* .

Code:-

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

const CombinedExample = () => {

const [userData, setUserData] = useState(null); // State for user data

const inputRef = useRef(null);

const username = 'octocat'; // GitHub username to fetch

useEffect(() => {

const fetchData = async () => {

try {

const response = await fetch(`https://api.github.com/users/${username}`);

if (!response.ok) {

throw new Error('Network response was not ok');

}

const result = await response.json();

setUserData(result);

} catch (error) {

console.error('Error fetching user data:', error);

}

};

fetchData();

// Focus on input after data is fetched

inputRef.current.focus();

}, []);

return (

<div>

<h1>GitHub User Details</h1>

<input type="text" ref={inputRef} placeholder="Type here..." />

{userData ? (

<div>

<h2>{userData.name}</h2>

<p><strong>Username:</strong> {userData.login}</p>

<p><strong>Bio:</strong> {userData.bio}</p>

<p><strong>Public Repos:</strong> {userData.public\_repos}</p>

<p><strong>Followers:</strong> {userData.followers}</p>

<p><strong>Following:</strong> {userData.following}</p>

<img src={userData.avatar\_url} alt={`${userData.login}'s avatar`} width="100" />

<p><a href={userData.html\_url} target="\_blank" rel="noopener noreferrer">View Profile</a></p>

</div>

) : (

<p>Loading user data...</p>

)}

</div>

);

};

export default CombinedExample;

**1. useEffect Hook**

**Overview**

* The useEffect hook is used for managing side effects in functional components.
* Side effects can include data fetching, subscriptions, or manually changing the DOM.
* It runs after every render, but you can control when it runs using dependency arrays.

**Syntax**

javascript

Copy code

useEffect(() => {

// Code to run after render (side effects)

return () => {

// Cleanup code (optional)

};

}, [dependencies]);

**2. useRef Hook**

**Overview**

* The useRef hook is primarily used to create a mutable object that persists for the full lifetime of the component.
* It can hold a reference to a DOM element or store any mutable value that does not trigger a re-render when changed.

**Syntax**

javascript

Copy code

const ref = useRef(initialValue);

**Key Concepts**

* **Accessing DOM Elements**: When you assign a ref to a DOM element, you can access it directly via ref.current.
* **Mutable Value**: You can store any value in a ref, and it won't cause a re-render when the value changes.
* **Initial Value**: The argument passed to useRef sets the initial value of the current property of the ref.

**Key Concepts**

* **Dependency Array**: The second argument is an array that determines when the effect should run:
  + **Empty Array ([])**: Runs only once after the initial render (similar to componentDidMount).
  + **With Dependencies**: Runs after every render when any of the dependencies change.

**Cleanup Function**: If your effect creates subscriptions or timers, return a function that cleans them up to avoid memory leaks or unintended behavior