Advanced Hooks

Advanced hooks are a set of hooks in React that allow developers to handle more complex state and side effect logic. These include hooks like useReducer, useMemo, useCallback, useRef, and custom hooks. These hooks provide more control and flexibility for managing state and lifecycle events in functional components.

useReducer for Complex State Management

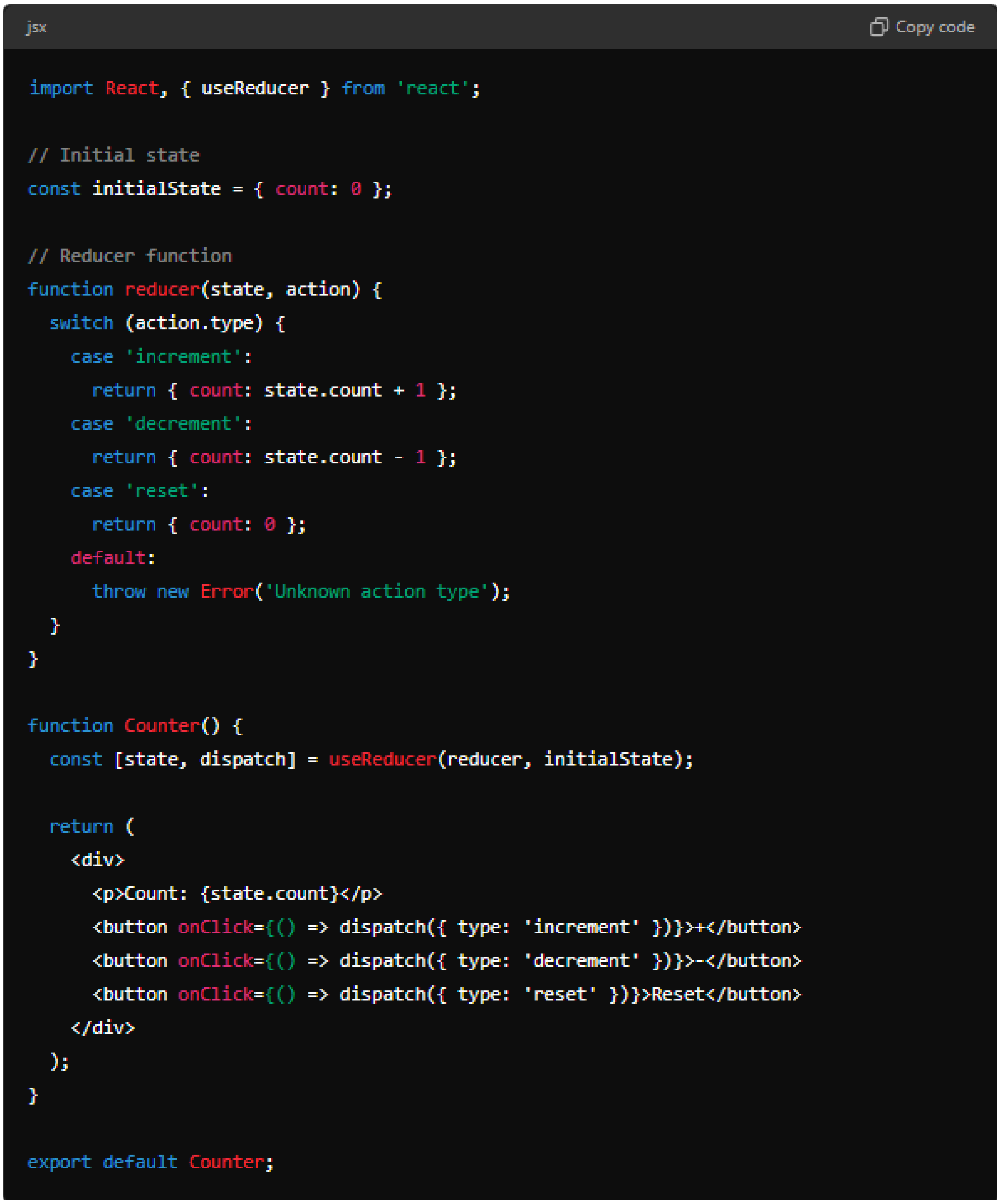
Definition: useReducer is a hook that is used for managing complex state logic in React. It's an alternative to useState and is particularly useful when you have a complex state object or when the next state depends on the previous one.

useReducer(<reducer>, <initialState>)

The **reducer** function contains your custom state logic and the **initialStatecan** be a simple value but generally will contain an object.

Example:

Imagine you have a counter application where you can increment, decrement, and reset the count. Using useReducer, we can manage these actions more efficiently.



jsx	Copy code

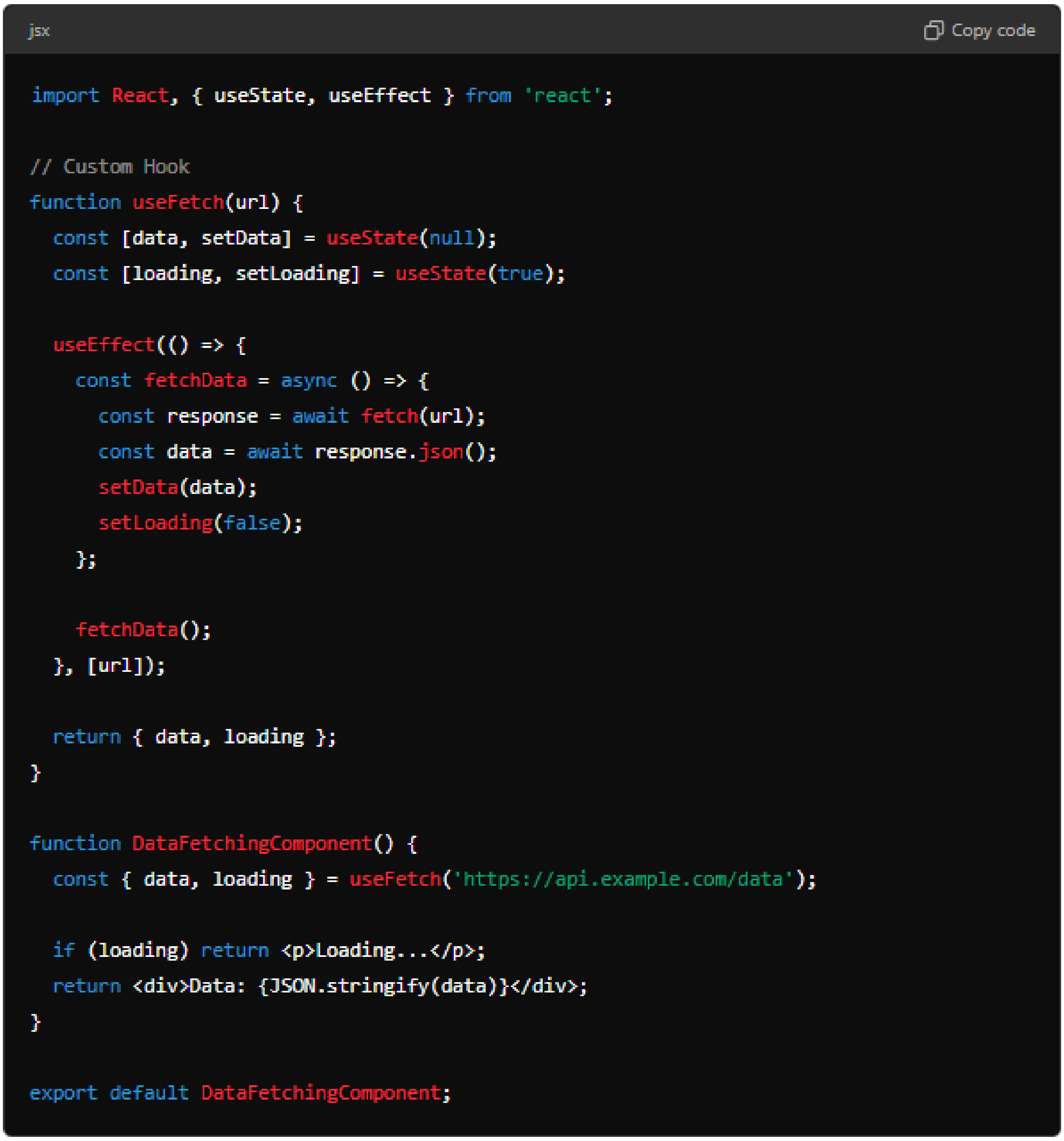
```jsx
import React, { useReducer } from 'react';

// Initial state
const initialState = { count: 0 };

// Reducer function
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    case 'reset':
      return { count: 0 };
    default:
      throw new Error('Unknown action type');
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
      <button onClick={() => dispatch({ type: 'reset' })}>Reset</button>
    </div>
  );
}

export default Counter;
```

Custom Hooks

Definition: Custom hooks are JavaScript functions that allow you to reuse stateful logic across multiple components. They start with the word "use" and can call other hooks.

Example:

Let's create a custom hook useFetch that fetches data from an API.



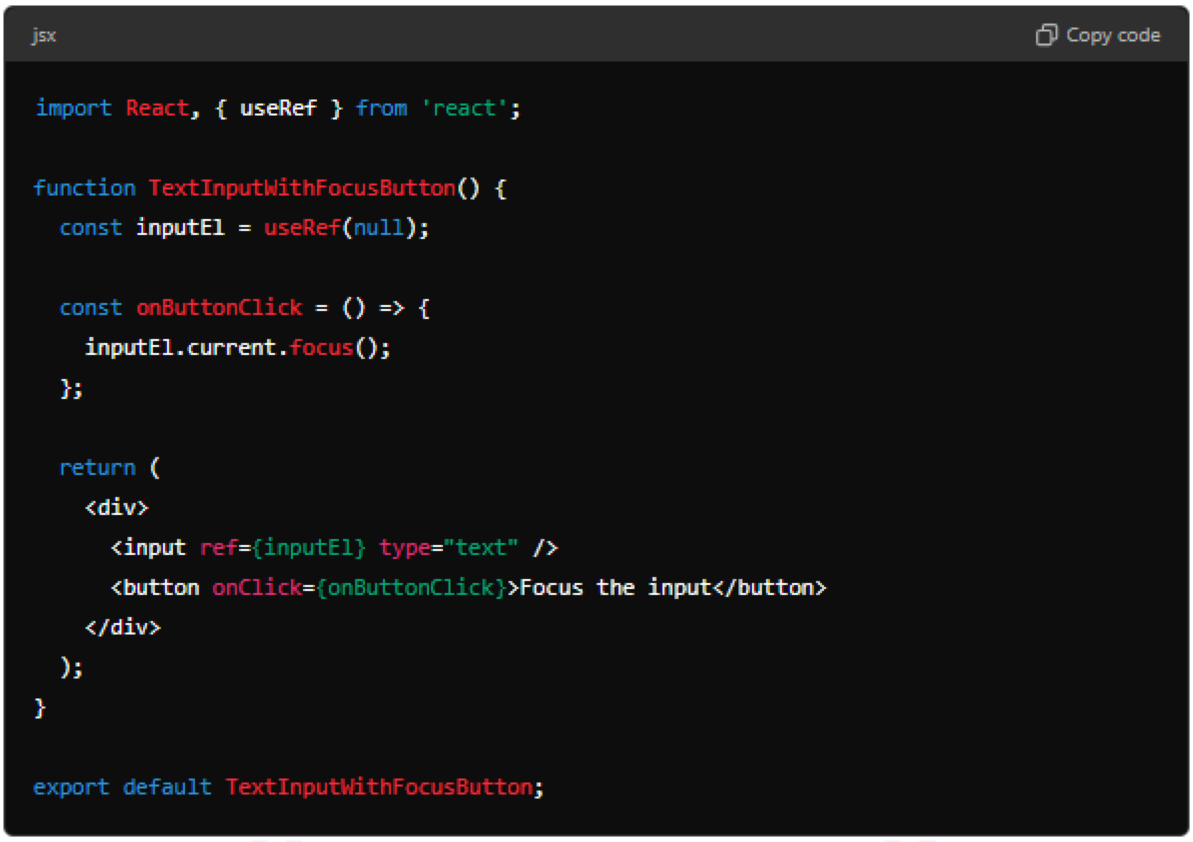
```jsx
import React, { useState, useEffect } from 'react';

// Custom Hook
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      const response = await fetch(url);
      const data = await response.json();
      setData(data);
      setLoading(false);
    };

    fetchData();
  }, [url]);

  return { data, loading };
}

function DataFetchingComponent() {
  const { data, loading } = useFetch('https://api.example.com/data');

  if (loading) return <p>Loading...</p>;
  return <div>Data: {JSON.stringify(data)}</div>;
}

export default DataFetchingComponent;
```

useRef for DOM Manipulation

Definition: useRef is a hook that returns a mutable ref object whose .current property is initialized to the passed argument (initialValue). It can be used to persist a value across renders or to directly access a DOM element.

Example:

Consider a text input that you want to focus on when a button is clicked.



```jsx
import React, { useRef } from 'react';

function TextInputWithFocusButton() {
  const inputEl = useRef(null);

  const onButtonClick = () => {
    inputEl.current.focus();
  };

  return (
    <div>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </div>
  );
}

export default TextInputWithFocusButton;
```

Summary

- **Advanced Hooks**: A set of hooks in React for handling complex state and side effects.
- **useReducer**: Manages complex state logic with a reducer function.
- **Custom Hooks**: Allows reuse of stateful logic across components.
- **useRef**: Accesses and manipulates DOM elements directly.