Book Store – User Module

IIHT

Time To Complete: 2 hr

CONTENTS

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 book buyers, 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.

2 Problem Statement

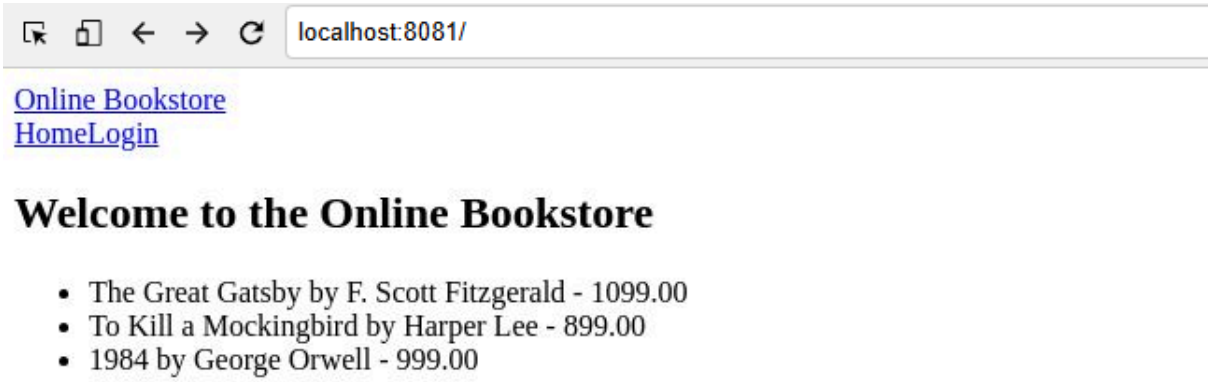
The "**Book Store App**" is a Single Page Application (SPA) designed to enhance the online book shopping experience. This system allows users to browse and search for books, add books to their cart, and place orders seamlessly.

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

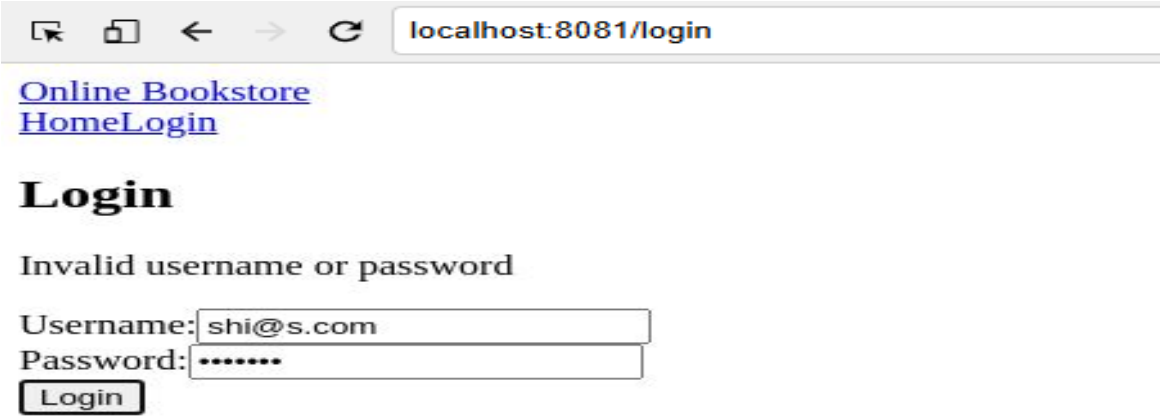
Already added three books with their details



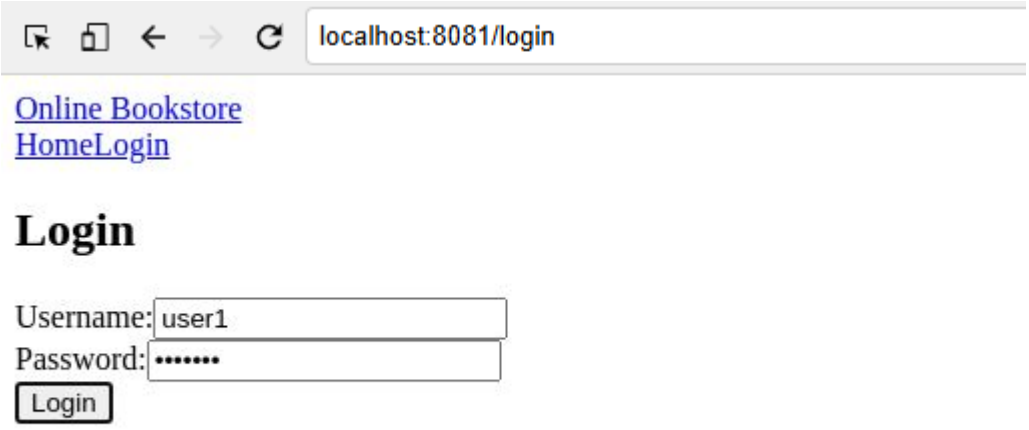
.

***** Login Page*****

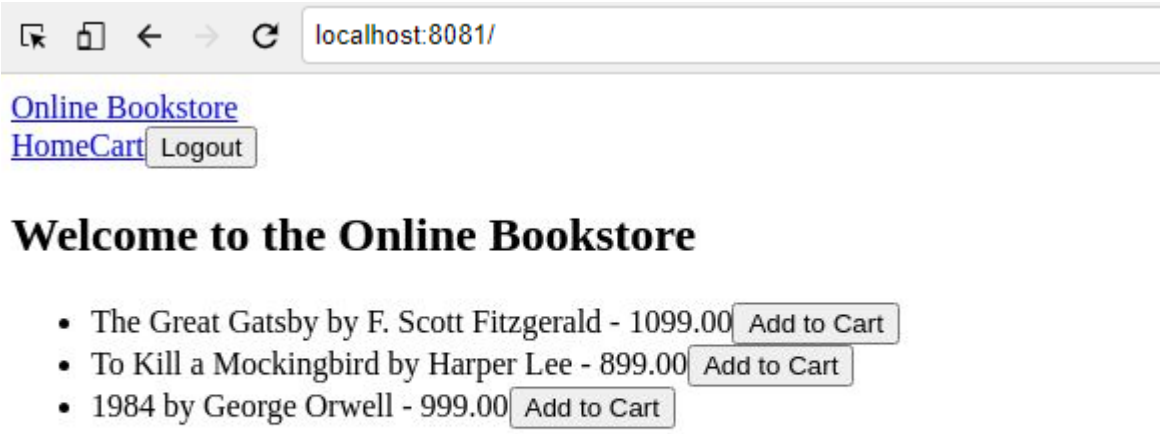
***** Invalid login*****



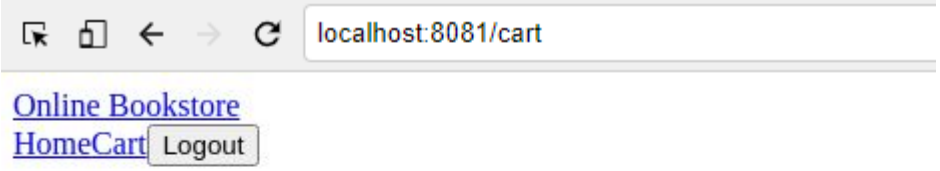
3.2 User Page



***** User Home Page*****

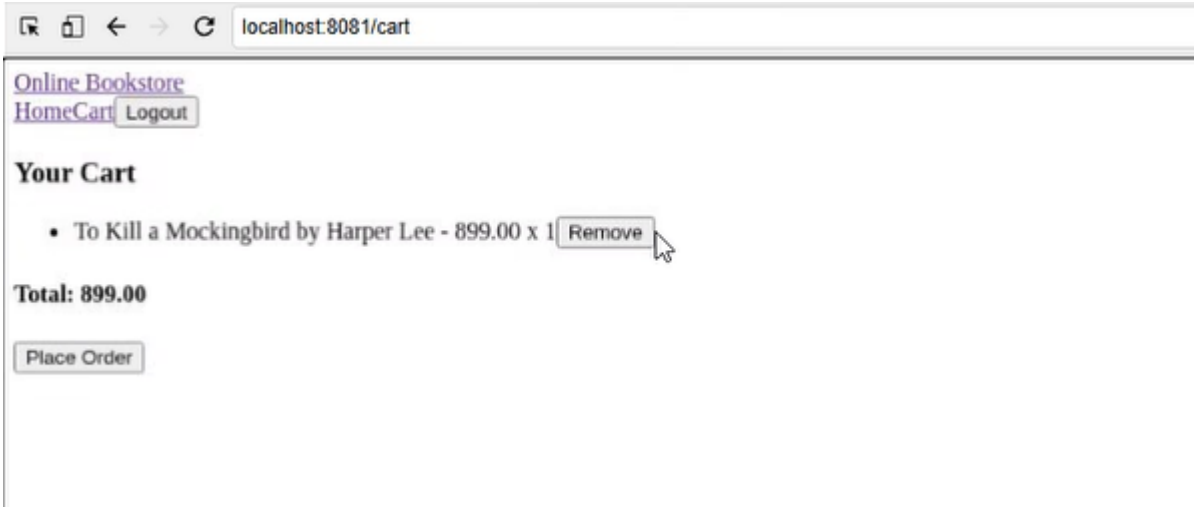


***** User Cart Page*****

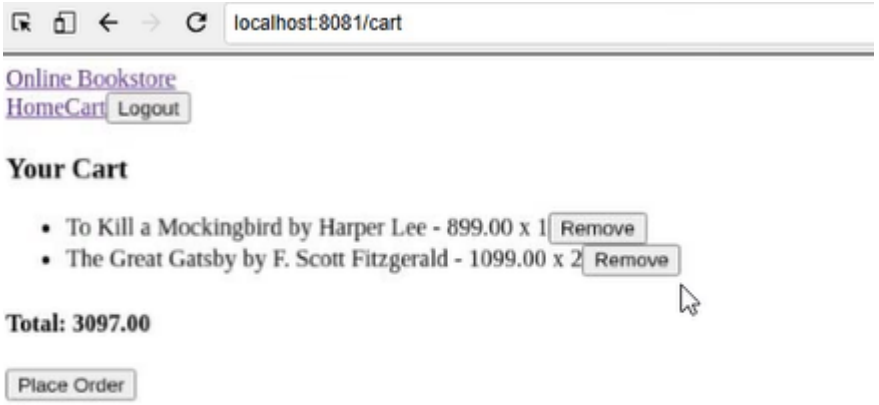


Your Cart

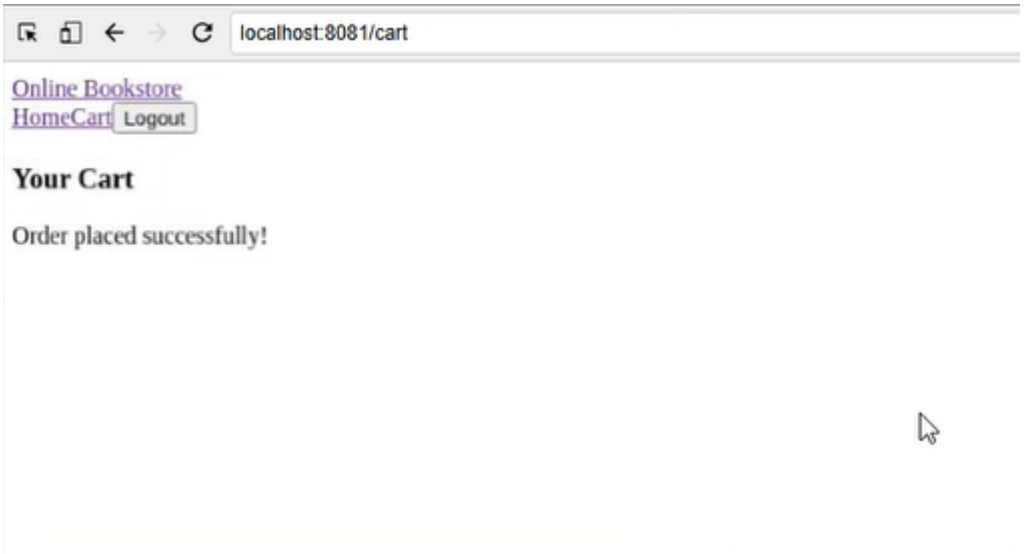
***** After removing a Book from Cart*****



***** After adding Books to Cart*****



***** After Placing an Order*****



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	**Acceptance Criteria - App Component** 1. Display a navigation bar at the top of the application (`<Navbar />`). 2. Route users to appropriate pages: ● Default route `/` shows the `HomePage`. ● Route `/login` shows the `LoginPage`. ● Route `/cart` accessible only to authenticated users with role `user`; otherwise redirects to `/login`. 3. Maintain user authentication state: ● Load from `localStorage` on mount. ● Store updates to `localStorage` on login/logout. 4. Use context (`AuthContext`) to provide global authentication access. **State & Context Overview** State Variables (via useState): - `auth`: null ● Represents the current authenticated user. ● Shape: `{username: string, role: 'user'}` - Context: `AuthContext` - Provides: ● `auth`: Current user info ● `login(user)`: 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:
- 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="/cart">`: Renders `<CartPage />` if user is a standard user; 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:
   - `/cart` route is protected:
     - ➔ Unauthenticated or unauthorized users are redirected to `/login`.

Cart Page

Acceptance Criteria – CartPage

1. Display a heading: “Your Cart”.

2. On component load:
   - Fetch the logged-in user's cart using their `userId`.

3. Display each book in the cart:
   - Title, author, price, quantity.
   - Include a “Remove” button to delete a book from the cart.

4. If the cart has items:
   - Show total cost.
   - Show a “Place Order” button.

5. On placing an order:
   - Send order details to the backend.
   - Clear the cart both in backend and UI.
   - Display success message in p tag as “Order placed successfully!”.

State & Props Overview

State Variables:

- `cart` (object | null): Contains cart data for the user.
  - ➔ Shape: `{ id, userId, books: [{ id, title, author, price, quantity }] }`

Context:
- `auth`: From `AuthContext`, contains the currently logged-in user info (`auth.id` used for cart fetch).

Functions & Responsibilities

1. fetchCart():
   - Axios GET request: `http://localhost:4000/cart?userId=${auth.id}`

		● Loads and sets cart state on component mount. 2. handleRemoveBook(bookId): ● Filters the book out of the current cart. ● Sends PUT request to update cart on the backend to `http://localhost:4000/cart/${cart.id}`. ● Updates `cart` state locally. 3. handlePlaceOrder(): ● Constructs an order object using cart data. ● Axios POST to `http://localhost:4000/orders` to create the order. ● Axios DELETE to remove the cart from backend using `http://localhost:4000/cart/${cart.id}`. ● Resets cart state to empty. ● Displays an alert on success. **HTML Structure** CartPage Component Layout: 1. `<div>`: Wraps the entire cart page. 2. `<h3>`: “Your Cart” 3. `<ul>`: Renders each item in `cart.books`: ● `<li>` for each book: ➔ Title, author, price, and quantity ➔ “Remove” button to delete the book 4. `<h4>`: Shows total price if there are items in cart. 5. `<button>`: “Place Order” button shown if cart is not empty. 6. `<p>`: To print order message on successfully order placed or any error as “Order placed successfully!” or “Failed to place order. Please try again.” respectively. **Dynamic Behaviour** 1. On Load: ● If user is not authenticated: display "Loading..." ● If authenticated: fetch cart and display contents. 2. Remove Book: ● Clicking “Remove” deletes the book via `PUT` request and updates local cart state. 3. Place Order: ● 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.

Home Page

Acceptance Criteria – HomePage

1. Display a welcome heading: “Welcome to the Online Bookstore”.

2. On page load:
   - Fetch and display a list of available books from the backend.

3. For authenticated users with the role `user`:
   - 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.

4. Show an alert upon successful addition to cart.

5. For non-authenticated or non-user roles, disable "Add to Cart".

State & Props Overview

State Variables:

- `books` (array): List of all books fetched from the backend.

Context:
- `auth`: From `AuthContext`, determines whether the user is logged in and their role.

Functions & Responsibilities

1. fetchBooks():
   - Axios GET request: `http://localhost:4000/books`
   - Fetches available books and sets the `books` state.

2. addToCart(book):
   - If the user is not logged in, exits early.
   - Checks if the user has an existing cart:
     - ➔ If not, creates a new cart with the selected book.
     - ➔ If yes, updates the cart:
       - ➢ 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.

HTML Structure

HomePage Component Layout:
1. `<div>`: Wraps entire page content.

2. `<h2>`: “Welcome to the Online Bookstore”

3. `<BookList />`: Displays the list of books.
   - Props:
     - ➔ `books`: Array of books from state.
     - ➔ `addToCart`: Function passed only if the user is authenticated and has the `user` role.

Dynamic Behavior

1. On Load:
   - Book list is fetched from the backend and rendered on the screen.

2. Add to Cart:
   - Only available to logged-in users with the `user` role.
   - Adds book to cart using backend requests:
     - ➔ Creates a new cart or updates existing one.
   - Book quantities are managed dynamically.
   - Alerts the user on success.

Login Page

Acceptance Criteria – LoginPage

1. Display a heading: “Login”.

2. Show a login form with:
   - Input fields for username and password.

   - Submit button labeled “Login”.

3. Validate form:
   - Both fields are required.

4. On form submission:
   - Make a request to fetch a user matching the provided credentials.
   - If a valid user is found:
     - ➔ Store authentication context using `login`.
     - ➔ Redirect:

		➢ Regular users → `/` ● If invalid, display an error message. 5. If already logged in, redirect to home page automatically. **State & Props Overview** **State Variables:** ● `credentials` (object): Stores the user's input. ➔ Shape: `{ username: string, password: string }` ● `error` (string): Stores any error message for invalid login or request failure. **Context:** ● `auth`: Current authenticated user. ● `login(user)`: Function from `AuthContext` to store login state globally. **Other Hooks:** ● `history`: From `useHistory` for programmatic navigation. **Functions & Responsibilities** 1. useEffect: ● Redirects to home (`/`) if a user is already authenticated. 2. handleChange(e): ● Updates `credentials` state as the user types into the form inputs. 3. handleSubmit(e): ● Prevents default form behavior. ● Makes a GET request to `http://localhost:4000/users` with query parameters. ● If user is found: ➔ Calls `login(user)` ➔ Redirects based on role. ● If not found or request fails: ➔ Sets error message accordingly. **HTML Structure** LoginPage Component Layout: 1. `<div className="login-page">`: Container for the login form.

		2. `<h2>`: “Login” 3. `<p className="error">`: Conditionally rendered if `error` exists. 4. `<form>`: Handles form submission. ● `<div>` with `<label>` and `<input>` for "Username" ● `<div>` with `<label>` and `<input>` for "Password" ● `<button>`: Submit button labeled “Login” **Dynamic Behavior** 1. Initial Redirect: ● If user is already logged in (`auth` exists), redirect to `/`. 2. Form Validation: ● Submit button is disabled by default by the `required` attribute. 3. Login Process: ● On successful match: ➔ Updates global auth context. ➔ Redirects based on role. ● On failure: ➔ Displays relevant error message. **BookList Component** **Acceptance Criteria - BookList** 1. Render a list of books. 2. For each book, display: ● Title, author, and price. 3. Show buttons conditionally: ● “Add to Cart” if `addToCart` function is passed. ● “View Details” if `selectBook` function is passed. ● “Delete” if `deleteBook` function is passed. **Props Overview** **Props Received:** ● `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.

Functions & Responsibilities

1. Direct event handlers within JSX:
   - Call appropriate prop function on button clicks.

HTML Structure

BookList Component Layout:
1. `<div class="book-list">`: Wraps the entire book list.

2. `<ul>`: Renders the book list.
   - `<li>`: For each book:
     - ➔ Displays:
       - ➢ Title
       - ➢ Author
       - ➢ Price
     - ➔ Conditionally renders buttons:
       - ➢ "Add to Cart"
       - ➢ "View Details"
       - ➢ "Delete"

Dynamic Behavior

1. Conditional UI:
   - Buttons only appear based on the props provided.
2. Event Handling:
   - Interactions are passed up to the parent via props.

Navbar Component

Acceptance Criteria - Navbar

1. Always display the brand link: "Online Bookstore" linking to `/`.

2. Display navigation links based on user authentication and role:
   - Everyone sees "Home"
   - Logged-in users with role `user` see:
     - ➔ "Cart"
   - If no user is logged in:
     - ➔ Show "Login"
   - If a user is logged in:
     - ➔ Show a "Logout" button

3. On clicking "Logout":

		● Clear authentication state. ● Redirect to the login page. **State & Context Overview** **Context Used:** ● `auth`: Authentication object from `AuthContext` ● `logout`: Function to clear authentication and local storage **Other Hooks:** ● `history`: From `useHistory` to navigate after logout **Functions & Responsibilities** 1. handleLogout(): ● Calls `logout()` to clear authentication context. ● Redirects to `/login`. **HTML Structure** Navbar Component Layout: 1. `<nav className="navbar">`: Wraps the entire navbar. ● `<div className="navbar-brand">`: Contains: ➔ `<Link>` to `/` labelled "Online Bookstore" ● `<div className="navbar-links">`: Contains: ➔ `<Link>` to `/` labelled "Home" ➔ Conditional links: ➢ If user role is `user`: `<Link>` to `/cart` ➔ If not authenticated: `<Link>` to `/login` ➔ If authenticated: `<button>` labelled "Logout" **Dynamic Behaviour** 1. Role-based Navigation: ● Links rendered based on `auth` and user role. 2. Authentication Switching: ● Toggling login/logout dynamically changes the visible links. 3.Logout Action: ● Clears authentication context. ● Navigates to login page. **** Kindly refer to the screenshots for any clarifications. ****

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.**