

REACT-LIFE CYCLE HOOKS

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1	Project Abstract	3
2	Problem Statement	3
3	Proposed Life cycle hooks Application Wireframe	4
3.1	Screenshots	5
4	Business-Requirement:	6
5	Constraints	6
6	Mandatory Assessment Guidelines	7

1 PROJECT ABSTRACT

React applications often need to perform actions during different phases of a component's life cycle, such as fetching data, initializing values, or cleaning up resources. This project introduces participants to the concept of React's **life cycle hooks**—particularly focusing on state updates and side effects. Participants will build a User Dashboard application where users can search for a user by ID. Although this version uses mock data, the design structure prepares developers to later integrate real-world asynchronous data fetching (e.g., using `useEffect`). The project highlights how React manages component updates and renders based on state and props changes.






2 PROBLEM STATEMENT

You are required to develop a React User Dashboard where users can input a user ID and retrieve user details from mock data. The `UserDashboard` component manages the state for user input, search results, and errors. When a user submits a form, the application dynamically searches for the corresponding user and updates the interface accordingly. Even though external data fetching is mocked here, the project focuses on understanding how state updates trigger re-renders, mimicking common life cycle behaviors seen in React applications when using hooks like `useEffect`.

3 PROPOSED LIFE CYCLE HOOKS APPLICATION WIREFRAME

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

3.1 SCREENSHOTS








localhost:8081/

User Dashboard

Get User

No user data available.

Get user




localhost:8081/

User Dashboard

Get User

No user data available.



localhost:8081/

User Dashboard

Get User

Leanne Graham

Email: Sincere@april.biz

Address: Kulas Light, Gwenborough

No user ID

localhost:8081/

User Dashboard

5

Leanne Graham

Email: Sincere@april.biz

Address: Kulas Light, Gwenborough

localhost:8081/

User Dashboard

Enter user ID

No user data available.

4 BUSINESS-REQUIREMENT:

As an application developer, develop the Life cycle hooks (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Welcome Page	<p>As a user I should be able to visit the welcome page as the default page.</p> <p>Acceptance criteria:</p> <ol style="list-style-type: none">1. Display a heading "User Dashboard".2. Provide an input form to:<ul style="list-style-type: none">• Enter a user ID

		<ul style="list-style-type: none"> • Submit the ID to search for a user <ol style="list-style-type: none"> 3. After submission: <ul style="list-style-type: none"> • If user found → display the user's name, email, and address. • If user not found → show an appropriate error message. 4. Handle state changes properly when submitting new searches. <p>State & Functionality Overview:</p> <p>UserDashboard Component:</p> <p>State Variables:</p> <ol style="list-style-type: none"> 1. <code>userId</code> <ul style="list-style-type: none"> • Type: <code>string</code> • Stores the input user ID. 2. <code>user</code> <ul style="list-style-type: none"> • Type: <code>object null</code> • Stores the found user details if available. 3. <code>error</code> <ul style="list-style-type: none"> • Type: <code>string</code> • Stores an error message if the user is not found. <p>Method: <code>handleSubmit(userId)</code>:</p> <ol style="list-style-type: none"> 1. Sets the current <code>userId</code>. 2. Resets <code>user</code> and <code>error</code>. 3. Searches in the <code>mockUsers</code> dataset. 4. Updates <code>user</code> or <code>error</code> based on the result. <p>UserForm Component</p> <p>Props Received:</p> <ol style="list-style-type: none"> 1. <code>onSubmit</code>: Function to handle user search when form is submitted. <p>Responsibilities:</p> <ol style="list-style-type: none"> 1. Render an input field to accept a user ID. 2. On form submit: <ul style="list-style-type: none"> • Call <code>onSubmit(userId)</code> to trigger a search. • Clear the input field afterward. <p>UserCard Component</p>
--	--	---

		<p>Props Received:</p> <ol style="list-style-type: none"> 1. <code>user</code>: Object with user details. 2. <code>error</code>: String showing an error if user is not found. <p>Responsibilities:</p> <ol style="list-style-type: none"> 1. If <code>user</code> is <code>null</code> → show message: "No user data available." 2. If <code>error</code> exists → show the error message. 3. Otherwise display: <ul style="list-style-type: none"> • User Name • Email • Full Address (Street + City) <p>HTML Structure:</p> <ol style="list-style-type: none"> 1. App Component: <ul style="list-style-type: none"> • Wraps everything inside a top-level <code><div></code>. • Renders the <code>UserDashboard</code> component. 2. UserDashboard: <ul style="list-style-type: none"> • <code><h1></code> heading ("User Dashboard") • <code>UserForm</code> (form with input and submit button) • <code>UserCard</code> (shows user details or fallback/error) 3. UserForm: <ul style="list-style-type: none"> • Input field (type "<code>number</code>") to accept a user ID. • Submit button to trigger search. 4. UserCard: <ul style="list-style-type: none"> • Conditionally renders: <ul style="list-style-type: none"> → Fallback message → Error message → Or user details <p>Dynamic Behavior:</p> <ol style="list-style-type: none"> 1. Typing into the form updates the <code>userId</code> state dynamically. 2. Submitting the form triggers a lookup into mock data. 3. The result is conditionally displayed based on success or failure.
--	--	---

		** Kindly refer to the screenshots for any clarifications. **

5 CONSTRAINTS

1. You should be able to press the "TAB" key and "SHIFT + TAB" to navigate from top field to bottom field and vice-versa.

6 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. 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

6. 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 jest -> to run all test cases and see the summary.
 - d. npm run test -> to run all test cases. **It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**

7. 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.