REACT

# What is React?

* + **React** is a JavaScript library for building user interfaces, primarily single-page applications (SPA).
  + React allows developers to build reusable UI components and manage the state of these components efficiently.
  + **Declarative**: React allows you to describe the UI in a declarative way, which means you specify what the UI should look like for any given state, and React will update the UI automatically when the state changes.
  + **Component-Based**: React applications are composed of components that can be reused, nested, and combined.
  + **Virtual DOM**: React uses a Virtual DOM to optimize updates, ensuring efficient rendering.

# Setting Up a React Project

You can create a React app using create-react-app for an easy and quick setup. bash

Copy

npx create-react-app my-app cd my-app

npm start

# Components in React

Components are the building blocks of a React app. React provides two types of components:

## Class Components

* + Older style of defining components.
  + Can manage state and have lifecycle methods.

Example:

jsx Copy

import React, { Component } from 'react';

class MyClassComponent extends Component { constructor(props) {

super(props);

this.state = { counter: 0 };

}

increment = () => {

this.setState({ counter: this.state.counter + 1 });

};

render() { return (

<div>

<p>Counter: {this.state.counter}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

export default MyClassComponent;

## Functional Components

* + Simpler way of writing components.
  + Can manage state and side effects using **Hooks**.

Example:

jsx Copy

import React, { useState } from 'react';

function MyFunctionalComponent() {

const [counter, setCounter] = useState(0);

const increment = () => { setCounter(counter + 1);

};

return (

<div>

<p>Counter: {counter}</p>

<button onClick={increment}>Increment</button>

</div>

);

}

export default MyFunctionalComponent;

# Props

* + **Props (short for "properties")** are used to pass data from a parent component to a child component.
  + Props are **read-only** and cannot be modified by the child component.

Example:

jsx Copy

function Greeting(props) {

return <h1>Hello, {props.name}!</h1>;

}

function App() {

return <Greeting name="Alice" />;

}

# State

* + **State** is used to store dynamic data that can change over time.
  + Only **class components** (before hooks) and **functional components with hooks**

can have state.

* + useState is the hook used in functional components to manage state.

Example:

jsx Copy

import React, { useState } from 'react'; function Counter() {

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

const increment = () => setCount(count + 1); const decrement = () => setCount(count - 1);

return (

<div>

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

<button onClick={increment}>Increment</button>

<button onClick={decrement}>Decrement</button>

</div>

);

}

# Events

* + React uses **synthetic events**, which are wrappers around the native DOM events.
  + You can handle events like clicks, form submissions, etc., using event handlers.

Example:

jsx Copy

function MyButton() {

const handleClick = () => { alert('Button clicked!');

};

return <button onClick={handleClick}>Click Me</button>;

}

# Hooks

React Hooks allow functional components to have state, side effects, and more without converting them into class components.

## useState

* + useState is a hook that adds state to functional components.

jsx

Copy

const [state, setState] = useState(initialValue);

## useEffect

* + useEffect is a hook used to perform side effects in functional components (e.g., data fetching, subscriptions, DOM updates).
  + It runs after the first render and whenever dependencies change.

jsx Copy

useEffect(() => {

// side effect code here (e.g., fetch data)

}, [dependencies]); // dependency array

Example:

jsx Copy

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

function FetchData() {

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

useEffect(() => { fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => setData(data));

}, []); // Empty array means it runs only once (on mount)

return (

<div>

<h1>Fetched Data</h1>

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

</div>

);

}

## useContext

* + useContext is a hook used to access values from a context provider.

Example:

jsx Copy

const MyContext = React.createContext();

function MyComponent() {

const value = useContext(MyContext); return <div>{value}</div>;

}

function App() { return (

<MyContext.Provider value="Hello, Context!">

<MyComponent />

</MyContext.Provider>

);

}

## useRef

* + useRef is used to persist values across renders and access DOM elements directly.

Example:

jsx Copy

import React, { useRef } from 'react';

function FocusInput() { const inputRef = useRef();

const focusInput = () => { inputRef.current.focus();

};

return (

<div>

<input ref={inputRef} />

<button onClick={focusInput}>Focus Input</button>

</div>

);

}

**Use Memo**

* useMemo is a React Hook that **memoizes** the result of a computation, preventing unnecessary recalculations for performance optimization.

Example :

import React, { useState, useMemo } from "react";

function App() {

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

const [text, setText] = useState(" ");

// Memoized value

const expensiveCalculation = useMemo(() => {

console.log("Calculating...");

return count \* 2; // Simulating an expensive operation

}, [count]);

return (

<div>

<h1>Count: {count}</h1>

<h2>Expensive Value: {expensiveCalculation}</h2>

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

<input value={text} onChange={(e) => setText(e.target.value)}

placeholder="Type something" />

</div>

);

}

# React Router (For Navigation)

React Router enables navigation in React applications. It lets you handle multiple views (routes) and change the URL dynamically.

## Installation:

bash Copy

npm install react-router-dom

## Basic Example:

jsx Copy

import React from 'react';

import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';

function Home() {

return <h2>Home Page</h2>;

}

function About() {

return <h2>About Page</h2>;

}

function App() { return (

<Router>

<div>

<nav>

<Link to="/">Home</Link> |

<Link to="/about">About</Link>

</nav>

<Switch>

<Route exact path="/" component={Home} />

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

</Switch>

</div>

</Router>

);

}

export default App;

# Lifecycle Methods (Class Components)

Class components come with several lifecycle methods that allow you to hook into different stages of the component’s life, such as mounting, updating, and unmounting.

## Common Lifecycle Methods:

* + **componentDidMount()**: Called after the component is first rendered.
  + **componentDidUpdate(prevProps, prevState)**: Called after the component re-renders.
  + **componentWillUnmount()**: Called just before the component is removed from the DOM.

Example:

jsx Copy

class MyComponent extends React.Component { componentDidMount() {

console.log('Component mounted');

}

componentWillUnmount() { console.log('Component will unmount');

}

render() {

return <h1>Hello World</h1>;

}

}

# PropTypes (Type Checking)

You can use **PropTypes** to validate the types of props passed to a component. This can help you avoid bugs due to type mismatches.

bash

Copy

npm install prop-types

Example:

jsx Copy

import PropTypes from 'prop-types';

function MyComponent({ name, age }) {

return <div>{name} is {age} years old.</div>;

}

MyComponent.propTypes = {

name: PropTypes.string.isRequired, age: PropTypes.number.isRequired

};

# Styling in React

## Inline Styling:

You can use **inline styles** in React by passing a JavaScript object to the style attribute. Example:

jsx Copy

const divStyle = { color: 'blue', fontSize: '20px'

};

function MyComponent() {

return <div style={divStyle}>Styled Text</div>;

}

## CSS Modules:

You can also use **CSS Modules** to scope styles locally to a component. css

Copy

/\* MyComponent.module.css \*/

.title { color: red;

}

jsx Copy

import styles from './MyComponent.module.css';

function MyComponent() {

return <h1 className={styles.title}>Hello, World!</h1>;

}

# Conclusion

React is a powerful library for building dynamic and interactive user interfaces. By mastering key concepts like **components**, **state**, **props**, **hooks**,