React.js

# 1. React Components

**Definition:** Components are the building blocks of a React application. They let you split the UI into independent, reusable pieces.

**Types:**

* **Functional Components**: Plain JavaScript functions that return JSX.
* **Class Components**: ES6 classes that extend React.Component.

**Example:**

function Welcome() {

return <h1>Hello, React!</h1>;

}

**Advantages:**

* Reusability
* Separation of concerns

**Disadvantage:**

* Too many small components can make structure complex

# 2. Props (Properties)

**Definition:** Props are inputs to components. They are passed from parent to child and are read-only.

**Example:**

function Greet(props) {

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

}

<Greet name="John" />

**Advantage:** Allows dynamic content

**Disadvantage:** Cannot modify props inside child

# 3. State

**Definition:** State is used to store data that can change over time in a component.

**Example (with useState hook):**

import { useState } from 'react';

function Counter() {

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

return (

<div>

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

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

</div>

);

}

**Advantage:** Enables dynamic interactivity

**Disadvantage:** Can lead to performance issues if not managed properly

# 4. Hooks

**Definition:** Hooks let you use state and lifecycle features in functional components.

**Common Hooks:**

* useState() – for state management
* useEffect() – for side effects
* useRef(), useContext() etc.

**Example:**

import { useEffect, useState } from 'react';

function Example() {

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

useEffect(() => {

fetch('https://api.example.com')

.then(res => res.json())

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

}, []);

return <div>{JSON.stringify(data)}</div>;

}

**Advantage:** Simplifies functional components

**Disadvantage:** Can become complex with many hooks

# 5. Lifecycle Methods

**In Class Components:**

* componentDidMount(), componentDidUpdate(), componentWillUnmount()

**Functional Alternative:** Use useEffect()

**Example:**

useEffect(() => {

console.log("Component Mounted");

return () => console.log("Component Unmounted");

}, []);

**Advantage:** Helps manage setup and cleanup

**Disadvantage:** Improper use may cause bugs

# 6. Forms

**Definition:** React handles form elements through state.

**Example:**

function FormExample() {

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

const handleSubmit = (e) => {

e.preventDefault();

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

};

return (

<form onSubmit={handleSubmit}>

<input type="text" value={name} onChange={e => setName(e.target.value)} />

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

</form>

);

}

**Advantage:** Controlled inputs ensure predictable data

**Disadvantage:** More code compared to HTML forms

# 7. Event Handling

**Definition:** React uses camelCase for events and passes a function as the event handler.

**Example:**

function ClickButton() {

const handleClick = () => alert("Button Clicked");

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

}

**Advantage:** Easy to bind handlers

**Disadvantage:** Binding in class components can be tricky

# 8. Conditional Rendering

**Definition:** Render elements based on conditions.

**Example:**

{isLoggedIn ? <Dashboard /> : <Login />}

**Advantage:** Flexible UI

**Disadvantage:** Nested conditions can reduce readability

# 9. Routing

**Using React Router DOM**

**Setup:**

npm install react-router-dom

**Example:**

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

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

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

</Routes>

</BrowserRouter>

**Advantage:** Single Page App experience

**Disadvantage:** SEO limitations unless SSR is used

# 10. Storage (Local, Session, Cookies)

**LocalStorage:** Persistent until manually cleared

localStorage.setItem('user', 'John');

localStorage.getItem('user');

**SessionStorage:** Cleared when the tab is closed

sessionStorage.setItem('token', 'abc');

sessionStorage.getItem('token');

**Cookies:** Expire based on time

document.cookie = "user=John; expires=Fri, 31 Dec 2025 12:00:00 UTC";

**Advantage:** Stores user data on client-side

**Disadvantage:** Can be accessed and modified, so not for sensitive info

# 11. Fetching APIs

**Using Fetch with useEffect:**

useEffect(() => {

async function fetchData() {

const res = await fetch('https://api.example.com/data');

const json = await res.json();

setData(json);

}

fetchData();

}, []);

**Advantage:** Connects frontend with backend/API

**Disadvantage:** Needs error handling, async handling