**Lists , Hooks , Localstorage , Api Project**

**Lists**

**THEORY EXERCISE**

**Question 1:**  
**How do you render a list of items in React? Why is it important to use keys when rendering lists?**

* To render a list in React, you use JavaScript’s .map() method to iterate over the array and return JSX for each item.
* **Keys** are important because they help React identify which items have changed, added, or removed, improving rendering performance and preventing bugs.

**LAB EXERCISE**

**Task 1:**  
Create a React component that renders a list of fruit names using .map().

function FruitList() {

const fruits = ['Apple', 'Banana', 'Orange', 'Mango'];

return (

<ul>

{fruits.map((fruit, index) => (

<li key={index}>{fruit}</li>

))}

</ul>

);

}

export default FruitList;

**Hooks**

Great! Here's a detailed explanation for your React hooks theory questions and starter code examples for the lab exercises.

**THEORY EXERCISE**

**Question 1:**  
**What are React hooks? How do useState() and useEffect() hooks work in functional components?**

* React **hooks** are special functions that let you use state and other React features in functional components.
* useState() allows you to add state to a functional component by returning a state variable and a function to update it.
* useEffect() lets you perform side effects (like data fetching, subscriptions) in functional components. It runs after render and can depend on specific variables.

**Question 2:**  
**What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

* Hooks solved the problem of managing state and lifecycle methods only in class components, enabling stateful logic in functional components.
* They promote code reuse, cleaner components, and better organization of logic without needing classes.

**Question 3:**  
**What is useReducer? How do we use it in a React app?**

* useReducer is a hook to manage complex state logic by using a reducer function and dispatching actions, similar to Redux but localized.
* It returns the current state and a dispatch function to update the state based on actions.

**Question 4:**  
**What is the purpose of useCallback & useMemo hooks?**

* useCallback memoizes functions to avoid unnecessary re-creation on every render, useful for performance optimization.
* useMemo memoizes the result of expensive calculations to avoid recalculations on every render.

**Question 5:**  
**What’s the difference between useCallback & useMemo hooks?**

* useCallback returns a memoized function.
* useMemo returns a memoized value/result of a function.

**Question 6:**  
**What is useRef? How does it work in a React app?**

* useRef returns a mutable object whose .current property persists across renders.
* It’s used to reference DOM elements directly or store mutable values without causing re-renders.

**LAB EXERCISE**

**Task 1:** Counter with useState

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<h2>Count: {count}</h2>

<button onClick={() => setCount(count + 1)}>Increment</button>

<button onClick={() => setCount(count - 1)}>Decrement</button>

</div>

);

}

export default Counter;

**Task 2:** Fetch data with useEffect

import React, { useState, useEffect } from 'react';

function DataFetcher() {

const [data, setData] = useState(null);

useEffect(() => {

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(res => res.json())

.then(json => setData(json))

.catch(err => console.error(err));

}, []); // Empty dependency array => run once on mount

return (

<div>

{data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}

</div>

);

}

export default DataFetcher;

**Task 3:** Using useSelector and useDispatch (Redux hooks)

// Assuming you have redux set up with a store and slice

import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { increment, decrement } from './counterSlice'; // example actions

function ReduxCounter() {

const count = useSelector(state => state.counter.value);

const dispatch = useDispatch();

return (

<div>

<h2>Count: {count}</h2>

<button onClick={() => dispatch(increment())}>Increment</button>

<button onClick={() => dispatch(decrement())}>Decrement</button>

</div>

);

}

export default ReduxCounter;

**Task 4:** Avoid re-renders using useRef

import React, { useState, useRef } from 'react';

function RefExample() {

const [count, setCount] = useState(0);

const renders = useRef(0);

renders.current += 1;

return (

<div>

<p>Render count: {renders.current}</p>

<h2>Count: {count}</h2>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

export default RefExample;