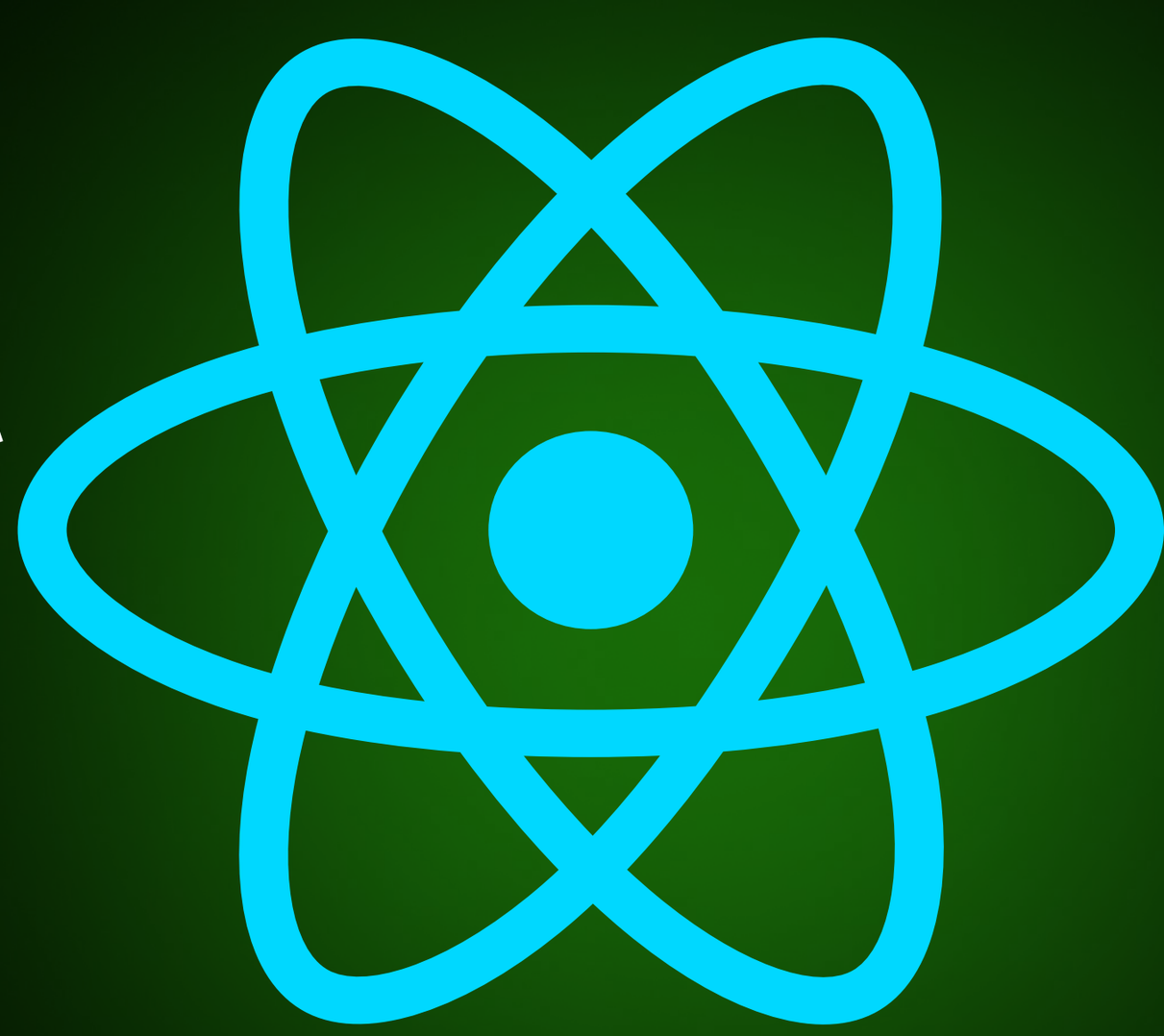
Mastering React's useEffect Hook

Lets Start!



Talha Ali
Full-Stack Developer

What is useEffect?

The **useEffect** hook is a built-in React hook that allows you to perform side effects in your functional components.

Side effects include operations like data fetching, subscriptions, or manually changing the DOM.

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Basic Syntax

App.jsx

```
useEffect(() => {}, [dependencies]);
```

The first argument is a function where you write your side effect.
eg: Refetch data

The second argument is an array of dependencies that dictate when the effect runs.

Every time the value in dependency array changes the function will re executed

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Running Effects on Mount and Unmount

If you want the effect to run only once when the component mounts, pass an empty dependency array:

App.jsx

```jsx
useEffect(() => {
  // eg. Fetch initial data on render
  return () => {
    // eg. Remove any event listeners
  };
}, []);
```

Mounting is when a component is first rendered and added to the DOM.
Unmounting is when a component is removed from the DOM, allowing it to perform cleanup tasks.

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Effects with Dependencies

When you include dependencies in the array, the effect runs whenever any of those dependencies change:

App.jsx

```jsx
useEffect(() => {
  // Runs on mount and whenever `count` changes
}, [count]);
```

The useEffect will execute this function every time the value of count changes

Talha Ali
Full-Stack Developer

Cleanup Function

You can return a cleanup function from your effect, which React calls when the component unmounts or before re-running the effect:

App.jsx

```jsx
useEffect(() => {
  // Set a timer to execute after 1 second
  const timer = setTimeout(() => {
    console.log("Timer expired"); // Log message on timer expiration
  }, 1000);

  // Cleanup function to clear the timer on unmount
  return () => clearTimeout(timer);
}, []); // Runs only once on mount
```

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Multiple Effects

You can use multiple useEffect hooks in a single component to separate different side effects:

App.jsx

```jsx
useEffect(() => {
  // Effect 1
}, []);

useEffect(() => {
  // Effect 2
}, [dependency]);
```

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Common Use Cases

Fetching data from APIs

Setting up subscriptions or timers

Manually changing the DOM (e.g., focus on an input)

Running cleanup tasks

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Project to practice

Overview:

Fetches and displays posts from the JSON Placeholder API using React, highlighting the useEffect hook for data fetching.

Key Features:

- **Data Fetching:** Retrieves posts from https://jsonplaceholder.typicode.com/posts.
- **Loading State:** Shows a loading message while fetching.
- **Error Handling**: Displays error messages if the fetch fails.
- **Dynamic Rendering**: Lists post titles and content.

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Example Snippet

App.jsx

```
useEffect(() => {
  const fetchPosts = async () => {
    const response = await
fetch('https://jsonplaceholder.typicode.com/posts');
    const data = await response.json();
    setPosts(data);
  };
  fetchPosts();
}, []); // Runs once on mount
```

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Wrapping Up

The useEffect hook is powerful for managing side effects in your functional components. Remember to specify dependencies carefully to avoid unnecessary renders or missed updates.

Talha Ali
Full-Stack Developer

Mastering React hooks part 2

Want to learn more about **React** hooks and **state management**?

Talha Ali

Full-Stack Developer

Follow for more tips on mastering **React**!