Don Bosco Institute of Technology
Kurla (W), Mumbai 400 070

Internet Programming Lab

Name: Umme Atiya Quraishi
Class: TEIT
Roll no.: 47
Date : 30/08/2024
Experiment no. 9
Title:To Design a web pages using ReactJS Hooks

Theory:

React Hooks are functions that let developers use state and other React features in functional components. Introduced in React 16.8, hooks aim to simplify component logic and enhance code reuse.

Key Concepts

1. **Purpose of Hooks**:
    - Hooks allow functional components to have state and lifecycle capabilities, which were previously exclusive to class components.
    - They promote cleaner and more maintainable code by enabling better separation of concerns and logic encapsulation.
2. **Types of Hooks**:
    - **Basic Hooks**: These include `useState` for managing state and `useEffect` for handling side effects (like data fetching or subscriptions).
    - **Additional Hooks**: Other hooks like `useContext`, `useReducer`, `useRef`, and others are available for specific use cases, enhancing the capabilities of functional components.
3. **Advantages of Using Hooks**:
    - **Simplified State Management**: Hooks provide a straightforward way to manage component state without needing classes.
    - **Improved Readability**: By avoiding the complexity of class components, hooks make code easier to read and understand.

- **Enhanced Code Reusability**: Hooks allow you to extract component logic into reusable functions, promoting better modularization.
- **Encourages Functional Programming**: Hooks align with functional programming paradigms, making it easier to think about your components in terms of functions and data transformations.

4. **Lifecycle Methods**:
   - Hooks provide alternatives to lifecycle methods found in class components. For instance, `useEffect` can replicate behaviors of `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.
5. **Best Practices**:
   - Hooks should only be called at the top level of a component or from other hooks to ensure the order of hook calls remains consistent across renders.
   - It's essential to use hooks in functional components only, as they are not compatible with class components.
6. **Limitations**:
   - Although hooks enhance functional components, they might require a paradigm shift for developers who are accustomed to class-based components.
   - Hooks do not provide a performance boost by themselves but help in organizing code in a more manageable way.

Conclusion

React Hooks revolutionize the way developers create functional components by providing powerful tools for state management and side effects. They enable cleaner, more organized, and reusable code while simplifying the component lifecycle, making them a key feature in modern React development.

Code:

```
// UserDashboard.js

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

function UserDashboard() {
  const [users, setUsers] = useState([]);
  const [newUser, setNewUser] = useState({ name: '', email: '' });
```

```
useEffect(() => {
  const fetchedUsers = [
    { id: 1, name: 'Alice', email: 'alice@example.com' },
    { id: 2, name: 'Bob', email: 'bob@example.com' },
  ];
  setUsers(fetchedUsers);
}, []);

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setNewUser((prevUser) => ({ ...prevUser, [name]: value }));
};

const handleAddUser = () => {
  if (newUser.name && newUser.email) {
    const newUserObj = {
      id: users.length + 1,
      name: newUser.name,
      email: newUser.email,
    };
    setUsers([...users, newUserObj]);
    setNewUser({ name: '', email: '' });
  }
};

return (
  <div style={dashboardStyle}>
    <h1>User Profile Dashboard</h1>

    <div style={formStyle}>
      <h2>Add New User</h2>
      <input
        type="text"
        name="name"
        placeholder="Enter name"
        value={newUser.name}
```

```
          onChange={handleInputChange}
          style={inputStyle}
        />
        <input
          type="email"
          name="email"
          placeholder="Enter email"
          value={newUser.email}
          onChange={handleInputChange}
          style={inputStyle}
        />
        <button onClick={handleAddUser} style={buttonStyle}>Add User</button>
      </div>

      <div style={userListStyle}>
        <h2>Users</h2>
        {users.length > 0 ? (
          users.map((user) => (
            <div key={user.id} style={userCardStyle}>
              <h3>{user.name}</h3>
              <p>{user.email}</p>
            </div>
          ))
        ) : (
          <p>No users available</p>
        )}
      </div>
    </div>
  );
}

// Styles
const dashboardStyle = {
  padding: '20px',
  fontFamily: 'Arial, sans-serif',
  textAlign: 'center',
};
```

```
const formStyle = {
  marginBottom: '20px',
};

const inputStyle = {
  margin: '10px',
  padding: '10px',
  width: '200px',
  borderRadius: '5px',
  border: '1px solid #ccc',
};

const buttonStyle = {
  padding: '10px 20px',
  backgroundColor: '#4CAF50',
  color: 'white',
  border: 'none',
  borderRadius: '5px',
  cursor: 'pointer',
};

const userListStyle = {
  marginTop: '20px',
};

const userCardStyle = {
  border: '1px solid #ccc',
  padding: '15px',
  margin: '10px auto',
  width: '300px',
  borderRadius: '8px',
  textAlign: 'left',
};

export default UserDashboard;
```

App.js:

```
import React from 'react';
import './App.css';
import UserDashboard from './exp9';
import ProductList from './ProductList'; // Import ProductList component


function App() {
  return (
    <div className="App">
        <UserDashboard />

            {/* <ProductList />  */}

    </div>
  );
}

export default App;
```

Output:

User Profile Dashboard

Add New User

Enter name

Enter email

Add User

Users

Alice

alice@example.com

Bob

bob@example.com