Day 10: React Hooks - `useEffect` & Lifecycle Methods

Today, you'll learn about **React's lifecycle methods** and the **`useEffect` hook**, which helps manage **side effects** like API calls, event listeners, and timers.

---

1 What is `useEffect`?

`useEffect` is a **React Hook** that runs **side effects** in functional components, replacing lifecycle methods like:

- `componentDidMount` ✅ (Runs when the component **mounts**)
- `componentDidUpdate` ✅ (Runs when the component **updates**)
- `componentWillUnmount` ✅ (Runs when the component **unmounts**)

Syntax of `useEffect`

```
useEffect(() => {
  // Side effect code here
}, [dependencies]);
```

✅ **Runs the effect when dependencies change**
✅ **Runs once if dependency array `[]` is empty**

---

2 Using `useEffect` Without Dependencies (`componentDidMount`)

- Runs **only once** after the component is mounted.
- Useful for **fetching data** or **setting up subscriptions**.

Example: Log Message on Mount

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

function ExampleComponent() {
  useEffect(() => {
    console.log("Component Mounted!");

    // Cleanup function (optional)
    return () => console.log("Component Unmounted!");
  }, []); // Empty array means it runs only once

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

export default ExampleComponent;
```

✅ **Runs once when the component loads**
✅ **Logs "Component Mounted!" to the console**

3️⃣ Using `useEffect` with Dependencies (`componentDidUpdate`)

- Runs **whenever a specified state or prop changes**.

Example: Updating Title Dynamically

```jsx
import { useState, useEffect } from "react";

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

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]); // Runs when `count` changes

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase Count</button>
    </div>
  );
}

export default DynamicTitle;
```

✅ **Updates the page title whenever `count` changes**

4️⃣ Cleaning Up Effects (`componentWillUnmount`)

- Runs **before the component is removed from the DOM**.
- Useful for **removing event listeners** or **stopping timers**.

Example: Cleanup Timer on Unmount

```jsx
import { useState, useEffect } from "react";

function TimerComponent() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);

    return () => clearInterval(interval); // Cleanup function
  }, []);

  return <p>Timer: {seconds} seconds</p>;
}

export default TimerComponent;
```

✅ **Starts a timer when the component mounts**
✅ **Stops the timer when the component unmounts**

---

5 Fetching Data with `useEffect`

- **Common use case:** Fetching data from an API when the component loads.

Example: Fetching User Data

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

function FetchUser() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users/1")
      .then((response) => response.json())
      .then((data) => setUser(data));
  }, []);

  return (
    <div>
      {user ? <p>Name: {user.name}</p> : <p>Loading...</p>}
    </div>
  );
}

export default FetchUser;
```

✅ **Fetches user data from an API**
✅ **Displays "Loading..." while waiting for data**

---

6 Summary of Day 10

✔ **`useEffect`** runs side effects in functional components
✔ **Runs once** (`[]` as dependency array) → `componentDidMount`
✔ **Runs when dependencies change** (`[state]`) → `componentDidUpdate`
✔ **Cleans up effects** (`return function`) → `componentWillUnmount`
✔ **Used `useEffect` for API calls, timers, and event listeners**

🚀 Next Step: Day 11 - React Custom Hooks