States

State is nothing but an object which is privately maintained inside a component. State can influence what is rendered on the browser, state can be changed within the component

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

class Message extends Component {

   constructor() {
      super();
      this.state= {
         message: "Welcome Visitor"
      };
   }

   changeMessage = () => {
      this.setState({
         message: "Thank you for subscribing!"
      });
   };

   render() {
      return (
         <div>
            <h1>{this.state.message}</h1>
            <button  onClick={this.changeMessage}>Subscribe</button>
         </div>
      );
   }
}

export default Message;
```

Updating the state In Functional Component by using useState hook

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

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

  return (
    <div>
       Count - {count}<br/>
       <button type='submit' onClick={()=>setCount(prevCount => prevCount + 1)}>Increment</button>
    </div>
  )
}

export default Counter
```

Difference b/w props and states

props	**states**
1. props get passed to a component	state is managed within the component
2. Function Parameter	Variables declared in the function body
3. props are immutable	state can be changed
4. props-Functional Component	useState Hook-Functional Component
5. this.props-Class Component	this.state-Class Component

React Router

React Router is a popular library for handling navigation (routing) in single-page React applications. It enables switching between views (components) without reloading the page, offering a smooth user experience.

BrowserRouter

BrowserRouter wraps your entire app and enables routing by using the browser's history system. It listens for URL changes and tells React Router to load the appropriate component.

```
import { BrowserRouter } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <App />
    </BrowserRouter>
  );
}
```

Routes

<Routes> contains all <Route> elements. It checks the current URL and matches it with the first <Route> that fits, rendering the corresponding component.

```
import { Routes, Route } from 'react-router-dom';


function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
  );
}
```

Route

A <Route> defines a path and the component that should be displayed when the path matches.

<Route path="/about" element={<About />} />

Link

<Link> allows navigation between pages without a full page reload (similar to an anchor <a> tag).

```
import { Link } from 'react-router-dom';
<Link to="/about">About Page</Link>
```

useNavigate Hook

useNavigate() is a hook for programmatically navigating to a different route in your app.

```
import { useNavigate } from 'react-router-dom';

function Form() {
  const navigate = useNavigate();
  const handleSubmit = () => {
    // Perform some action and then navigate
    navigate('/success');
  };
}
```

useParams Hook

useParams() is used to extract dynamic values from the URL, useful for routes like /user/:id

```
import { useParams } from 'react-router-dom';

function User() {
  const { id } = useParams();
  return <div>User ID: {id}</div>;
}
```

Nested Routes

Nested routes allow you to define sub-routes inside a parent route. Use <Outlet> in the parent route to display nested components.

```
<Route path="/dashboard" element={<Dashboard />}>
  <Route path="profile" element={<Profile />} />
</Route>

// In Dashboard component
import { Outlet } from 'react-router-dom';

function Dashboard() {
  return (
    <div>
      <h2>Dashboard</h2>
      <Outlet />  {/* Nested routes are displayed here */}
    </div>
  );
}
```

Navigate

The <Navigate> component allows you to programmatically redirect the user to another route.

```
import { Navigate } from 'react-router-dom';
<Route path="/old" element={<Navigate to="/new" />} />
```

Private Routes

Private routes are used to restrict access based on user conditions, such as being logged in.

```
function PrivateRoute({ children }) {
  const isLoggedIn = true; // Check if the user is logged in
  return isLoggedIn ? children : <Navigate to="/login" />;
}

<Route path="/dashboard" element={<PrivateRoute><Dashboard /></PrivateRoute>} />
```

404 Page

A wildcard route (*) catches all undefined paths and shows a "Page Not Found" component.

```
<Route path="*" element={<NotFound />} />
```

Lazy Loading

Lazy loading helps to load components only when needed, improving app performance by reducing the initial bundle size.

```
import { lazy, Suspense } from 'react';

const Home = lazy(() => import('./Home'));

<Route path="/" element={
  <Suspense fallback={<div>Loading...</div>}>
    <Home />
  </Suspense>
} />
```

React Forms

A React form is a way to collect user input using form elements like:

- ★ Text inputs
- ★ Checkboxes
- ★ Radio buttons
- ★ Dropdowns
- ★ Textareas

React controls form elements using **state** and **event handlers**. This makes it easy to track changes and respond to user input.

1. Controlled Components

In React, form elements are usually **controlled components**, meaning their value is controlled by React state.

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

function SimpleForm() {

 const [name, setName] = useState('');


 const handleChange = (e) => {

  setName(e.target.value);  // Update state when input changes

 };

 const handleSubmit = (e) => {

  e.preventDefault(); // Prevent page refresh

  alert(`Submitted Name: ${name}`);

 };

 return (

  <form onSubmit={handleSubmit}>

   <label>Enter your name:</label>

   <input type="text" value={name} onChange={handleChange} />

   <button type="submit">Submit</button>

  </form>

 );  }
```

- useState() is used to create a name state variable.
- The input field's value is linked to the state (value={name}).
- When the user types, the onChange event updates the state.
- On form submission, we handle the data and prevent the default page reload.

2. Handling Multiple Inputs

We can manage multiple form fields with one state object.

```
function MultiInputForm() {
  const [formData, setFormData] = useState({ name: '', email: '' });

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

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name: ${formData.name}, Email: ${formData.email}`);
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <input name="name" value={formData.name} onChange={handleChange} placeholder="Name" />
      <input name="email" value={formData.email} onChange={handleChange} placeholder="Email" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

3. Handling Checkboxes and Radio Buttons

Checkbox Example:

```
function CheckboxForm() {
  const [agreed, setAgreed] = useState(false);

  const handleCheckboxChange = () => {
    setAgreed(!agreed);
  };

  return (
    <form>
      <label>
        <input
```

```
        type="checkbox"
        checked={agreed}
        onChange={handleCheckboxChange}
      />
      I agree to terms
    </label>
  </form>
 );
}
```

Radio Button Example:

```
function RadioForm() {
 const [gender, setGender] = useState('male');

 return (
  <form>
   <label>
    <input type="radio" value="male"
     checked={gender === 'male'}
     onChange={(e) => setGender(e.target.value)}
    />
    Male
```

```
      </label>
      <label>
        <input
          type="radio"
          value="female"
          checked={gender === 'female'}
          onChange={(e) => setGender(e.target.value)}
        />
        Female
      </label>
    </form>
  );
}
```

4. Textarea Example

```
function TextareaForm() {
  const [message, setMessage] = useState('');
  return (
    <form>
      <label>Message:</label>
      <textarea value={message} onChange={(e) => setMessage(e.target.value)} />
    </form>
  );    }
```

- ❖ Controlled Inputs --> Inputs whose value is managed by React state.
- ❖ useState() --> Used to create and update state variables for form fields.
- ❖ onChange --> Updates the state when user types or selects input.
- ❖ onSubmit --> Handles form submission and prevents default reload.
- ❖ Checkbox/Radio --> Use checked instead of value, and handle onChange.
- ❖ Multiple Fields -->Manage using one state object with dynamic key/value pairs.

React Hooks

Hooks were added to React in version 16.8.

What is a Hook?

Hooks allow functional components to have access to state and other React features.

Hook Rules

There are 3 rules for hooks:

- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional

1. useState

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

useState accepts an initial state and returns two values:

- The current state.
- A function that updates the state.

Example:

```
import { useState } from "react";

import ReactDOM from "react-dom/client";

function FavoriteColor() {

  const [color, setColor] = useState("red");
```

```
  return <h1>My favorite color is {color}!</h1>
}
```

2. useEffect

The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

Example:

```
function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });
  return <h1>I've rendered {count} times!</h1>;
}
```

useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

Example

1. No dependency passed:

```
useEffect(() => {
  //Runs on every render
});
```

2. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```