REACT THEORY QUESTIONS :

1. What is useState in React? Why is it used in form handling?

useState is a Hook that lets you add state to functional components. It returns an array with two things: the current value and a function to change it.

In form handling, useState tracks what the user types in each input field. When they type, it updates the state, and React re-renders to show the latest value. This makes forms interactive and allows you to save the form data for submission.

2. What is the difference between controlled and uncontrolled components?

Controlled components: React controls the form input. The value comes from state, and onChange updates the state. React is always the source of truth for what is shown.

Uncontrolled components: The DOM controls the form input. React does not track the value - it just exists in the HTML element. You use useRef to grab the value when needed.

Key difference: Controlled = React manages data. Uncontrolled = DOM manages data.

3. How does localStorage work in React applications?

localStorage is like a small storage box in the browser that saves data even after you close the page. It uses key-value pairs like a dictionary.

In React, you use localStorage.setItem() to save data and localStorage.getItem() to read it. This is useful for saving login info or form data so users do not lose their information when they refresh the page.

4. Why do we convert objects to JSON before storing in localStorage?

localStorage only stores text strings. If you try to save a JavaScript object directly, it becomes "[object Object]" - useless.

JSON.stringify() converts objects into a text format that keeps all the information. JSON.parse() converts it back to an object. This way, complex data like user profiles stay intact when stored and retrieved.

5. What is a Higher Order Component (HOC)? Where is it used?

An HOC is a function that takes a component as input and returns a new, enhanced component. It wraps a component to add extra features.

Example use: You have a Login page and a Dashboard. The Dashboard should only show if the user is logged in. Create an HOC that checks localStorage for currentUser. If not found, redirect to login. If found, show the Dashboard. This way, you protect the Dashboard from unauthorized access.

6. What is the purpose of React Router DOM?

React Router DOM lets you navigate between different pages without refreshing the browser. It is like having multiple pages in one application.

Instead of separate HTML files for each page, React Router switches between components based on the URL. This makes the app faster and keeps the state alive during navigation.

7. What is the use of <Link> instead of <a> tag in React?

<Link> from React Router is smarter than <a> tags. When you click <a>, the browser reloads the entire page. When you click <Link>, only the component changes.

<Link> keeps your app state intact, loads faster, and gives a better user experience. That is why <Link> is preferred in React Router apps.

8. Name three array methods commonly used in React and their purpose.

1. .map() - Transforms each item in an array. Most common use: turning a list of users into list items to display on the page.

2. .filter() - Creates a new array with only items that match a condition. Example: filter users by age, or show only active users.

3. .find() - Returns the first item that matches a condition. Example: find a user by ID from a list of users.

9. How does conditional rendering work in React?

Conditional rendering means showing different content based on a condition. Use if-else, ternary operators (? :), or && logic.

Example: If user is logged in, show Dashboard. If not, show Login page. This keeps the UI dynamic and responsive to the application state.

10. What is the role of useParams in React Router?

useParams extracts variables from the URL. When you have a route like /user/:id, useParams lets you grab the id value.

Example: If URL is /user/5, useParams returns {id: "5"}. You can then use this to fetch that user data from an API.

REACT LOGIC QUESTIONS :

1. Register Form with localStorage

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

export default function Register() {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    const existingUsers = JSON.parse(localStorage.getItem('users')) || [];
    const newUser = {
      id: Date.now(),
      name: formData.name,
      email: formData.email,
      password: formData.password
    };

    existingUsers.push(newUser);
    localStorage.setItem('users', JSON.stringify(existingUsers));
    setFormData({ name: '', email: '', password: '', confirmPassword: '' });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" name="name" value={formData.name} onChange={handleChange} />
      <input type="email" name="email" value={formData.email} onChange={handleChange}
/>
      <input type="password" name="password" value={formData.password}
onChange={handleChange} />
      <input type="password" name="confirmPassword" value={formData.confirmPassword}
onChange={handleChange} />
      <button type="submit">Register</button>
    </form>
  );
}
```

2. Login Page with Validation

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

export default function Login() {
  const [credentials, setCredentials] = useState({ email: '', password: '' });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setCredentials(prev => ({ ...prev, [name]: value }));
  };

  const handleLogin = (e) => {
    e.preventDefault();

    const users = JSON.parse(localStorage.getItem('users')) || [];
    const user = users.find(u => u.email === credentials.email && u.password ===
credentials.password);

    if (user) {
      localStorage.setItem('currentUser', JSON.stringify({ id: user.id, name: user.name, email:
user.email }));
      window.location.href = '/dashboard';
    } else {
      alert('Invalid credentials');
    }
  };

  return (
    <form onSubmit={handleLogin}>
      <input type="email" name="email" value={credentials.email} onChange={handleChange}
/>
      <input type="password" name="password" value={credentials.password}
onChange={handleChange} />
      <button type="submit">Login</button>
    </form>
  );
}
```

3. Display Multiple Users with .map()

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

export default function UsersList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    const storedUsers = JSON.parse(localStorage.getItem('users')) || [];
    setUsers(storedUsers);
  }, []);

  return (
    <div>
      <h2>Users List</h2>
      {users.map(user => (
        <div key={user.id}>
          <p>Name: {user.name}</p>
          <p>Email: {user.email}</p>
        </div>
      ))}
    </div>
  );
}
```

4. Edit and Delete Users

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

export default function ManageUsers() {
  const [users, setUsers] = useState([]);
  const [editingId, setEditingId] = useState(null);
  const [editData, setEditData] = useState({ name: '', email: '' });

  useEffect(() => {
    setUsers(JSON.parse(localStorage.getItem('users')) || []);
  }, []);

  const handleDelete = (id) => {
    const updatedUsers = users.filter(user => user.id !== id);
    setUsers(updatedUsers);
    localStorage.setItem('users', JSON.stringify(updatedUsers));
  };

  const handleEdit = (user) => {
    setEditingId(user.id);
    setEditData({ name: user.name, email: user.email });
  };

  const handleSave = (id) => {
    const updatedUsers = users.map(user =>
      user.id === id ? { ...user, name: editData.name, email: editData.email } : user
    );
    setUsers(updatedUsers);
    localStorage.setItem('users', JSON.stringify(updatedUsers));
    setEditingId(null);
  };

  return (
    <table>
      <tbody>
        {users.map(user => (
          <tr key={user.id}>
            <td>{editingId === user.id ? <input value={editData.name} onChange={(e) =>
setEditData({ ...editData, name: e.target.value })} /> : user.name}</td>
            <td>{editingId === user.id ? <input value={editData.email} onChange={(e) =>
setEditData({ ...editData, email: e.target.value })} /> : user.email}</td>
            <td>
              {editingId === user.id ? (
                <>
                  <button onClick={() => handleSave(user.id)}>Save</button>
                  <button onClick={() => setEditingId(null)}>Cancel</button>
                </>
              ) : (
                <>
                  <button onClick={() => handleEdit(user)}>Edit</button>
                  <button onClick={() => handleDelete(user.id)}>Delete</button>
                </>
              )}
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  );
}
```

5. Protected Route Using HOC

```
import React from 'react';

const ProtectedRoute = (Component) => {
  return (props) => {
    const currentUser = localStorage.getItem('currentUser');

    if (!currentUser) {
      window.location.href = '/login';
      return null;
    }

    return <Component {...props} />;
  };
};

const Dashboard = () => {
  const currentUser = JSON.parse(localStorage.getItem('currentUser'));

  const handleLogout = () => {
    localStorage.removeItem('currentUser');
    window.location.href = '/login';
  };

  return (
    <div>
      <h2>Welcome, {currentUser.name}</h2>
      <p>Email: {currentUser.email}</p>
      <button onClick={handleLogout}>Logout</button>
    </div>
  );
};

export default ProtectedRoute(Dashboard);
```