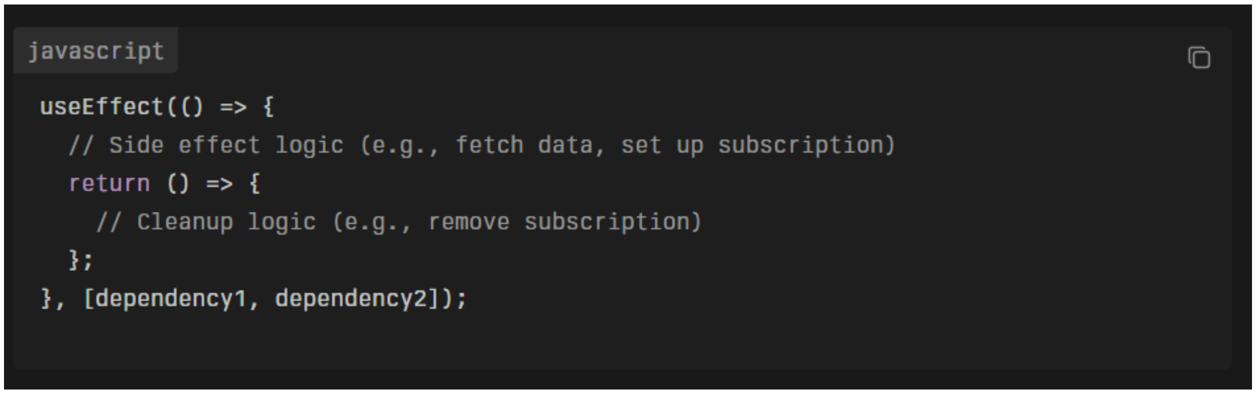
What is the useEffect in React?

useEffect is a React Hook that lets you perform side effects in function components. Side effects are operations that interact with the outside world or systems outside of React's rendering process.



javascript

```javascript
useEffect(() => {
  // Side effect logic (e.g., fetch data, set up subscription)
  return () => {
    // Cleanup logic (e.g., remove subscription)
  };
}, [dependency1, dependency2]);
```

Side Effect

In React, a side effect is any operation that interacts with the world outside the scope of a React component's rendering process. This includes actions that do not directly relate to rendering the user interface but are necessary for the application to function correctly

Strict Mode

Strict Mode in React is a development-only tool designed to help developers identify potential problems in their applications by enabling extra checks and warnings for its child components.

What is a Dependency in `useEffect`?

The **dependency** is basically a list of variables (state or props) that `useEffect` watches. When any variable in that list **changes**, React runs the effect function again.

Why is it important?

- If you put variables in the dependency array, your effect **reacts** to changes in those variables.

- If the dependency array is empty `[]`, the effect runs **only once** after the initial render.
- If you don't specify dependencies at all, the effect runs **after every render** (which might cause performance issues or bugs).

What is the cleanup function?

- It's a **function you return inside your `useEffect` callback**.
- React calls this cleanup function **when the component is about to unmount** or **before running the effect again** (if dependencies changed).

Why do you need cleanup?

Because some side effects create things that need to be cleaned up to avoid bugs or memory leaks. For example:

- Removing event listeners
- Clearing timers (`setTimeout` or `setInterval`)
- Canceling subscriptions or API calls

Code 1:

```
import React, { useState, useEffect } from 'react';

function App3() {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(0);

  // This effect runs ONLY when countA changes
  useEffect(() => {
    alert(`countA changed to ${countA}`);
  }, [countA]); // <-- Dependency only on countA

  return (
    <div>
      <h1>useEffect Dependency Demo</h1>

      <button onClick={() => setCountA(prev => prev + 1)}>
        Increment Count A ({countA})
      </button>
      <br /><br />
      <button onClick={() => setCountB(prev => prev + 1)}>
        Increment Count B ({countB})
      </button>
    </div>
  );
}

export default App3;
```

Code 2:

```
import { useState } from "react";
import EX from "./ex";
function App2(){
    const [count,setCount] = useState(1);
    return(
        <>
    {count && <EX/>}
    <button onClick={()=>setCount(0)}>Stop</button>
        </>
    );
}
export default App2;

–------------------------------------------------------------------

import { useEffect } from "react"

function EX()
{
     useEffect(()=>{
        let interval=setInterval(()=>{
            console.log("hello i am interval")
        },1000)
        return ()=>{
                console.log("component unmounted")
                clearInterval(interval)
            }
    },[])
    return(
        <h1>This is example</h1>
    )
}
export default EX
```

Code 3:

```
import { useEffect, useState } from 'react';
import './App1.css'

function ProductList() {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://fakestoreapi.com/products')
      .then(res => res.json())
      .then(data => {
        setProducts(data);
        setLoading(false);
      });
  }, []);

  if (loading) return <p className='loading'>Loading...</p>;

  return (
    <div className='product-list'>
      {products.map(product => (
        <div key={product.id} className='product-card'>
          <h3 className='product-title'>{product.title}</h3>
          <p className='product-price'>${product.price}</p>
          <img src={product.image} alt={product.title} width="100"
className='product-image'/>
        </div>
      ))}
    </div>
  );
}

export default ProductList;
```

```
/* Loader styling */
.loading {
    font-size: 1.5rem;
    text-align: center;
    margin-top: 2rem;
```

```css
  color: #555;
}

/* Container for all products */
.product-list {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(220px, 1fr));
  gap: 1.5rem;
  padding: 1rem;
}

/* Individual product card */
.product-card {
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 1rem;
  background-color: #fff;
  text-align: center;
  box-shadow: 0 2px 6px rgb(0 0 0 / 0.1);
  transition: box-shadow 0.3s ease;
}

.product-card:hover {
  box-shadow: 0 4px 12px rgb(0 0 0 / 0.2);
}

/* Product image */
.product-image {
  max-width: 100%;
  height: 150px;
  object-fit: contain;
  margin-bottom: 1rem;
}

/* Product title */
.product-title {
  font-size: 1rem;
  font-weight: 600;
  margin-bottom: 0.5rem;
  color: #333;
}

/* Product price */
.product-price {
  font-size: 1.1rem;
  font-weight: bold;
  color: #0077cc;
}
```

Why Should we use useEffect for fetch?

React Components Should Be Pure
Control When the Fetch Happens
Prevents Infinite Loops
Cleanups Are Easier

Extra code 1:

```
import React, { useEffect, useState } from 'react';

function ImageRepeater() {
  const [imageList, setImageList] = useState([]);
  const [isRunning, setIsRunning] = useState(true);

  const repeatingImageUrl =
"https://media1.tenor.com/m/aJRliinjGkEAAAAC/prabhas-rebel-star.gif";
  // const repeatingImageUrl =
"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSDyPvo0QHapnZ8e_2Rf
eCPTYIXXEsI50jEyA&s";
  const stopImageUrl =
"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTDsJJ9iVbmcwu8dlvSi
E-06TAekmbtwlCR3A&s";

  useEffect(() => {
    let intervalId;

    if (isRunning) {
      intervalId = setInterval(() => {
        setImageList(prev => [...prev, repeatingImageUrl]);
      }, 1000);
    }

    return () => {
      clearInterval(intervalId);
    };
  }, [isRunning]);

  const handleStopClick = () => {
    setIsRunning(false); // triggers cleanup to stop repeating
  };

  return (
    <div>
      <h2>Click the second image to stop repeating</h2>

      {/* The stop image */}
      <img
        src={stopImageUrl}
        alt="Click to stop"
        onClick={handleStopClick}
        style={{ cursor: 'pointer', marginBottom: '20px' }}
      />

      {/* Repeating images container */}
      <div style={{ display: 'flex', flexWrap: 'wrap' }}>
        {imageList.map((src, index) => (
          <img
            key={index}
            src={src}
            alt={`repeating-${index}`}
            style={{ width: '200px', height: 'auto', margin: '5px' }}
          />
        ))}
      </div>
    </div>
  );
}

export default ImageRepeater;
```

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

function Clock(){
  const [time,setTime] = useState(new Date().toLocaleTimeString());
  const [isRunning, setIsRunning] = useState(true);

  useEffect(()=>{
    let intervalId;
    if(isRunning){
      intervalId = setInterval(()=>{
        setTime(new Date().toLocaleTimeString());
      },1000);
    }

    return() => {
      clearInterval(intervalId);
    };
  },[isRunning]);

  const handleStop = () => {
    setIsRunning(false);
  };

  return(
    <>
      <h2>Current Time</h2>
      <p style={{fontSize:'24px'}}>{time}</p>
      <button onClick={handleStop}>Stop</button>
    </>
  );
}

// export default Clock;
```