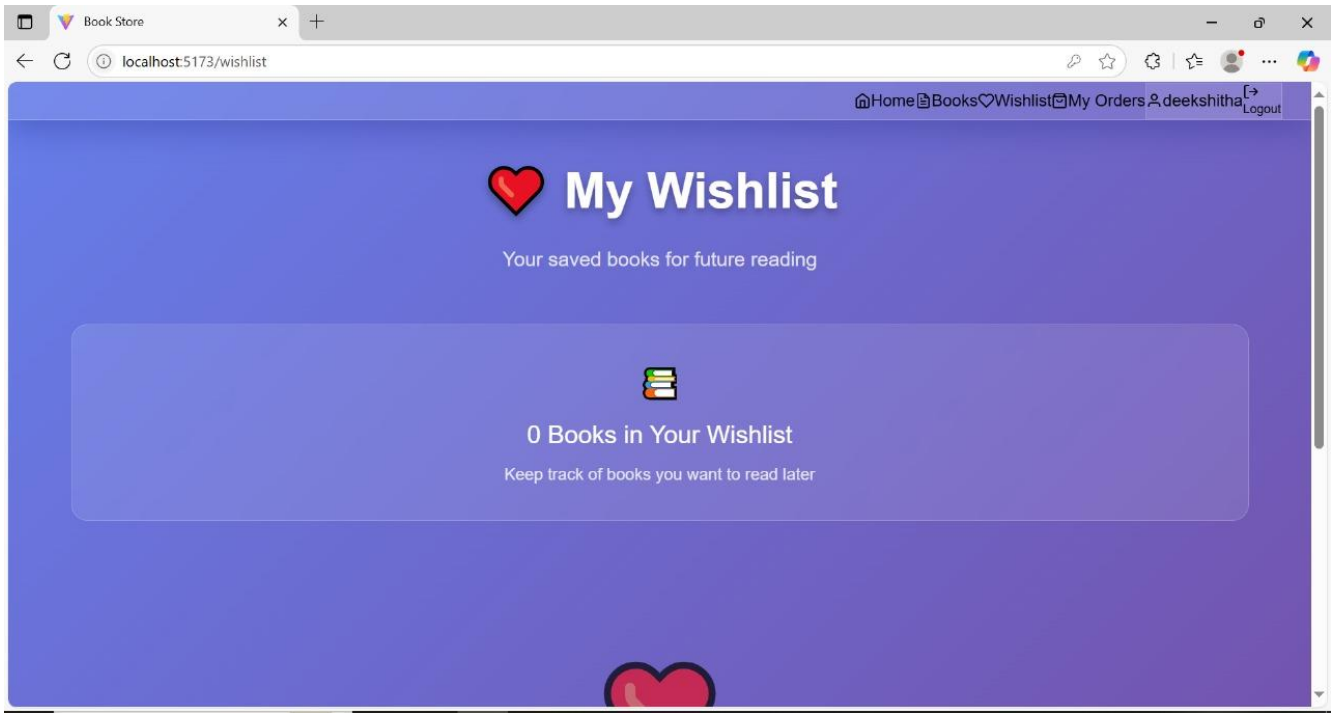


.

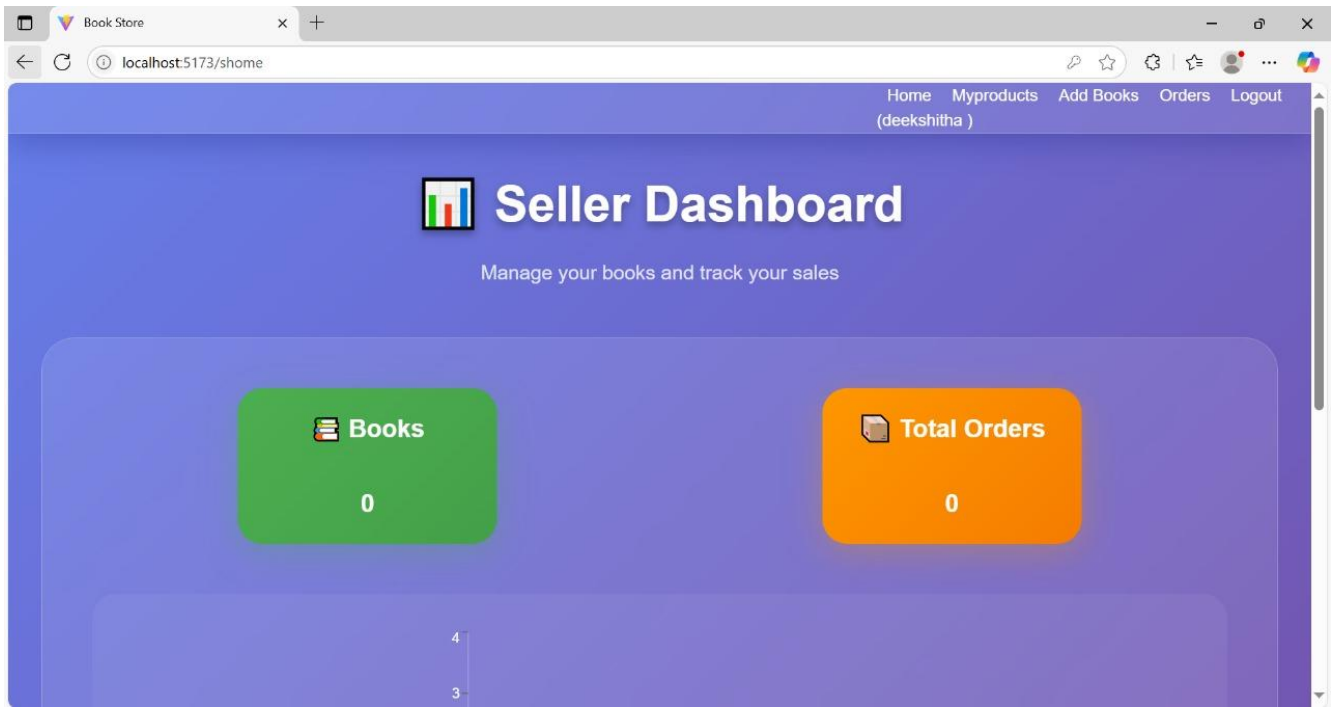
Book Collections:



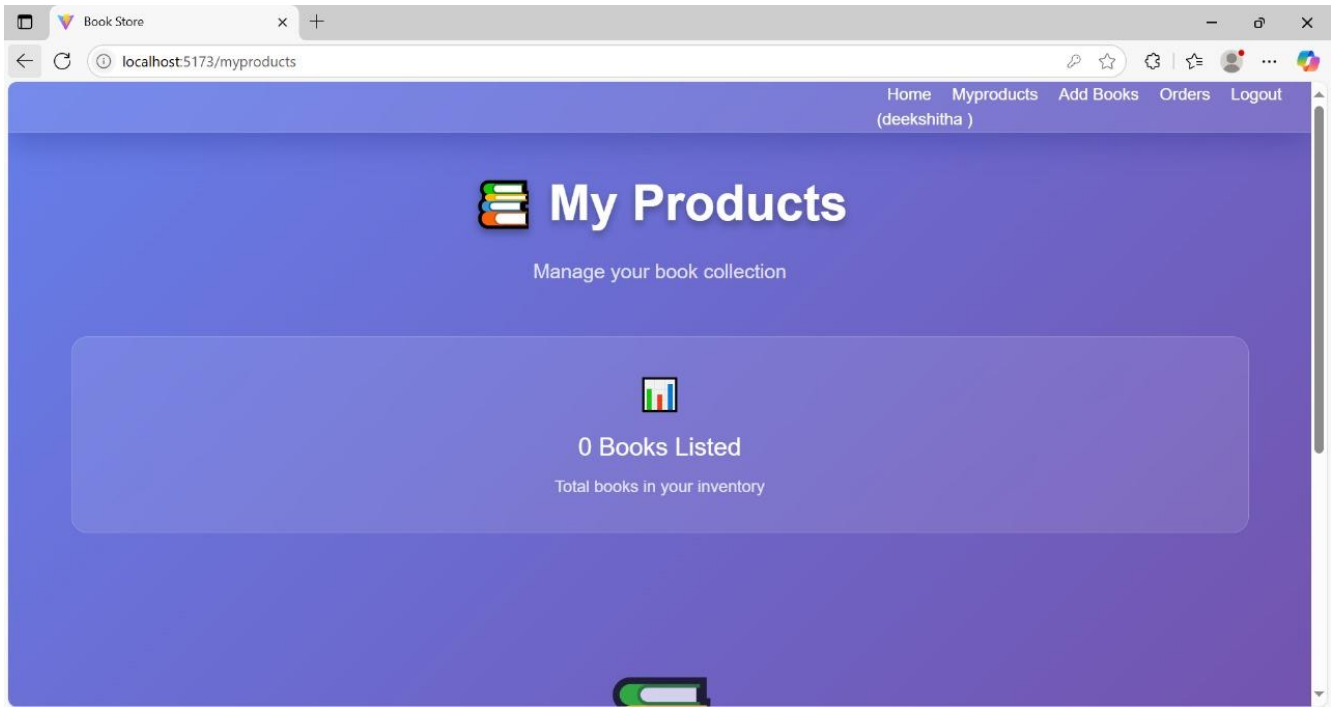
Mywishlist:



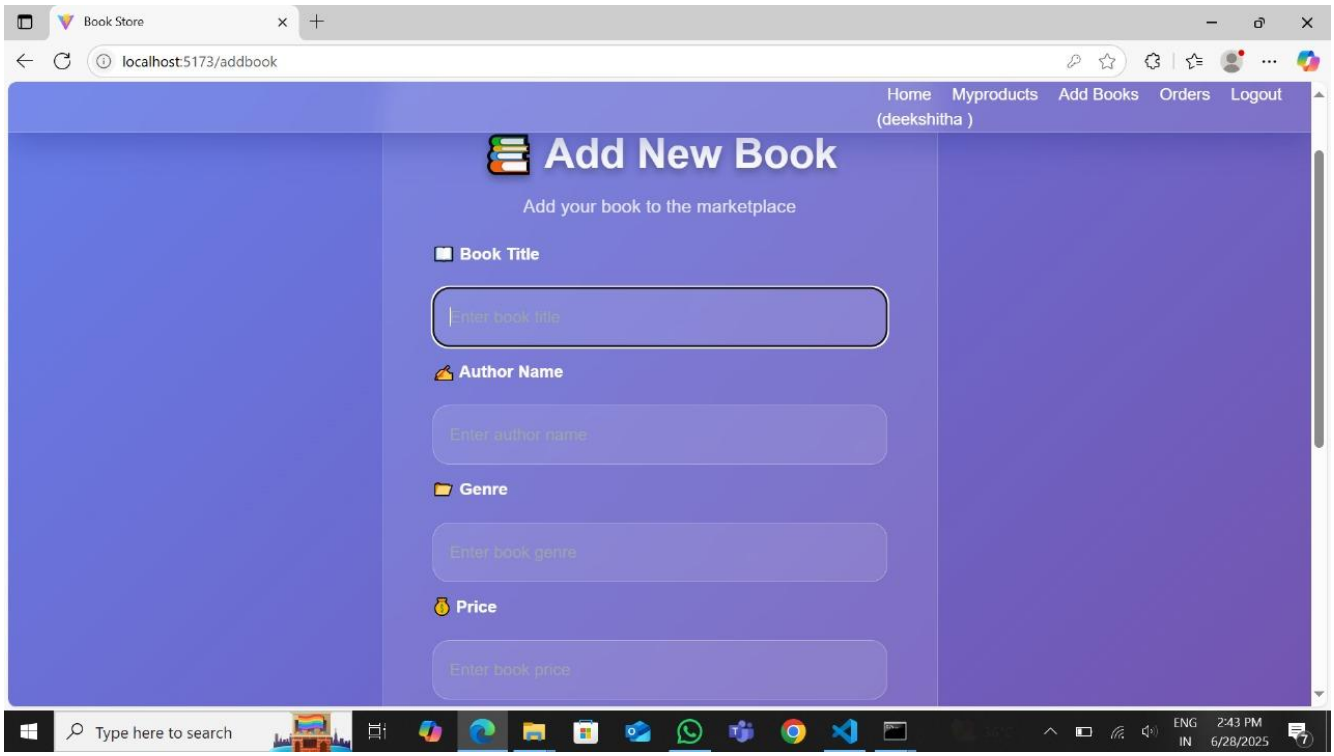
Seller dashboard



My Projects:

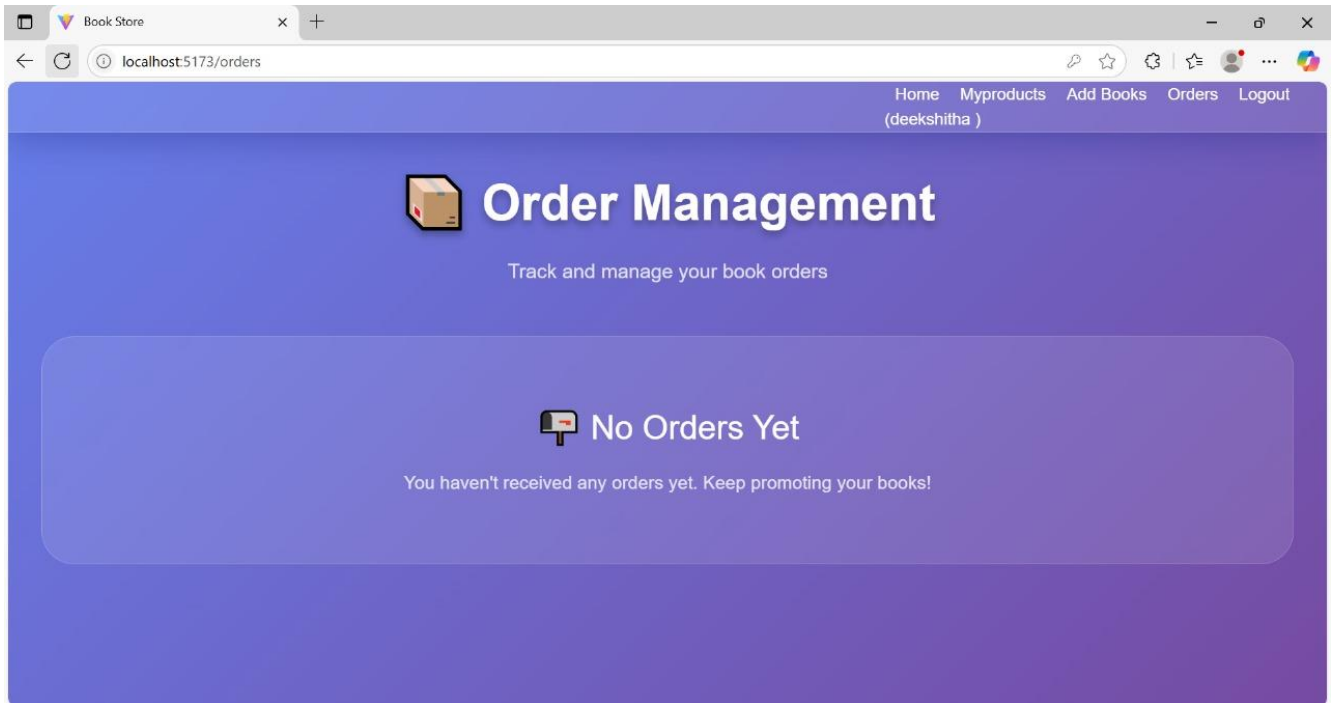


Add Books:

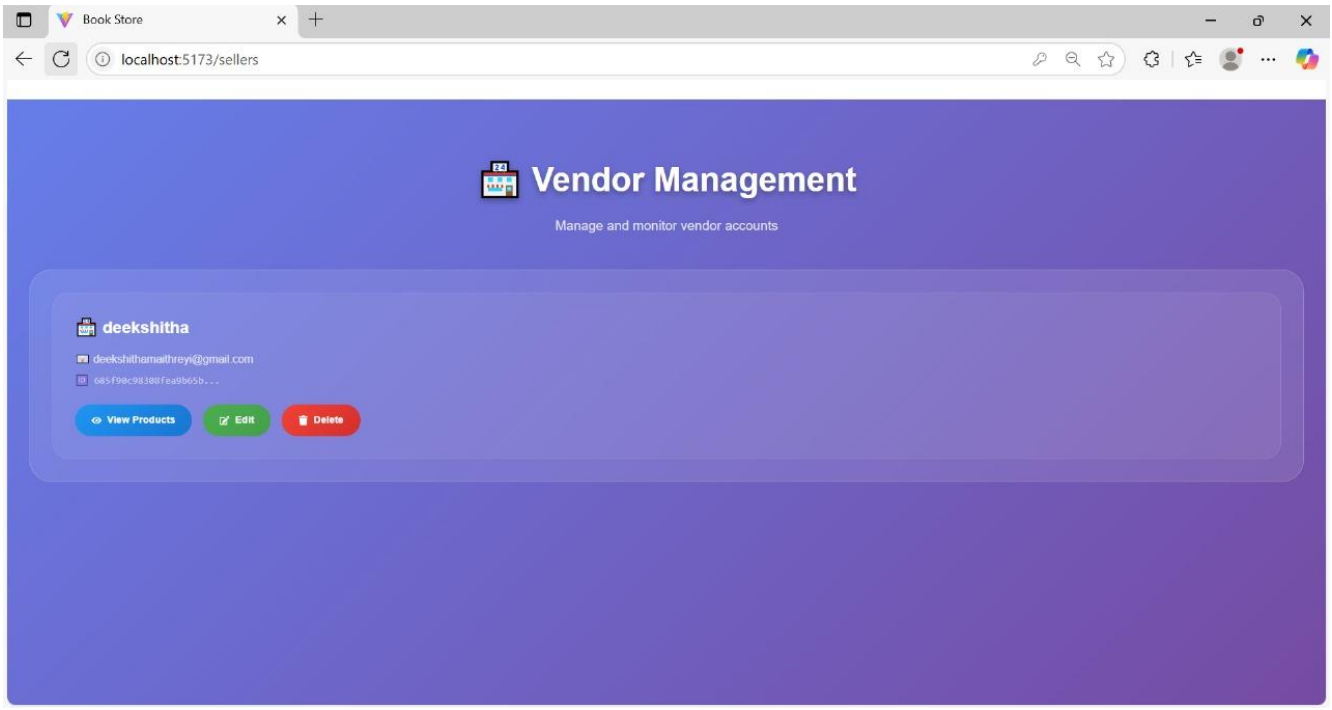


.

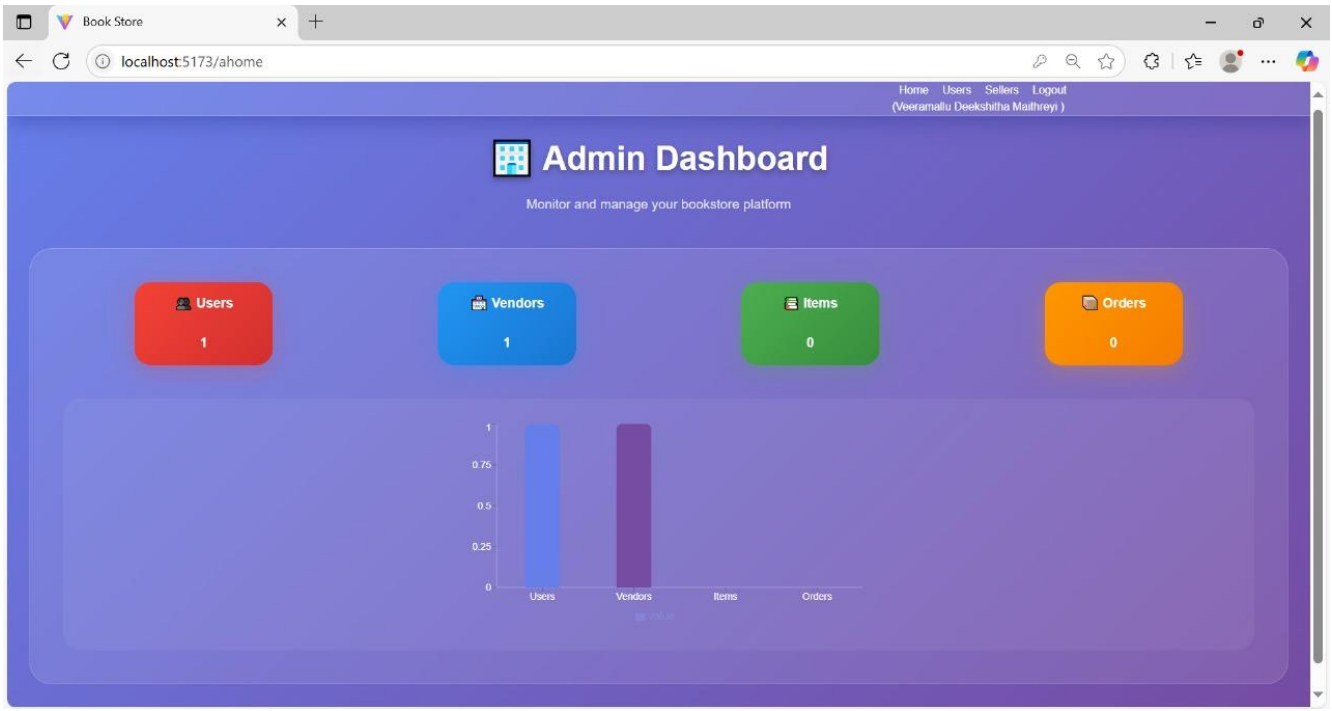
Order Management:



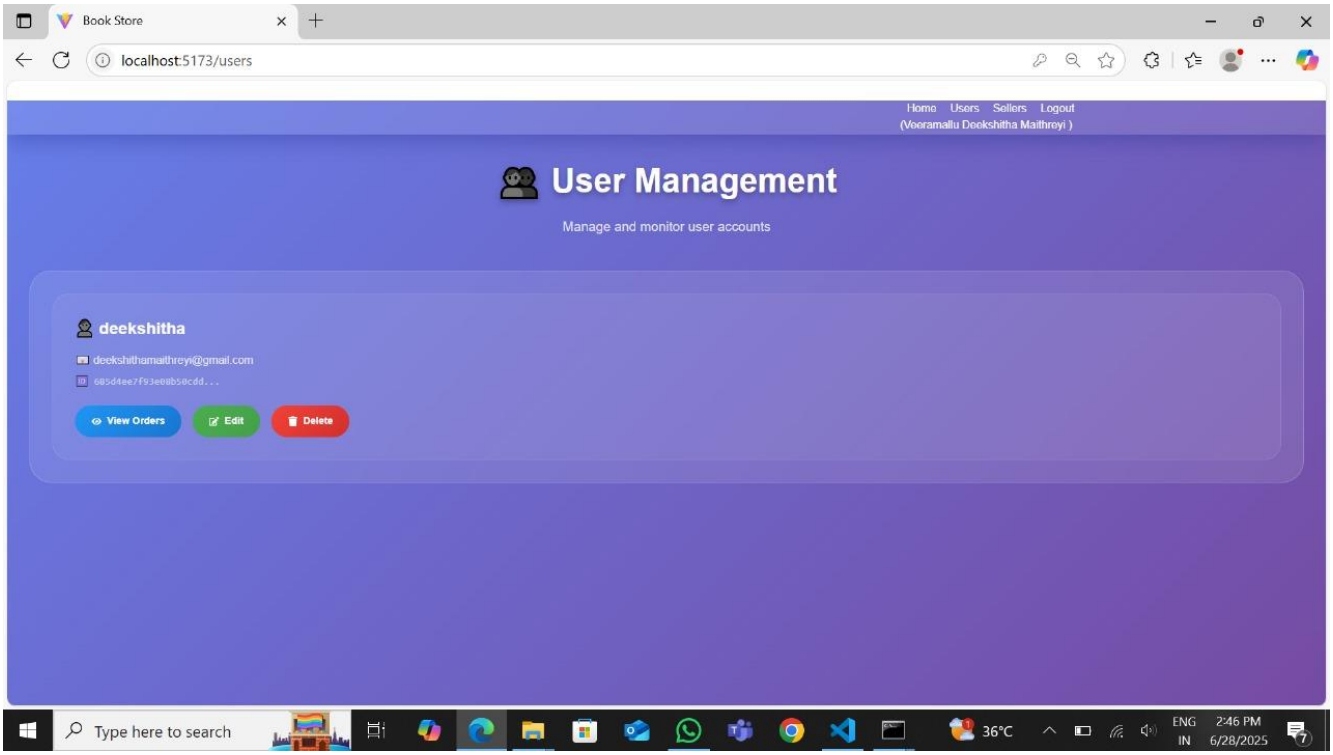
Vender Management:



Admin Dashboard:



User Management:



For any further doubts or help, please consider the drive,

https://drive.google.com/drive/folders/13GfDZKAYzkzK9pwMStaaVUEgY3kNY9CO?usp=sharing

The demo of the app is available at:
https://drive.google.com/file/d/1LXRrIHydpSGso5S6fwyv3yYW7xmyQsm_/view?usp=sharing

TESTING

Manual testing done for all flows: register, login, cart, order, admin panel

Testing Scope

Features and Functionalities to be Tested:

- User registration and login
- Product browsing and search
- Cart management
- Order placement and order history
- Admin product/user/order management

User Stories/Requirements to be Tested:

- USN-1: User registration
- USN-2: Registration confirmation
- USN-3: User login
- USN-4: Product browsing/search
- USN-5: Cart and order placement
- USN-6: Admin management

KNOWN ISSUES

- book images might not appear if uploads folder is empty or missing
- Basic error messages need better UX design
- No support for password reset yet

FUTURE ENHANCEMENTS

- Integrate Stripe or Razorpay for payment gateway
- Allow book reviews and star ratings
- Enable email notifications on order status
- Multi-language support
- Add accessibility features and dark mode