CUSTOM HOOKS

What is a Custom Hook, and why do we use it?

A Custom Hook is a reusable JavaScript function that uses one or more built-in React hooks (like useState, useEffect, etc.) to share logic between components.

We use them to:

- Avoid code duplication
- Keep components cleaner
- Improve reusability and testability

RULES YOU MUST FOLLOW WHEN CREATING CUSTOM HOOKS

- The hook’s name must start with use (e.g., useFetch, useThrottle).
- Hooks must be called only at the top level (not inside loops, conditions, or nested functions).
- Hooks can only be called from React components or other custom hooks — not regular JS functions.

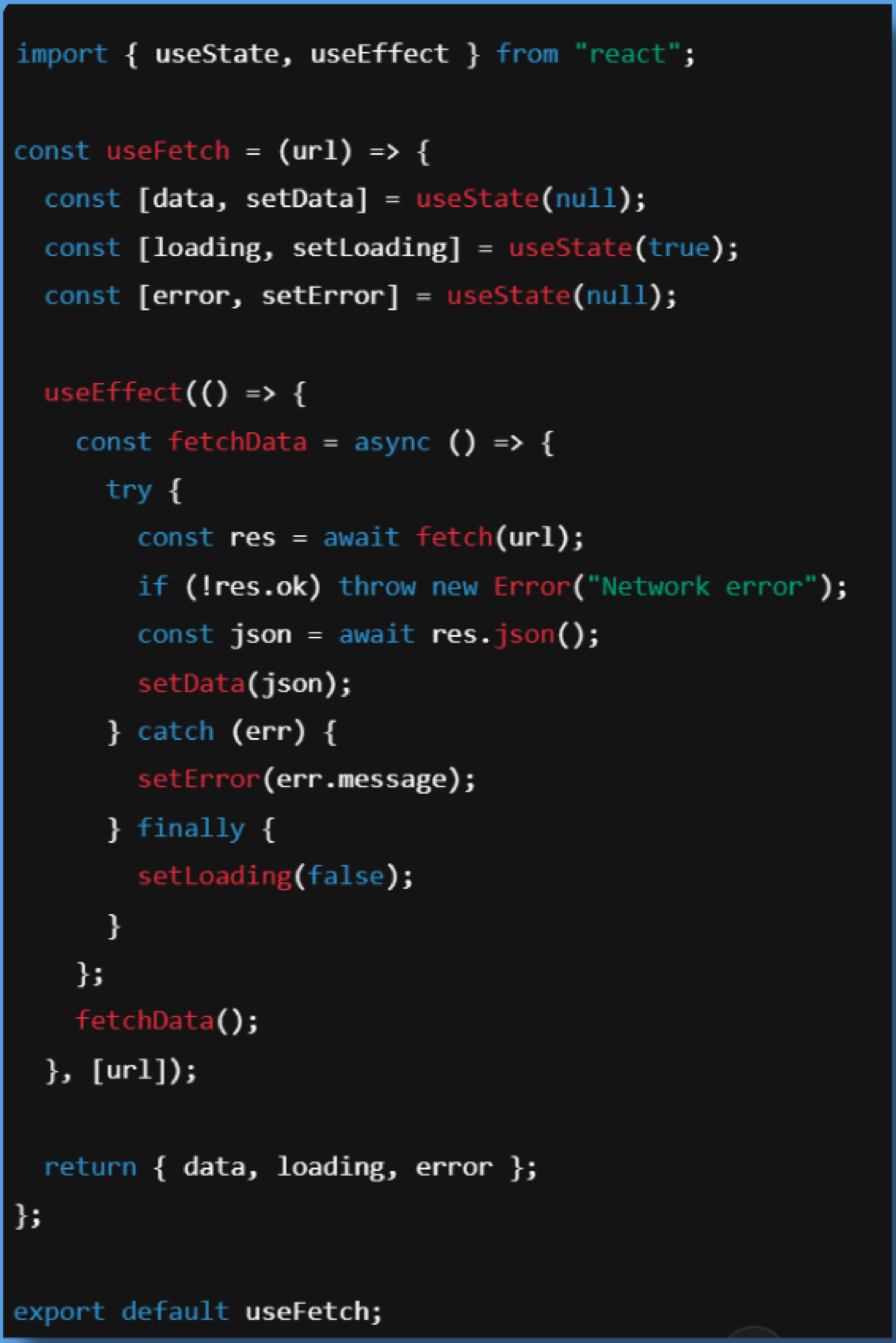
1 You have repeated logic for fetching API data in multiple components. What's the best way to reuse this logic?

2 You want to debounce a search input so that the API call happens only after the user stops typing. Which React feature would you use to implement this behavior cleanly?

3 You need to track window width across multiple components for responsive design without duplicating code. What should you create?

4 You want to store and retrieve user preferences from localStorage and share that logic across components. What's the ideal React solution?

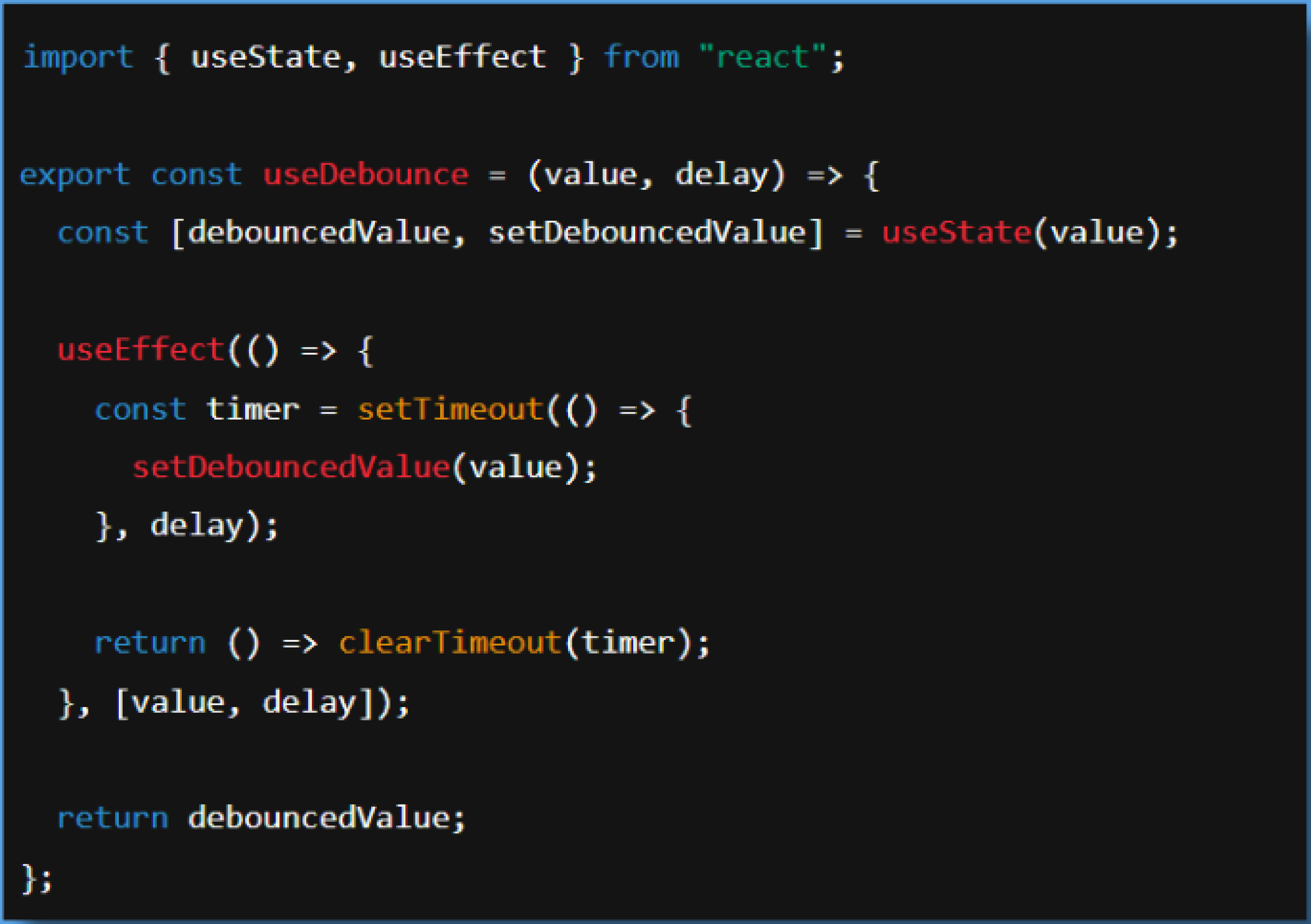
useFetch Hook (Reusable API Fetcher)



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

const useFetch = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch(url);
        if (!res.ok) throw new Error("Network error");
        const json = await res.json();
        setData(json);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };
    fetchData();
  }, [url]);

  return { data, loading, error };
};

export default useFetch;
```

const { data, loading, error } = useFetch("https://api.github.com/users");

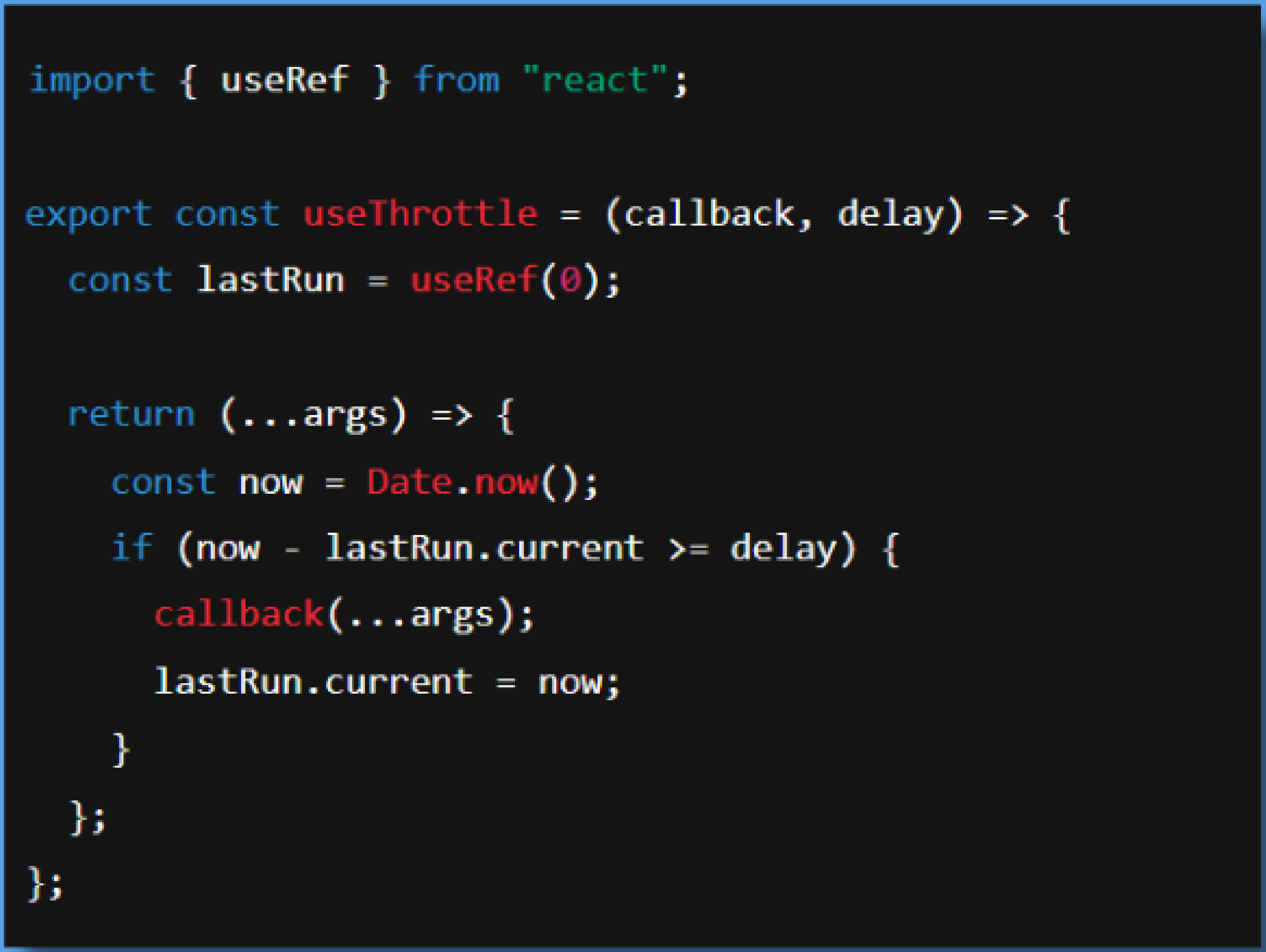
useDebounce Hook



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

export const useDebounce = (value, delay) => {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const timer = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => clearTimeout(timer);
  }, [value, delay]);

  return debouncedValue;
};
```

const debouncedSearch = useDebounce(searchQuery, 500);

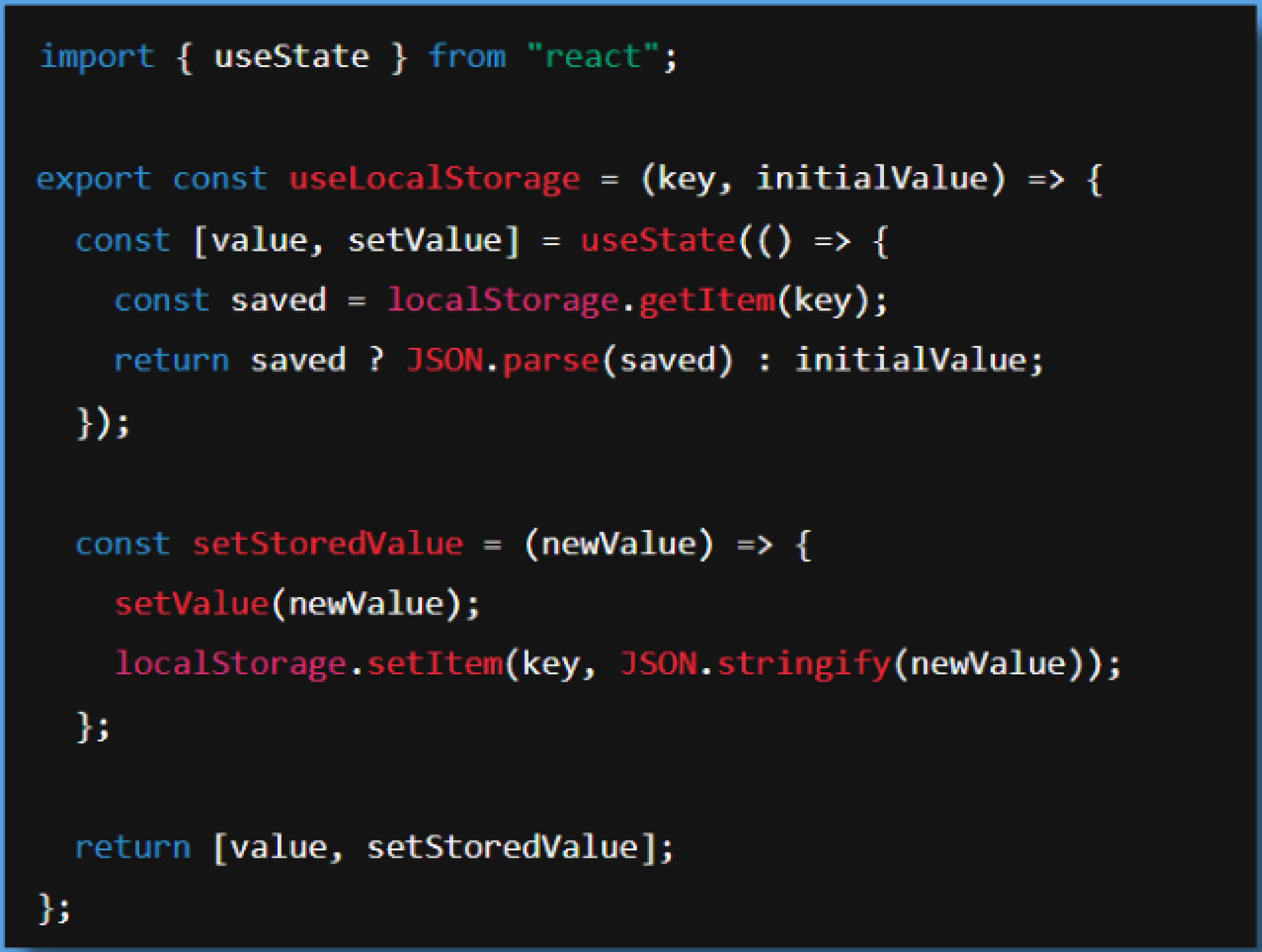
useThrottle Hook



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

export const useThrottle = (callback, delay) => {
  const lastRun = useRef(0);

  return (...args) => {
    const now = Date.now();
    if (now - lastRun.current >= delay) {
      callback(...args);
      lastRun.current = now;
    }
  };
};
```

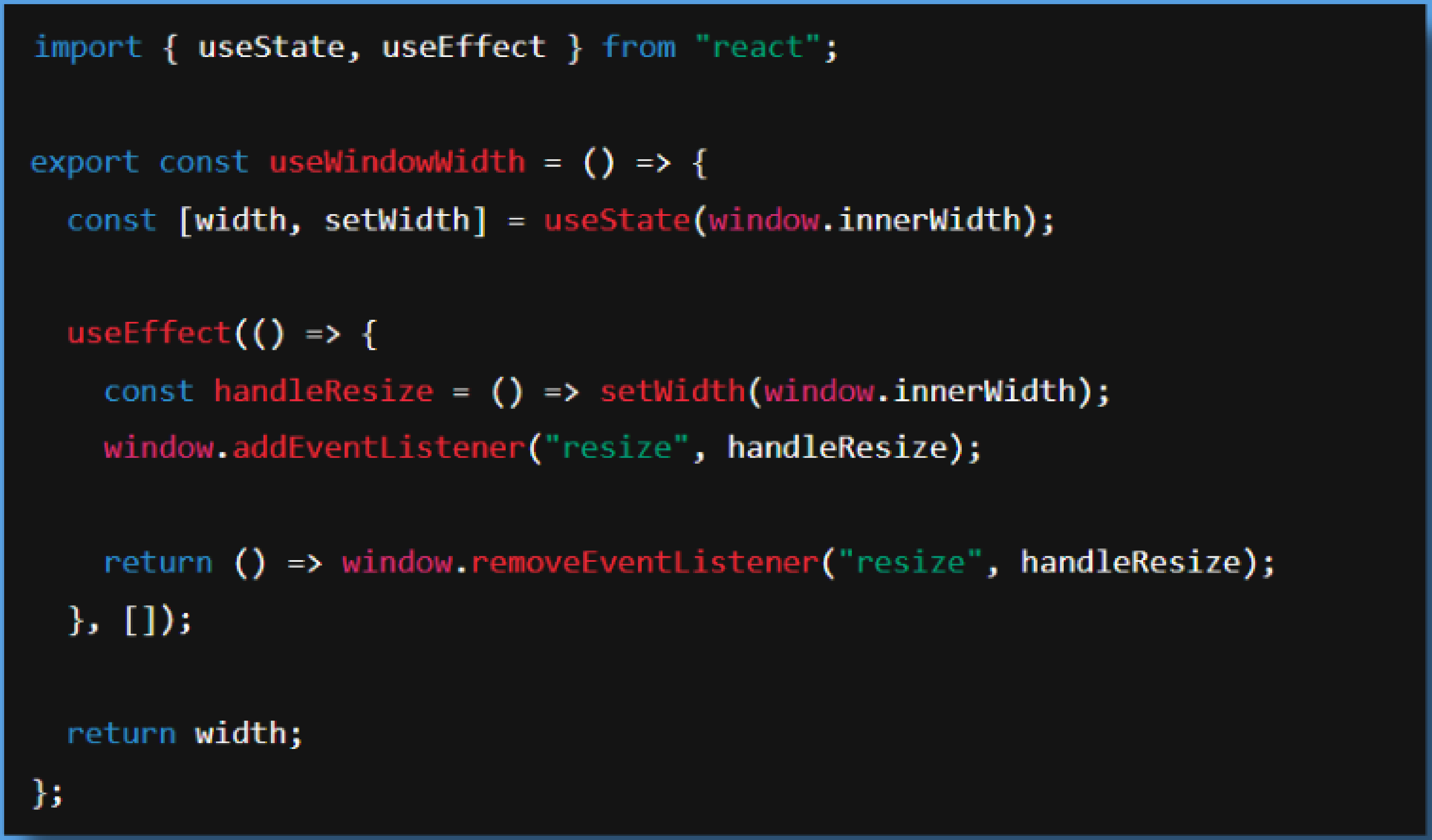
```
const handleScroll = useThrottle(() => {
console.log("Scrolled!");
}, 2000);
```

useLocalStorage Hook


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

export const useLocalStorage = (key, initialValue) => {
  const [value, setValue] = useState(() => {
    const saved = localStorage.getItem(key);
    return saved ? JSON.parse(saved) : initialValue;
  });

  const setStoredValue = (newValue) => {
    setValue(newValue);
    localStorage.setItem(key, JSON.stringify(newValue));
  };

  return [value, setStoredValue];
};
```

const [theme, setTheme] = useLocalStorage("theme", "light");

useWindowWidth Hook



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

export const useWindowWidth = () => {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener("resize", handleResize);

    return () => window.removeEventListener("resize", handleResize);
  }, []);

  return width;
};
```

const width = useWindowWidth();

Hook	Purpose
useFetch()	Fetch data from an API
useDebounce()	Delay function calls (typing, search)
useThrottle()	Limit function execution rate
useLocalStorage()	Persist state between reloads
useWindowWidth()	Track screen size for responsive UI