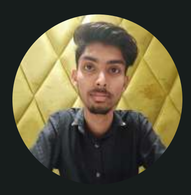
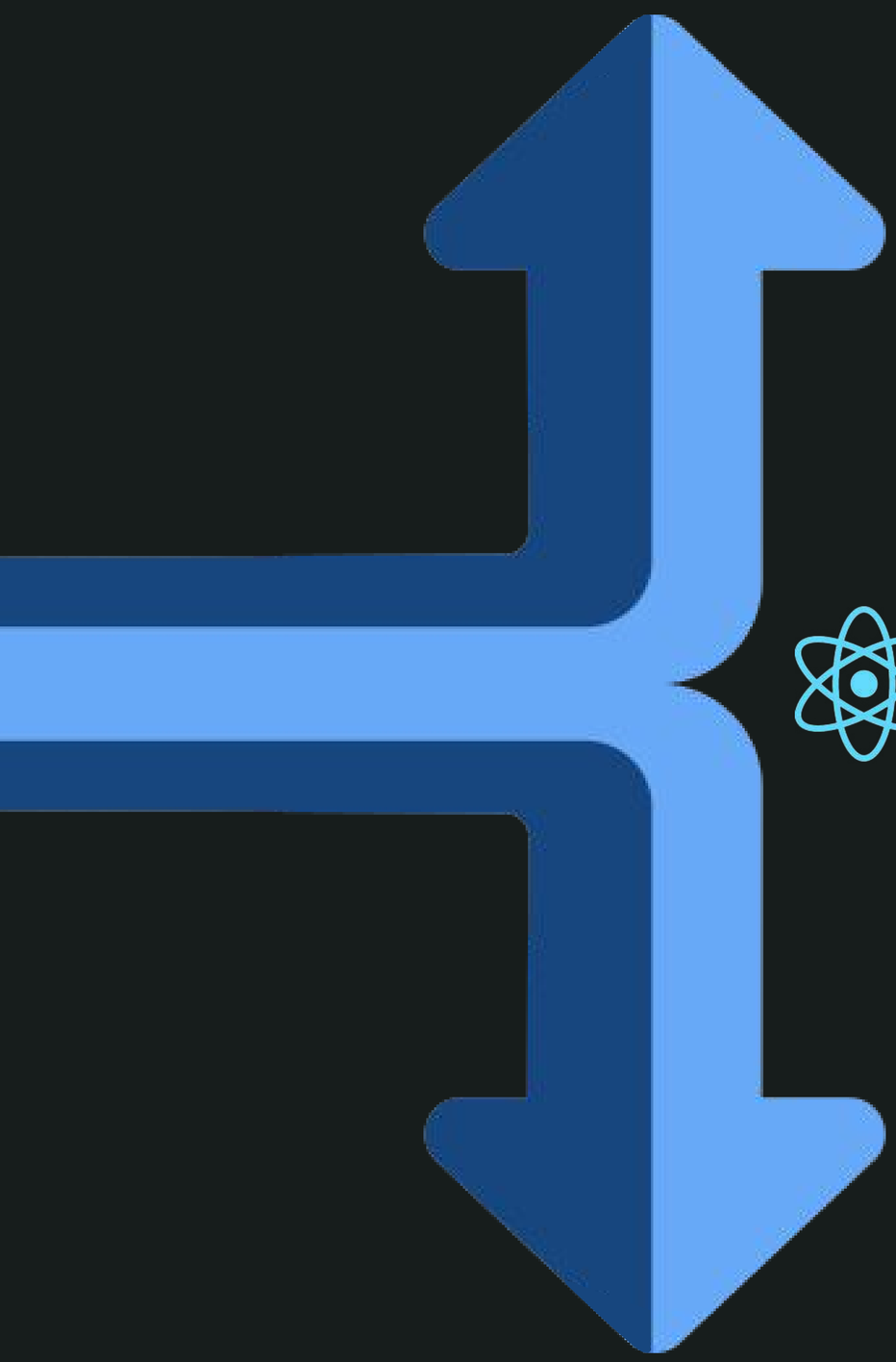


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

useState()



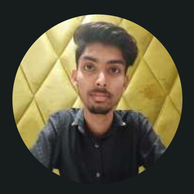
What they are?

REACT HOOKS

Where you use them?

useEffect()

Created By
Gagan Saini



Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

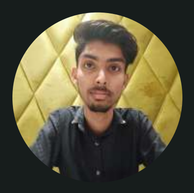
THE STATE SAVIOR

useState()

React, **useState()** acts as the gateway to **state management**, allowing components to hold and **update** their **own state internally**.

If you want to update the **UI in real-time**?
You got it! useState() is for you.

useState() **triggers re-renders** whenever the state **changes**, keeping your **components dynamic**.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Why you need useState()?

Empowerment

↳ Gives components autonomy to **manage their state**, reducing reliance on external data sources.

Efficiency

↳ Enables **lightning-fast updates** & **renders** keeping your app's performance in top shape.

Simplicity

↳ Simplifies state handling, making your **codebase cleaner** and more **manageable**.

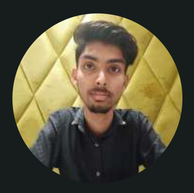
Code

state.jsx

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

const LightSwitch = () => {
  const [isOn, setIsOn] = useState(false);
  const toggleLight = () => setIsOn(!isOn);

  return (
    <div>
      <p>Light: {isOn ? 'On' : 'Off'}</p>
      <button onClick={toggleLight}>
        {isOn ? 'Turn Off' : 'Turn On'}
      </button>
    </div>
  );
}
```


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

THE SIDE EFFECT SPECIALIST

useEffect()

useEffect() is a powerful React Hook that allows you to perform **side effects** in your functional components.

useEffect() handles these **tasks outside the render phase**, keeping your components clean and efficient.

Why you need useEffect()?

Action After Render

↳ Perform actions after renders, like **fetching data** or setting up subscriptions.

Clean-up Tasks

↳ Clean up after themselves! useEffect() can return a **cleanup function** to **prevent memory leaks** in your component.

Gagan Saini
@gagan-saini-gs

Dependency Array

By **default**, it runs on **every render**. But with the dependency array, you can **control** when you want to **re-render**.

Tell useEffect exactly what to watch. Include **state variables** or **props** in the array to trigger the effect only when they change.

Beware of **infinite loops**! If a dependency in the array causes a **re-render**, which in turn **triggers the useEffect()** that updates the dependency you've got a problem.

Code

effect.jsx

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

const CatFact = () => {
  const [fact, setFact] = useState('');

  useEffect(() => {
    fetch('https://catfact.ninja/fact')
      .then(response => response.json())
      .then(data => setFact(data.fact));
  }, []);

  return (
    <div>{fact}</div>
  );
}
```

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like Comment Repost