

BOOK STORE

IIHT

Time To Complete: 2 hr

CONTENTS

| | | |
|-----|---|----|
| 1 | Project Abstract | 2 |
| 2 | Problem Statement | 2 |
| 3 | Proposed Book Store Application Wireframe | 3 |
| 3.1 | Welcome Page | 3 |
| 3.2 | Admin | 5 |
| 3.3 | User | 7 |
| 3 | Business-Requirement | 10 |
| 4 | Validations | 14 |
| 5 | Constraints | 14 |
| 6 | Mandatory Assessment Guidelines | 15 |

1 PROJECT ABSTRACT

In the rapidly advancing digital age, the demand for convenient and accessible platforms for purchasing books has significantly increased. With the vision of creating an innovative and user-friendly Book Store Management System, the CEO of a budding e-commerce startup, Mr. X, assigns a team of developers to develop a Book Store application using React.

This application aims to provide a user-friendly and efficient platform for both book buyers and store administrators, enhancing the overall book shopping and management experience.

Your task is to develop a comprehensive digital solution that enables users to effortlessly browse through a variety of books, add their desired books to a cart, and place orders with ease. Additionally, the platform will include an admin interface where administrators can manage users, books, and orders efficiently.

2 PROBLEM STATEMENT

The "**Book Store App**" is a Single Page Application (SPA) designed to enhance the online book shopping experience. This system allows users to browse and search for books, add books to their cart, and place orders seamlessly. It also provides an admin interface for managing users, books, and orders.

3 PROPOSED BOOK STORE WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 WELCOME PAGE



Welcome to the Online Bookstore

Already added three books with their details



Welcome to the Online Bookstore

- The Great Gatsby by F. Scott Fitzgerald - 1099.00
- To Kill a Mockingbird by Harper Lee - 899.00
- 1984 by George Orwell - 999.00

*** Login Page***

🔍 📄 ⬅ ➡ ↻ localhost:8081/login

[Online Bookstore](#)
[HomeLogin](#)

Login

Username:

Password:

Login

🔍 📄 ⬅ ➡ ↻ localhost:8081/login

[Online Bookstore](#)
[HomeLogin](#)

Login

Invalid username or password

Username:

Password:

Login

🔍 📄 ⬅ ➡ ↻ localhost:8081/login

[Online Bookstore](#)
[HomeLogin](#)

Login

Username:

Password:

Login

3.2 ADMIN PAGE

*** Admin Home Page***

 localhost:8081/

[Online Bookstore](#)
[HomeAdmin PanelOrders](#)

Welcome to the Online Bookstore

- The Great Gatsby by F. Scott Fitzgerald - 1099.00
- To Kill a Mockingbird by Harper Lee - 899.00
- 1984 by George Orwell - 999.00

*** Admin Panel Page***

 localhost:8081/admin

[Online Bookstore](#)
[HomeAdmin PanelOrders](#)

Admin Panel

Users

- user1 (user)
- user2 (user)

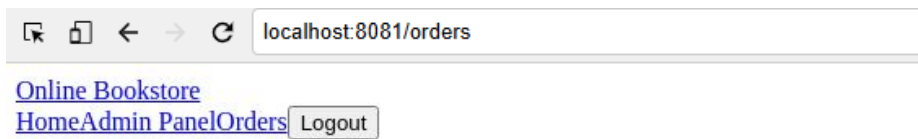
Books

Add a Book

Title:
Author:
Price:
Description:

- The Great Gatsby by F. Scott Fitzgerald - 1099.00
- To Kill a Mockingbird by Harper Lee - 899.00
- 1984 by George Orwell - 999.00

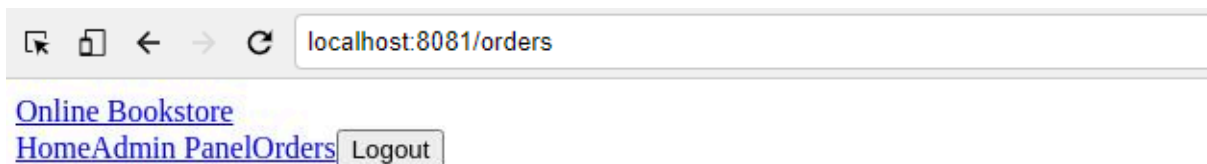
*** Admin Order Page***



Orders

Order List

- Order ID: 1User ID: 2Total: 1099.00Status: confirmed
- Order ID: 2User ID: 3Total: 1898.00Status: confirmed
- Order ID: 3User ID: 3Total: 1099.00Status: confirmed



Orders

Order List

- Order ID: 1User ID: 2Total: 1099.00Status: confirmed
- Order ID: 2User ID: 3Total: 1898.00Status: confirmed
- Order ID: 3User ID: 3Total: 1099.00Status: confirmed

Order Detail

Order ID: 3

User ID: 3


Total: 1099.00

Status: confirmed

Books:

- The Great Gatsby by F. Scott Fitzgerald - 1099.00
-

3.3 USER PAGE



A screenshot of a web browser showing the login page. The address bar displays 'localhost:8081/login'. Below the address bar are two links: 'Online Bookstore' and 'HomeLogin'. The main heading is 'Login'. There are two input fields: 'Username:' with the value 'user1' and 'Password:' with masked characters '*****'. A 'Login' button is located below the password field.

localhost:8081/login

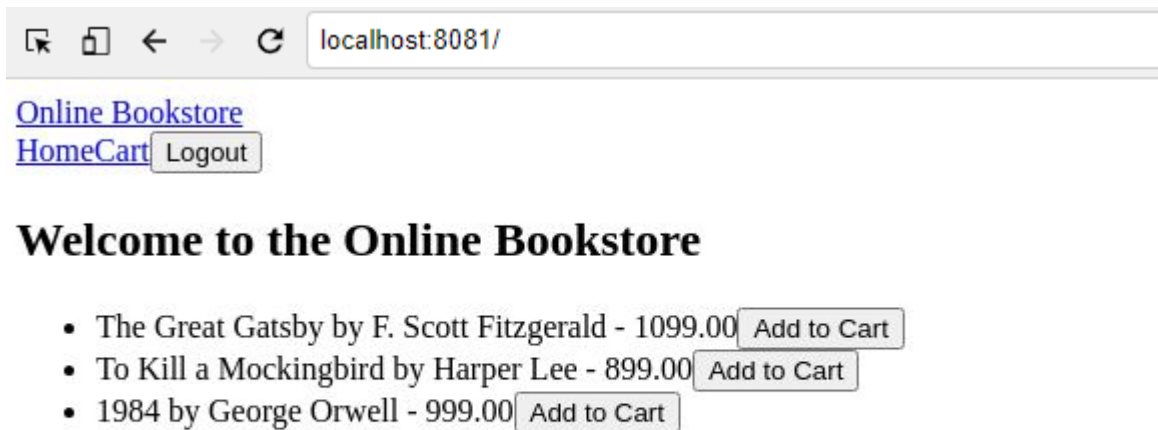
[Online Bookstore](#)
[HomeLogin](#)

Login

Username:

Password:

*** User Home Page***



A screenshot of a web browser showing the user home page. The address bar displays 'localhost:8081/'. Below the address bar are two links: 'Online Bookstore' and 'HomeCart', followed by a 'Logout' button. The main heading is 'Welcome to the Online Bookstore'. Below the heading is a list of three books, each with an 'Add to Cart' button.

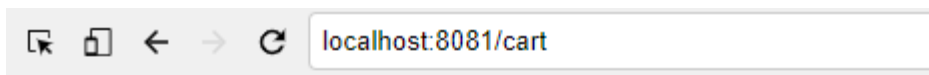
localhost:8081/

[Online Bookstore](#)
[HomeCart](#)

Welcome to the Online Bookstore

- The Great Gatsby by F. Scott Fitzgerald - 1099.00
- To Kill a Mockingbird by Harper Lee - 899.00
- 1984 by George Orwell - 999.00

*** User Cart Page***

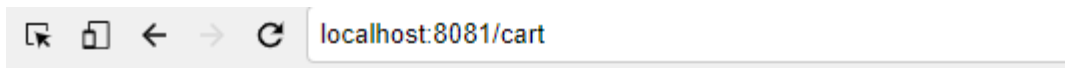


[Online Bookstore](#)

[HomeCart](#) [Logout](#)

Your Cart

*** After adding Books to Cart***



[Online Bookstore](#)

[HomeCart](#) [Logout](#)


Your Cart

- To Kill a Mockingbird by Harper Lee - 899.00 x 2 [Remove](#)
- 1984 by George Orwell - 999.00 x 1 [Remove](#)

Total: 2797.00

[Place Order](#)

*** After removing a Book from Cart***



[Online Bookstore](#)
[HomeCart](#)

Your Cart

- 1984 by George Orwell - 999.00 x 1

Total: 999.00

4 BUSINESS-REQUIREMENT:

As an application developer, develop the Book Store (Single Page App) with below guidelines:

| User Story # | User Story Name | User Story |
|--------------|-----------------|--|
| US_01 | Welcome Page | <p>Acceptance Criteria - App Component</p> <ol style="list-style-type: none">1. Display a navigation bar at the top of the application (<code><Navbar /></code>).2. Route users to appropriate pages:<ul style="list-style-type: none">• Default route <code>/</code> shows the <code>HomePage</code>.• Route <code>/login</code> shows the <code>LoginPage</code>.• Route <code>/admin</code> accessible only to admin users; redirects to <code>/login</code> if not authorized.• Route <code>/cart</code> accessible only to authenticated users with role <code>user</code>; otherwise redirects to <code>/login</code>.• Route <code>/orders</code> accessible only to admin users; otherwise redirects to <code>/login</code>.3. Maintain user authentication state:<ul style="list-style-type: none">• Load from <code>localStorage</code> on mount.• Store updates to <code>localStorage</code> on login/logout.4. Use context (<code>AuthContext</code>) to provide global authentication access. <p>State & Context Overview</p> <p>State Variables (via <code>useState</code>):</p> <ul style="list-style-type: none">- <code>auth</code>: null<ul style="list-style-type: none">• Represents the current authenticated user.• Shape: <code>{ username: string, role: 'user' 'admin' }</code>- Context: <code>AuthContext</code>- Provides:<ul style="list-style-type: none">• <code>auth</code>: Current user info• <code>login(user)</code>: Log in a user and store info |

- `logout()`: Clear authentication info

Functions & Responsibilities

1. `useEffect`: On component mount:

- Load `auth` object from `localStorage` (if present).
- Set it in component state.

2. `login(user)`:

- Saves user data to `auth` state.
- Persists to `localStorage`.

3. `logout()`:

- Clears user data from state and `localStorage`.

4. Route Handling: Conditionally render routes based on `auth` role:

- Admins: `/admin`, `/orders`
- Users: `/cart`
- Redirect unauthorized access to `/login`.

HTML Structure

App Component Layout:

- `<AuthContext.Provider>`: Provides global authentication state and functions.
- `<Router>`: Wraps the routing logic.
- `<div>`: Wraps all main content.
- `<Navbar />`: Displays navigation links across the top.
- `<Switch>`: Manages route-based rendering.
 - `<Route path="/" exact>`: Renders `<HomePage />`.
 - `<Route path="/login">`: Renders `<LoginPage />`.
 - `<Route path="/admin">`: Renders `<AdminPage />` if user is an admin; otherwise redirects to `/login`.
 - `<Route path="/cart">`: Renders `<CartPage />` if user is a standard user; otherwise redirects to `/login`.
 - `<Route path="/orders">`: Renders `<OrderPage />` if user is an admin; otherwise redirects to `/login`.
 - `<Redirect to="/" />`: Handles all undefined

routes by redirecting to the home page.

Dynamic Behavior

1. On App Load:

- Reads and sets authentication state from `localStorage`.

2. Login:

- Calls `login(user)` from LoginPage.
- Updates context and persists to `localStorage`.

3. Logout:

- Calls `logout()`, clearing both context and local storage.
- Can trigger UI updates or redirects based on context.

4. Route Access Control:

- Routes like `/admin`, `/cart`, and `/orders` are protected:
 - Unauthenticated or unauthorized users are redirected to `/login`.

Admin Page

Acceptance Criteria - AdminPage

1. Display an "Admin Panel" heading.
2. Fetch and display a list of all users who are not admins.
3. Fetch and display a list of all books.
4. Allow the admin to:
 - Add a new book using the `BookForm`.
 - Select a book for viewing or editing.
 - Delete a book from the system.
5. All changes should reflect in the UI and update the backend accordingly.

State & Props Overview

State Variables:

- `users` (array): Stores non-admin users fetched from the backend.
- `books` (array): Stores all books fetched from the backend.

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> ● selectedBook (object null): Stores the currently selected book for potential editing. <p>Context:</p> <ul style="list-style-type: none"> ● auth: Received from AuthContext to verify admin access. <p>Functions & Responsibilities</p> <ol style="list-style-type: none"> 1. fetchUsers(): <ul style="list-style-type: none"> ● Axios GET request to http://localhost:4000/users. ● Filters out admin users and updates the users state. 2. fetchBooks(): <ul style="list-style-type: none"> ● Axios GET request to http://localhost:4000/books. ● Updates the books state. 3. handleBookSelect(book): <ul style="list-style-type: none"> ● Sets the selected book in selectedBook state (used for editing or detail view). 4. handleBookCreate(book): <ul style="list-style-type: none"> ● Sends POST request to create a new book to http://localhost:4000/books. ● Updates books state with the new entry. 5. handleBookDelete(bookId): <ul style="list-style-type: none"> ● Sends DELETE request to remove the book to http://localhost:4000/books/\${bookId} ● Updates books state to exclude the deleted book. <p>HTML Structure</p> <p>AdminPage Component Layout:</p> <ol style="list-style-type: none"> 1. <div className="admin-page">: Wraps entire admin content. 2. <h2>: "Admin Panel" 3. <section> for users list: <ul style="list-style-type: none"> ● <h3>: "Users" ● : Loops through users array. <ul style="list-style-type: none"> ➔ : Displays username and role of each non-admin user. 3. <section> for book management: <ul style="list-style-type: none"> ● <h3>: "Books" ● <BookForm />: Used to add a new book. |
|--|--|--|

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> • <code><BookList /></code>: Displays all books with options to select or delete. <p>Dynamic Behavior</p> <ol style="list-style-type: none"> 1. Initial Load: <ul style="list-style-type: none"> • Fetches both users and books using <code>useEffect</code> and updates their respective states. 2. Adding a Book: <ul style="list-style-type: none"> • Submitting the form in <code>BookForm</code> triggers <code>handleBookCreate</code>. • New book is added to backend and reflected in the <code>books</code> state. 3. Deleting a Book: <ul style="list-style-type: none"> • Clicking delete on a book calls <code>handleBookDelete</code>. • Removes the book from both backend and local state. 4. Selecting a Book: <ul style="list-style-type: none"> • Calls <code>handleBookSelect</code> which sets the selected book in state for potential editing. <p>Cart Page</p> <p>Acceptance Criteria – CartPage</p> <ol style="list-style-type: none"> 1. Display a heading: “Your Cart”. 2. On component load: <ul style="list-style-type: none"> • Fetch the logged-in user's cart using their <code>userId</code>. • Show a loading message until the cart is available. 3. Display each book in the cart: <ul style="list-style-type: none"> • Title, author, price, quantity. • Include a “Remove” button to delete a book from the cart. 4. If the cart has items: <ul style="list-style-type: none"> • Show total cost. • Show a “Place Order” button. 5. On placing an order: <ul style="list-style-type: none"> • Send order details to the backend. • Clear the cart both in backend and UI. • Display success alert. |
|--|--|--|

| | | |
|--|--|--|
| | | <h2>State & Props Overview</h2> <p>State Variables:</p> <ul style="list-style-type: none"> <code>cart</code> (object null): Contains cart data for the user. <ul style="list-style-type: none"> → Shape: <code>{ id, userId, books: [{ id, title, author, price, quantity }] }</code> <p>Context:</p> <ul style="list-style-type: none"> <code>auth</code>: From <code>AuthContext</code>, contains the currently logged-in user info (<code>auth.id</code> used for cart fetch). <h2>Functions & Responsibilities</h2> <ol style="list-style-type: none"> <code>fetchCart()</code>: <ul style="list-style-type: none"> Axios GET request: <code>http://localhost:4000/cart?userId=\${auth.id}</code> Loads and sets cart state on component mount. <code>handleRemoveBook(bookId)</code>: <ul style="list-style-type: none"> Filters the book out of the current cart. Sends PUT request to update cart on the backend to <code>http://localhost:4000/cart/\${cart.id}</code>. Updates <code>cart</code> state locally. <code>handlePlaceOrder()</code>: <ul style="list-style-type: none"> Constructs an order object using cart data. Axios POST to <code>http://localhost:4000/orders</code> to create the order. Axios DELETE to remove the cart from backend. Resets cart state to empty. Displays an alert on success. <h2>HTML Structure</h2> <p>CartPage Component Layout:</p> <ol style="list-style-type: none"> <code><div></code>: Wraps the entire cart page. <code><h3></code>: "Your Cart" <code></code>: Renders each item in <code>cart.books</code>: <ul style="list-style-type: none"> <code></code> for each book: <ul style="list-style-type: none"> → Title, author, price, and quantity → "Remove" button to delete the book <code><h4></code>: Shows total price if there are items in cart. <code><button></code>: "Place Order" button shown if cart is not empty. |
|--|--|--|

| | | |
|--|--|--|
| | | <p>Dynamic Behavior</p> <ol style="list-style-type: none"> On Load: <ul style="list-style-type: none"> If user is not authenticated: display "Loading..." If authenticated: fetch cart and display contents. Remove Book: <ul style="list-style-type: none"> Clicking "Remove" deletes the book via PUT request and updates local cart state. Place Order: <ul style="list-style-type: none"> Clicking "Place Order" sends cart data as an order to backend. Deletes the cart from backend. Updates UI to show empty cart and alerts the user. <p>Home Page</p> <p>Acceptance Criteria – HomePage</p> <ol style="list-style-type: none"> Display a welcome heading: "Welcome to the Online Bookstore". On page load: <ul style="list-style-type: none"> Fetch and display a list of available books from the backend. For authenticated users with the role user: <ul style="list-style-type: none"> Allow them to add books to their cart. If the user has no existing cart, create a new one. If the book already exists in the cart, increment its quantity. Otherwise, add the new book with a quantity of 1. Show an alert upon successful addition to cart. For non-authenticated or non-user roles, disable "Add to Cart". <p>State & Props Overview</p> <p>State Variables:</p> <ul style="list-style-type: none"> books (array): List of all books fetched from the backend. <p>Context:</p> <ul style="list-style-type: none"> auth: From AuthContext, determines whether the user is logged in and their role. |
|--|--|--|

| | | |
|--|--|---|
| | | <h2>Functions & Responsibilities</h2> <ol style="list-style-type: none"> 1. fetchBooks(): <ul style="list-style-type: none"> • Axios GET request: http://localhost:4000/books • Fetches available books and sets the books state. 2. addToCart(book): <ul style="list-style-type: none"> • If the user is not logged in, exits early. • Checks if the user has an existing cart: <ul style="list-style-type: none"> → If not, creates a new cart with the selected book. → If yes, updates the cart: <ul style="list-style-type: none"> ➢ Increases quantity if the book already exists. ➢ Adds book if it's not present. • Sends POST or PUT request as needed to http://localhost:4000/cart or http://localhost:4000/cart/\${userCart.id} • Alerts the user upon successful action. <h2>HTML Structure</h2> <p>HomePage Component Layout:</p> <ol style="list-style-type: none"> 1. <div>: Wraps entire page content. 2. <h2>: "Welcome to the Online Bookstore" 3. <BookList />: Displays the list of books. <ul style="list-style-type: none"> • Props: <ul style="list-style-type: none"> → books: Array of books from state. → addToCart: Function passed only if the user is authenticated and has the user role. <h2>Dynamic Behavior</h2> <ol style="list-style-type: none"> 1. On Load: <ul style="list-style-type: none"> • Book list is fetched from the backend and rendered on the screen. 2. Add to Cart: <ul style="list-style-type: none"> • Only available to logged-in users with the user role. • Adds book to cart using backend requests: <ul style="list-style-type: none"> → Creates a new cart or updates existing one. • Book quantities are managed dynamically. • Alerts the user on success. <h2>Login Page</h2> <h2>Acceptance Criteria – LoginPage</h2> |
|--|--|---|

| | | |
|--|--|--|
| | | <ol style="list-style-type: none"> 1. Display a heading: "Login". 2. Show a login form with: <ul style="list-style-type: none"> • Input fields for username and password. • Submit button labeled "Login". 3. Validate form: <ul style="list-style-type: none"> • Both fields are required. 4. On form submission: <ul style="list-style-type: none"> • Make a request to fetch a user matching the provided credentials. • If a valid user is found: <ul style="list-style-type: none"> → Store authentication context using <code>login</code>. → Redirect: <ul style="list-style-type: none"> ➤ Admin users → <code>/admin</code> ➤ Regular users → <code>/</code> • If invalid, display an error message. 5. If already logged in, redirect to home page automatically. <p>State & Props Overview</p> <p>State Variables:</p> <ul style="list-style-type: none"> • <code>credentials</code> (object): Stores the user's input. <ul style="list-style-type: none"> → Shape: <code>{ username: string, password: string }</code> • <code>error</code> (string): Stores any error message for invalid login or request failure. <p>Context:</p> <ul style="list-style-type: none"> • <code>auth</code>: Current authenticated user. • <code>login(user)</code>: Function from <code>AuthContext</code> to store login state globally. <p>Other Hooks:</p> <ul style="list-style-type: none"> • <code>history</code>: From <code>useHistory</code> for programmatic navigation. <p>Functions & Responsibilities</p> <ol style="list-style-type: none"> 1. <code>useEffect</code>: <ul style="list-style-type: none"> • Redirects to home (<code>/</code>) if a user is already authenticated. 2. <code>handleChange(e)</code>: <ul style="list-style-type: none"> • Updates <code>credentials</code> state as the user types into the form inputs. |
|--|--|--|

| | | |
|--|--|---|
| | | <p>3. <code>handleSubmit(e)</code>:</p> <ul style="list-style-type: none"> • Prevents default form behavior. • Makes a GET request to <code>http://localhost:4000/users</code> with query parameters. • If user is found: <ul style="list-style-type: none"> → Calls <code>login(user)</code> → Redirects based on role. • If not found or request fails: <ul style="list-style-type: none"> → Sets error message accordingly. <p>HTML Structure</p> <p>LoginPage Component Layout:</p> <ol style="list-style-type: none"> 1. <code><div className="login-page"></code>: Container for the login form. 2. <code><h2></code>: "Login" 3. <code><p className="error"></code>: Conditionally rendered if <code>error</code> exists. 4. <code><form></code>: Handles form submission. <ul style="list-style-type: none"> • <code><div></code> with <code><label></code> and <code><input></code> for "Username" • <code><div></code> with <code><label></code> and <code><input></code> for "Password" • <code><button></code>: Submit button labeled "Login" <p>Dynamic Behavior</p> <ol style="list-style-type: none"> 1. Initial Redirect: <ul style="list-style-type: none"> • If user is already logged in (<code>auth</code> exists), redirect to <code>/</code>. 2. Form Validation: <ul style="list-style-type: none"> • Submit button is disabled by default by the <code>required</code> attribute. 3. Login Process: <ul style="list-style-type: none"> • On successful match: <ul style="list-style-type: none"> → Updates global auth context. → Redirects based on role. • On failure: <ul style="list-style-type: none"> → Displays relevant error message. <p>Order Page</p> <p>Acceptance Criteria – OrderPage</p> |
|--|--|---|

| | | |
|--|--|--|
| | | <ol style="list-style-type: none"> 1. Display a heading: "Orders". 2. On component load: <ul style="list-style-type: none"> • If the user is an admin: <ul style="list-style-type: none"> → Fetch and display all orders from the backend. • If the user is a regular user: <ul style="list-style-type: none"> → Fetch and display only the orders placed by that user. 3. Display a list of fetched orders. <ul style="list-style-type: none"> • Allow the admin/user to click an order to view its details. 4. Show detailed information of the selected order using a separate component. <h3>State & Props Overview</h3> <p>State Variables:</p> <ul style="list-style-type: none"> • <code>orders</code> (array): List of orders fetched from the backend. • <code>selectedOrder</code> (object null): Stores the order selected for detailed view. <p>Context:</p> <ul style="list-style-type: none"> • <code>auth</code>: From <code>AuthContext</code>, determines whether the user is an admin or a regular user. <h3>Functions & Responsibilities</h3> <ol style="list-style-type: none"> 1. <code>fetchOrders()</code>: <ul style="list-style-type: none"> • If <code>auth.role === 'admin'</code>, sends GET request to <code>http://localhost:4000/orders</code>. • If <code>auth.role === 'user'</code>, sends GET request to <code>http://localhost:4000/orders?userId=\${auth.id}</code>. • Updates the <code>orders</code> state with the response. 2. <code>handleOrderSelect(order)</code>: <ul style="list-style-type: none"> • Sets the selected order for detailed viewing. <h3>HTML Structure</h3> <p>OrderPage Component Layout:</p> <ol style="list-style-type: none"> 1. <code><div></code>: Wraps the entire order page content. 2. <code><h3></code>: "Orders" 3. <code><OrderList /></code>: <ul style="list-style-type: none"> • Receives <code>orders</code> as prop to display the list. • Receives <code>selectOrder</code> callback to handle order selection. 4. <code><OrderDetail /></code>: |
|--|--|--|

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> Conditionally rendered. Displays detailed view of <code>selectedOrder</code>. <p>Dynamic Behavior</p> <ol style="list-style-type: none"> On Load: <ul style="list-style-type: none"> Orders are fetched based on the authenticated user's role. Order Selection: <ul style="list-style-type: none"> Clicking on an order in <code>OrderList</code> sets <code>selectedOrder</code>. <code>OrderDetail</code> is rendered to display order specifics. <p>=====</p> <p>AdminOrderList Component</p> <p>Acceptance Criteria - AdminOrderList</p> <ol style="list-style-type: none"> On component mount, fetch all orders from the backend. Display a list of all orders with: <ul style="list-style-type: none"> Order ID User ID Total amount Nested list of books with title, author, and price <p>State & Props Overview</p> <p>State Variables:</p> <ul style="list-style-type: none"> <code>orders</code> (array): List of all orders fetched from the backend. <p>Functions & Responsibilities</p> <ol style="list-style-type: none"> <code>fetchOrders()</code>: <ul style="list-style-type: none"> Sends a GET request to <code>http://localhost:4000/orders</code>. Updates the <code>orders</code> state with the received data. <p>HTML Structure</p> <p>AdminOrderList Component Layout:</p> <ol style="list-style-type: none"> <code><div></code>: Wraps the entire component content. <code></code>: Renders the list of orders. <ul style="list-style-type: none"> <code></code>: For each order: <ul style="list-style-type: none"> → Displays: <ul style="list-style-type: none"> ➤ <code></code> label for Order ID, User ID, and Total ➤ Nested <code></code> for books: |
|--|--|--|

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> Each <code></code> shows: book title, author, and price <p>Dynamic Behavior</p> <p>1. Initial Load:</p> <ul style="list-style-type: none"> Orders are fetched using <code>useEffect</code>. UI updates automatically as data is loaded. <p>AdminUserList Component</p> <p>Acceptance Criteria - AdminUserList</p> <p>1. On component mount, fetch all users from the backend.</p> <p>2. Display a list of users with:</p> <ul style="list-style-type: none"> Username Role <p>State & Props Overview</p> <p>State Variables:</p> <ul style="list-style-type: none"> <code>users</code> (array): List of users fetched from the backend. <p>Functions & Responsibilities</p> <p>1. <code>fetchUsers()</code>:</p> <ul style="list-style-type: none"> Sends a GET request to <code>http://localhost:4000/users</code>. Updates the <code>users</code> state with the received data. <p>HTML Structure</p> <p>AdminUserList Component Layout:</p> <p>1. <code><div></code>: Wraps the user list.</p> <p>2. <code></code>: Renders the list of users.</p> <ul style="list-style-type: none"> <code></code> for each user: <ul style="list-style-type: none"> → Displays: <ul style="list-style-type: none"> ➤ <code></code> label for Username and Role <p>Dynamic Behavior</p> <p>1. Initial Load:</p> <ul style="list-style-type: none"> User data is fetched on mount. List updates as soon as the data is loaded from the backend. |
|--|--|--|

| | | |
|--|--|---|
| | | <h2>BookForm Component</h2> <h3>Acceptance Criteria - BookForm</h3> <ol style="list-style-type: none"> 1. Display a heading: <ul style="list-style-type: none"> • “Add a Book” if creating a new book. • “Edit Book” if updating an existing one. 2. Render a form with the following fields: <ul style="list-style-type: none"> • Title (text) • Author (text) • Price (number) • Description (textarea) 3. The submit button should: <ul style="list-style-type: none"> • Display “Add Book” or “Update Book” based on context. • Be disabled unless all fields are filled. • Call <code>addBook</code> for new entries. • Call <code>updateBook</code> when editing. 4. After submission: <ul style="list-style-type: none"> • Clear the form fields. <h3>State & Props Overview</h3> <p>Props Received:</p> <ul style="list-style-type: none"> • <code>addBook</code>: Function to handle creation of a new book. • <code>editBook</code>: Object representing the book being edited. • <code>updateBook</code>: Function to handle updating a book. <p>State Variables:</p> <ul style="list-style-type: none"> • <code>book</code>: Object with fields <code>title</code>, <code>author</code>, <code>price</code>, and <code>description</code>. <h3>Functions & Responsibilities</h3> <ol style="list-style-type: none"> 1. <code>useEffect</code>: <ul style="list-style-type: none"> • If <code>editBook</code> is passed in, prefill the form fields. • Otherwise, reset to default empty form. 2. <code>handleChange(e)</code>: <ul style="list-style-type: none"> • Updates the <code>book</code> state with input changes. 3. <code>handleSubmit(e)</code>: |
|--|--|---|

| | | |
|--|--|---|
| | | <ul style="list-style-type: none"> • Prevents default form submission. • Calls <code>addBook</code> or <code>updateBook</code> depending on mode. • Resets the form. <p>HTML Structure</p> <p>BookForm Component Layout:</p> <ol style="list-style-type: none"> 1. <code><div className="book-form"></code>: Wraps the entire form. 2. <code><h3></code>: Displays either “Add a Book” or “Edit Book”. 3. <code><form></code>: Handles form submission. <ul style="list-style-type: none"> • Input fields: <ul style="list-style-type: none"> → Title → Author → Price → Description • <code><button></code>: “Add Book” or “Update Book” depending on context. <p>Dynamic Behavior</p> <ol style="list-style-type: none"> 1. Initial State: <ul style="list-style-type: none"> • Empty form unless <code>editBook</code> is provided. 2. Form Submission: <ul style="list-style-type: none"> • Calls appropriate function and resets form on success. 3. Form Updates: <ul style="list-style-type: none"> • Edits are reflected in real time in the local form state. <p>BookList Component</p> <p>Acceptance Criteria - BookList</p> <ol style="list-style-type: none"> 1. Render a list of books. 2. For each book, display: <ul style="list-style-type: none"> • Title, author, and price. 3. Show buttons conditionally: <ul style="list-style-type: none"> • “Add to Cart” if <code>addToCart</code> function is passed. • “View Details” if <code>selectBook</code> function is passed. • “Delete” if <code>deleteBook</code> function is passed. <p>Props Overview</p> <p>Props Received:</p> |
|--|--|---|

| | | |
|--|--|---|
| | | <ul style="list-style-type: none"> • books: Array of book objects to display. • selectBook (optional): Function to handle selection/viewing. • deleteBook (optional): Function to handle deletion. • addToCart (optional): Function to handle adding to cart. <p>Functions & Responsibilities</p> <ol style="list-style-type: none"> 1. Direct event handlers within JSX: <ul style="list-style-type: none"> • Call appropriate prop function on button clicks. <p>HTML Structure</p> <p>BookList Component Layout:</p> <ol style="list-style-type: none"> 1. <div className="book-list">: Wraps the entire book list. 2. : Renders the book list. <ul style="list-style-type: none"> • : For each book: <ul style="list-style-type: none"> → Displays: <ul style="list-style-type: none"> ➢ Title ➢ Author ➢ Price → Conditionally renders buttons: <ul style="list-style-type: none"> ➢ "Add to Cart" ➢ "View Details" ➢ "Delete" <p>Dynamic Behavior</p> <ol style="list-style-type: none"> 1. Conditional UI: <ul style="list-style-type: none"> • Buttons only appear based on the props provided. 2. Event Handling: <ul style="list-style-type: none"> • Interactions are passed up to the parent via props. <p>Navbar Component</p> <p>Acceptance Criteria - Navbar</p> <ol style="list-style-type: none"> 1. Always display the brand link: "Online Bookstore" linking to /. 2. Display navigation links based on user authentication and role: <ul style="list-style-type: none"> • Everyone sees "Home" • Logged-in users with role user see: <ul style="list-style-type: none"> → "Cart" • Logged-in users with role admin see: <ul style="list-style-type: none"> → "Admin Panel" → "Orders" |
|--|--|---|

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> • If no user is logged in: → Show “Login” • If a user is logged in: → Show a “Logout” button <p>3. On clicking “Logout”:</p> <ul style="list-style-type: none"> • Clear authentication state. • Redirect to the login page. <h3>State & Context Overview</h3> <p>Context Used:</p> <ul style="list-style-type: none"> • <code>auth</code>: Authentication object from <code>AuthContext</code> • <code>logout</code>: Function to clear authentication and local storage <p>Other Hooks:</p> <ul style="list-style-type: none"> • <code>history</code>: From <code>useHistory</code> to navigate after logout <h3>Functions & Responsibilities</h3> <p>1. <code>handleLogout()</code>:</p> <ul style="list-style-type: none"> • Calls <code>logout()</code> to clear authentication context. • Redirects to <code>/login</code>. <h3>HTML Structure</h3> <p>Navbar Component Layout:</p> <p>1. <code><nav className="navbar"></code>: Wraps the entire navbar.</p> <ul style="list-style-type: none"> • <code><div className="navbar-brand"></code>: Contains: → <code><Link></code> to <code>/</code> labeled “Online Bookstore” • <code><div className="navbar-links"></code>: Contains: → <code><Link></code> to <code>/</code> labeled “Home” → Conditional links: <ul style="list-style-type: none"> ➢ If user role is <code>user</code>: <code><Link></code> to <code>/cart</code> ➢ If user role is <code>admin</code>: <code><Link></code> to <code>/admin</code>, <code><Link></code> to <code>/orders</code> → If not authenticated: <code><Link></code> to <code>/login</code> → If authenticated: <code><button></code> labeled “Logout” <h3>Dynamic Behavior</h3> <p>1. Role-based Navigation:</p> <ul style="list-style-type: none"> • Links rendered based on <code>auth</code> and user role. <p>2. Authentication Switching:</p> <ul style="list-style-type: none"> • Toggling login/logout dynamically changes the visible links. <p>3. Logout Action:</p> <ul style="list-style-type: none"> • Clears authentication context. • Navigates to login page. |
|--|--|--|

| | | |
|--|--|--|
| | | <h2>OrderDetail Component</h2> <h3>Acceptance Criteria - OrderDetail</h3> <ol style="list-style-type: none"> 1. Display a heading: "Order Detail". 2. Show key information for the selected order: <ul style="list-style-type: none"> • Order ID • User ID • Total amount (formatted to 2 decimal places) • Status 3. Display a list of books in the order: <ul style="list-style-type: none"> • For each book, show title, author, and price (formatted). <h3>Props Overview</h3> <p>Props Received:</p> <ul style="list-style-type: none"> • <code>order</code> (object): The selected order to display. <ul style="list-style-type: none"> → Shape includes: <ul style="list-style-type: none"> ➤ <code>id, userId, total, status, books[]</code> <h3>Functions & Responsibilities</h3> <ol style="list-style-type: none"> 1. Render Static Info: <ul style="list-style-type: none"> • Displays values directly from the <code>order</code> prop: <ul style="list-style-type: none"> → <code>order.id, order.userId, order.total, order.status</code> 2. Render Book List: <ul style="list-style-type: none"> • Maps through <code>order.books</code> to generate <code></code> items displaying: <ul style="list-style-type: none"> → Title, author, and price for each book <h3>HTML Structure</h3> <p>OrderDetail Component Layout:</p> <ol style="list-style-type: none"> 1. <code><div className="order-detail"></code>: Wraps the entire detail view. 2. <code><h3></code>: "Order Detail" 3. <code><p></code> elements: <ul style="list-style-type: none"> • Display <code>Order ID, User ID, Total, Status</code> 4. <code><h4></code>: "Books:" |
|--|--|--|

| | | |
|--|--|---|
| | | <p>5. <code></code>: Renders list of books in the order.</p> <ul style="list-style-type: none"> • <code></code> for each book: <ul style="list-style-type: none"> → Displays: <ul style="list-style-type: none"> ➤ Title ➤ Author ➤ Price <p>Dynamic Behavior</p> <ol style="list-style-type: none"> 1. Renders detailed view dynamically when <code>order</code> prop is updated. 2. Book prices are formatted to 2 decimal places. <p>OrderList Component</p> <p>Acceptance Criteria - OrderList</p> <ol style="list-style-type: none"> 1. Display a heading: “Order List”. 2. Render a list of orders. 3. For each order, display: <ul style="list-style-type: none"> • Order ID • User ID • Total amount (formatted) • Status 4. When an order is clicked: <ul style="list-style-type: none"> • Call <code>selectOrder</code> with the clicked order object. <p>Props Overview</p> <p>Props Received:</p> <ul style="list-style-type: none"> • <code>orders</code> (array): List of order objects to render. • <code>selectOrder</code> (function): Callback to handle order selection. <p>Functions & Responsibilities</p> <ol style="list-style-type: none"> 1. Render Orders: <ul style="list-style-type: none"> • Iterates over <code>orders</code> array • Displays key details: ID, User ID, Total, and Status 2. <code>handleOrderSelect</code> (inline): <ul style="list-style-type: none"> • Each <code></code> includes an <code>onClick</code> event • Calls <code>selectOrder(order)</code> with the clicked order object |
|--|--|---|

| | | |
|--|--|---|
| | | <ul style="list-style-type: none"> Used by parent (e.g., <code>OrderPage</code>) to set selected order for detail view <p>HTML Structure</p> <p>OrderList Component Layout:</p> <ol style="list-style-type: none"> <code><div className="order-list"></code>: Wraps the component. <code><h3></code>: "Order List" <code></code>: Renders list of orders. <ul style="list-style-type: none"> <code></code> for each order: <ul style="list-style-type: none"> Displays: <ul style="list-style-type: none"> Order ID User ID Total Status Entire list item is clickable and triggers <code>selectOrder</code> <p>Dynamic Behavior</p> <ol style="list-style-type: none"> Clicking an order: <ul style="list-style-type: none"> Calls <code>selectOrder(order)</code> to show that order's details in parent. Prices are formatted to two decimal places. <p>** Kindly refer to the screenshots for any clarifications. **</p> |
|--|--|---|

5 VALIDATIONS

- All required fields must be fulfilled with valid data.
- When logging into the system all the fields must be filled.
- When adding a book into the system all fields are mandatory to be filled.

6 CONSTRAINTS

- You should be able to press the "TAB" key and "SHIFT + TAB" to navigate from top field to bottom field and vice-versa.
- Once a user is logged in there should not be a "/login" url available until logged out.

7 MANDATORY ASSESSMENT GUIDELINES

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. This editor Auto Saves the code.
4. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
5. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
6. This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

Note: The application will not run in the local browser

7. You can follow series of command to setup React environment once you are in your project-name folder:
 - a. `npm install` -> Will install all dependencies -> takes 10 to 15 min
 - b. `npm run start` -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:8080/8081 to open project in browser -> takes 2 to 3 min
 - c. `npm run json-start` -> As we are using a json server to mimic our db.json file as a database. So, this command is useful to start a json server.
 - d. `npm run jest` -> to run all test cases and see the summary. It takes 5 to 6 min to run.
 - e. `npm run test` -> to run all test cases. **It is mandatory to run this command before submission of workspace** -> takes 5 to 6 min
8. You may also run “`npm run jest`” while developing the solution to re-factor the code to pass the test-cases.

9. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on **“Submit Assessment”** after you are done with code.