Lists , Hooks , Localstorage , Api Project

List and keys:-

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

Answer:
To render a list of items in React, you can use the `map()` function to iterate over an array and return a JSX element for each item.

Example:

```
jsx
Copy code
const fruits = ["Apple", "Banana", "Cherry"];
const fruitList = fruits.map((fruit) => <li key={fruit}>{fruit}</li>);
```

Importance of keys:
Keys help React identify which items have changed, are added, or are removed. This improves rendering performance and ensures that the UI updates efficiently.

Question 2: What are keys in React, and what happens if you do not provide a unique key?

Answer:
Keys are unique identifiers assigned to elements in a list to help React distinguish between different items.

If you do not provide a unique key:

- React may not update the UI correctly because it cannot track changes accurately.
- Performance may degrade due to unnecessary re-renders.
- You might see a warning in the console about missing keys.

LAB EXERCISE

Task 1: Render a List of Items (Fruit Names)

```
jsx
Copy code
import React from "react";

const FruitList = () => {
  const fruits = ["Apple", "Banana", "Cherry", "Mango", "Orange"];
```

```
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
};

export default FruitList;
```

Task 2: Render a List of Users with Unique Keys

```
jsx
Copy code
import React from "react";

const UserList = () => {
  const users = [
    { id: 1, name: "Milan", age: 20 },
    { id: 2, name: "John", age: 25 },
    { id: 3, name: "Sara", age: 22 },
  ];

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>
          {user.name} - Age: {user.age}
        </li>
      ))}
    </ul>
  );
};

export default UserList;
```

HOOKS:-

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

Answer:
React hooks are special functions that let you use React features (like state and lifecycle) in functional components.

- **`useState()`**:
  - Allows you to add state to a functional component.
  - Syntax:

    ```
    jsx
    Copy code
    const [state, setState] = useState(initialValue);
    ```

- **`useEffect()`**:
  - Allows you to perform side effects in functional components (e.g., fetching data, subscribing to events).
  - Syntax:

    ```
    jsx
    Copy code
    useEffect(() => {
      // Side effect code
      return () => {
        // Cleanup code (optional)
      };
    }, [dependencies]);
    ```

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

Answer:
Hooks solved several problems:

1. **Code Reusability**: Eliminated the need for HOCs and render props to share logic.
2. **Simplified State Management**: Allowed state and lifecycle methods in functional components.
3. **Reduced Complexity**: Simplified class component lifecycle methods into a single `useEffect`.

Hooks are important because they make functional components as powerful as class components while keeping code cleaner and more maintainable.

Question 3: What is `useReducer`? How is it used in a React app?

Answer:
`useReducer` is a hook for managing complex state logic in functional components. It's an alternative to `useState` for state that depends on previous state or has multiple sub-values.

Syntax:

```
jsx
Copy code
const [state, dispatch] = useReducer(reducer, initialState);

function reducer(state, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
}
```

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

Answer:

- **`useCallback`**: Memoizes a function, preventing it from being re-created unless its dependencies change.
- **`useMemo`**: Memoizes a value, preventing expensive recalculations unless dependencies change.

Question 5: What's the difference between `useCallback` & `useMemo` hooks?

Answer:

- **`useCallback`**: Returns a memoized function.
- **`useMemo`**: Returns a memoized value.

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

Answer:
`useRef` is a hook that provides a way to access and manipulate DOM elements or persist mutable values across renders without causing re-renders.

Syntax:

```
jsx
Copy code
const ref = useRef(initialValue);
```

LAB EXERCISE

Task 1: Counter with `useState` Hook

```
jsx
Copy code
import React, { useState } from "react";

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

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
};

export default Counter;
```

Task 2: Fetch Data with `useEffect` Hook

```
jsx
Copy code
import React, { useState, useEffect } from "react";

const FetchData = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []);

  return (
    <div>
      <h1>Data from API</h1>
      <ul>
        {data.map((item) => (
          <li key={item.id}>{item.title}</li>
        ))}
      </ul>
    </div>
  );
};

export default FetchData;
```

Task 3: App with `useSelector` & `useDispatch`

Install Redux and React-Redux:

```
bash
Copy code
npm install redux react-redux
```

Example:

```
jsx
Copy code
import React from "react";
import { createStore } from "redux";
import { Provider, useSelector, useDispatch } from "react-redux";

const initialState = { count: 0 };

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const store = createStore(reducer);

const Counter = () => {
  const count = useSelector((state) => state.count);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => dispatch({ type: "INCREMENT"
})}>Increment</button>
      <button onClick={() => dispatch({ type: "DECREMENT"
})}>Decrement</button>
    </div>
  );
};

const App = () => (
  <Provider store={store}>
    <Counter />
  </Provider>
);

export default App;
```

Task 4: Avoid Re-renders with `useRef`

```
jsx
Copy code
import React, { useRef, useState } from "react";
```

```
const AvoidRerender = () => {
  const renderCount = useRef(0);
  const [count, setCount] = useState(0);

  renderCount.current++;

  return (
    <div>
      <h1>Count: {count}</h1>
      <h2>Renders: {renderCount.current}</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default AvoidRerender;
```

API:-

Question 1: What is the Context API in React? How is it used to manage global state across multiple components?

Answer:
The Context API in React is a feature that allows you to share state and data across multiple components without passing props down manually at every level.

How it works:

- It provides a way to create a global state accessible to any component in the component tree.
- It is useful for themes, authentication, and other global data.

Steps:

1. Create a context using `createContext()`.
2. Wrap components with a `Provider` to supply data.
3. Use `useContext()` to consume the data in child components.

Question 2: Explain how `createContext()` and `useContext()` are used in React for sharing state.

Answer:

- **`createContext()`**:
  - Creates a context object that can be used to provide and consume data.
  - Example:

    ```
    jsx
    Copy code
    const ThemeContext = createContext();
    ```

- **`useContext()`**:
  - Allows components to access the context value without manually passing props.
  - Example:

```
jsx
Copy code
const theme = useContext(ThemeContext);
```

LAB EXERCISE

Task 1: Theme Toggle (Light/Dark Mode)

Code Implementation:

```
jsx
Copy code
import React, { createContext, useContext, useState } from "react";

// Create Context
const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");

  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === "light" ? "dark" : "light"));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

const Header = () => {
  const { theme, toggleTheme } = useContext(ThemeContext);
  return (
    <header
      style={{
        backgroundColor: theme === "light" ? "#fff" : "#333",
        color: theme === "light" ? "#000" : "#fff",
        padding: "10px",
        textAlign: "center",
      }}
    >
      <h1>{theme === "light" ? "Light Mode" : "Dark Mode"}</h1>
      <button onClick={toggleTheme}>
        Toggle to {theme === "light" ? "Dark" : "Light"} Mode
      </button>
    </header>
  );
};

const App = () => {
  return (
```

```
    <ThemeProvider>
      <Header />
    </ThemeProvider>
  );
};

export default App;
```

Task 2: Global User Authentication System

Code Implementation:

```
jsx
Copy code
import React, { createContext, useContext, useState } from "react";

// Create Context
const AuthContext = createContext();

const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);

  const login = (username) => {
    setUser({ name: username });
  };

  const logout = () => {
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

const Header = () => {
  const { user, login, logout } = useContext(AuthContext);

  return (
    <header style={{ padding: "10px", textAlign: "center" }}>
      {user ? (
        <>
          <h1>Welcome, {user.name}!</h1>
          <button onClick={logout}>Logout</button>
        </>
      ) : (
        <>
          <h1>Please log in</h1>
          <button onClick={() => login("Milan")}>Login</button>
        </>
      )}
    </header>
  );
};

const App = () => {
  return (
```

```
    <AuthProvider>
      <Header />
    </AuthProvider>
  );
};

export default App;
```
