📘 Day 11: `useEffect` Hook – Side Effects & API Calls

🕰️ What is `useEffect`?

- `useEffect` is a **React Hook** that lets you run **side effects** in your components.
- A **side effect** = anything that affects something outside of React's rendering cycle.

👉 Examples of side effects: - Fetching data from an API
- Setting up event listeners
- Updating the `document.title`
- Using timers (`setInterval`, `setTimeout`)

Without `useEffect`, React would re-run these tasks **every render**, causing bugs or infinite loops.

🕰️ Syntax

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

useEffect(() => {
  // Your side effect code (runs after render)

  return () => {
    // optional cleanup code (runs before component unmounts)
  };
}, [dependencies]);
```

- **First argument**: a function (your side effect)
- **Second argument**: dependency array
- `[]` → runs only once (on mount)
- `[state]` → runs when that state changes
- no array → runs on **every render**

🕰️ Example 1: Run on Every Render

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

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

  useEffect(() => {
```

```
    console.log("Component rendered!"); // runs after each render
  });

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

❤️Since there's no dependency array → effect runs after **every render**.

🕰️Example 2: Run Only Once (on Mount)

```
useEffect(() => {
  console.log("Component mounted!"); // runs only once after first render
}, []); // empty array → only on mount
```

🕰️Example 3: With Dependencies

```
useEffect(() => {
  console.log(`Count changed to: ${count}`); // runs when count changes
}, [count]); // dependency is count
```

🕰️Example 4: Cleanup Function

```
useEffect(() => {
  const timer = setInterval(() => {
    console.log("Timer running...");
  }, 1000);

  return () => {
    clearInterval(timer); // cleanup when component unmounts
    console.log("Timer cleaned up!");
  };
}, []);
```

❤️Cleanup is important for removing listeners, intervals, or memory leaks.

🕰️Example 5: API Call Basics

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

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

  useEffect(() => {
    // fetch() is a built-in JS function for HTTP requests
    fetch("https://jsonplaceholder.typicode.com/users")
      .then(res => res.json()) // parse response as JSON
      .then(data => setUsers(data)) // update state with fetched data
      .catch(err => console.error("Error fetching users:", err));
  }, []); // only run once on mount

  return (
    <div>
      <h2>User List</h2>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name} - {user.email}</li>
        ))}
      </ul>
    </div>
  );
}
```

❤️ `fetch()` is asynchronous, so the component won't freeze while waiting for data.

📝Exercise (15–20 min)

👉Build a component `JokeFetcher`:

1. Create a component with:
2. A `joke` state (empty string initially).
3. A button **"Get New Joke"**.
4. Use `useEffect` to:
5. Fetch a random joke from API → `https://official-joke-api.appspot.com/random_joke`
6. Save it in state and display setup + punchline.

7. When the button is clicked, fetch a new joke (update state).

👉With this, you'll practice: - `useEffect` basics
- Dependencies
- API fetching
- Handling user-triggered effects