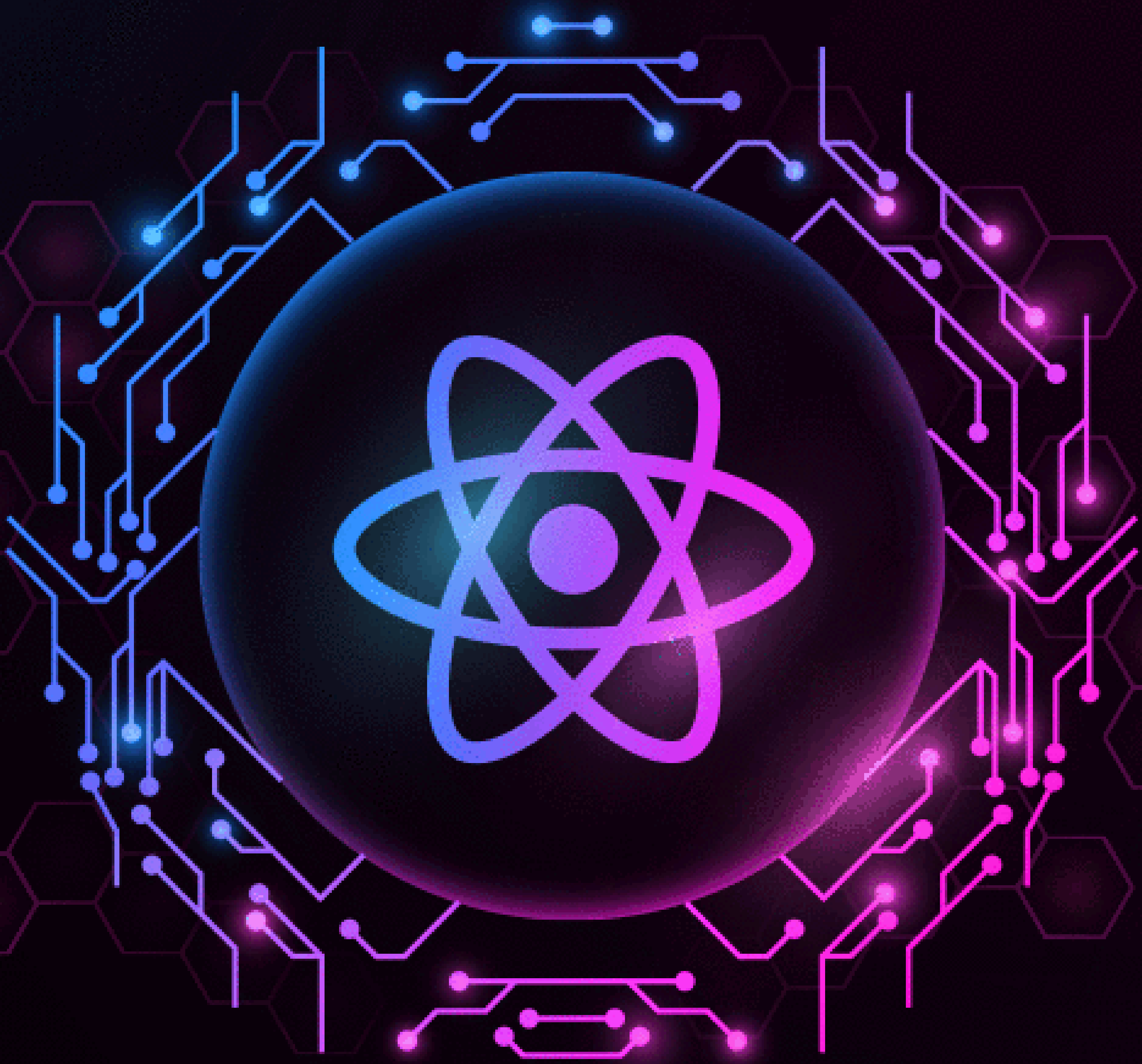


12 Key Techniques for Optimizing Your React Application

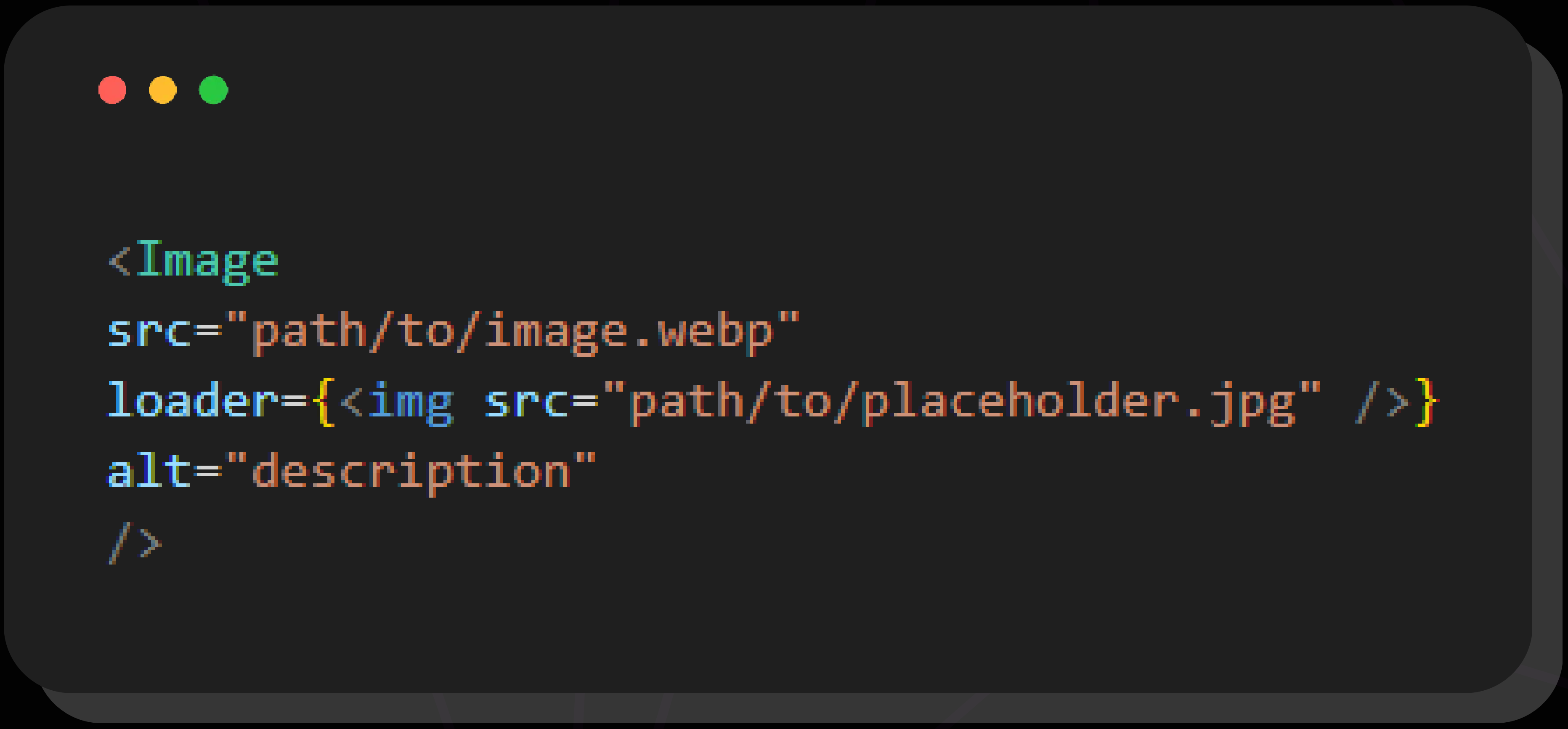
01

Image Optimization

Explanation: Optimizing images can significantly reduce the load time of your application.

Implementation:

- Use modern image formats (e.g., WebP) and tools for compressing images.
- Serve appropriately sized images based on the user's device.



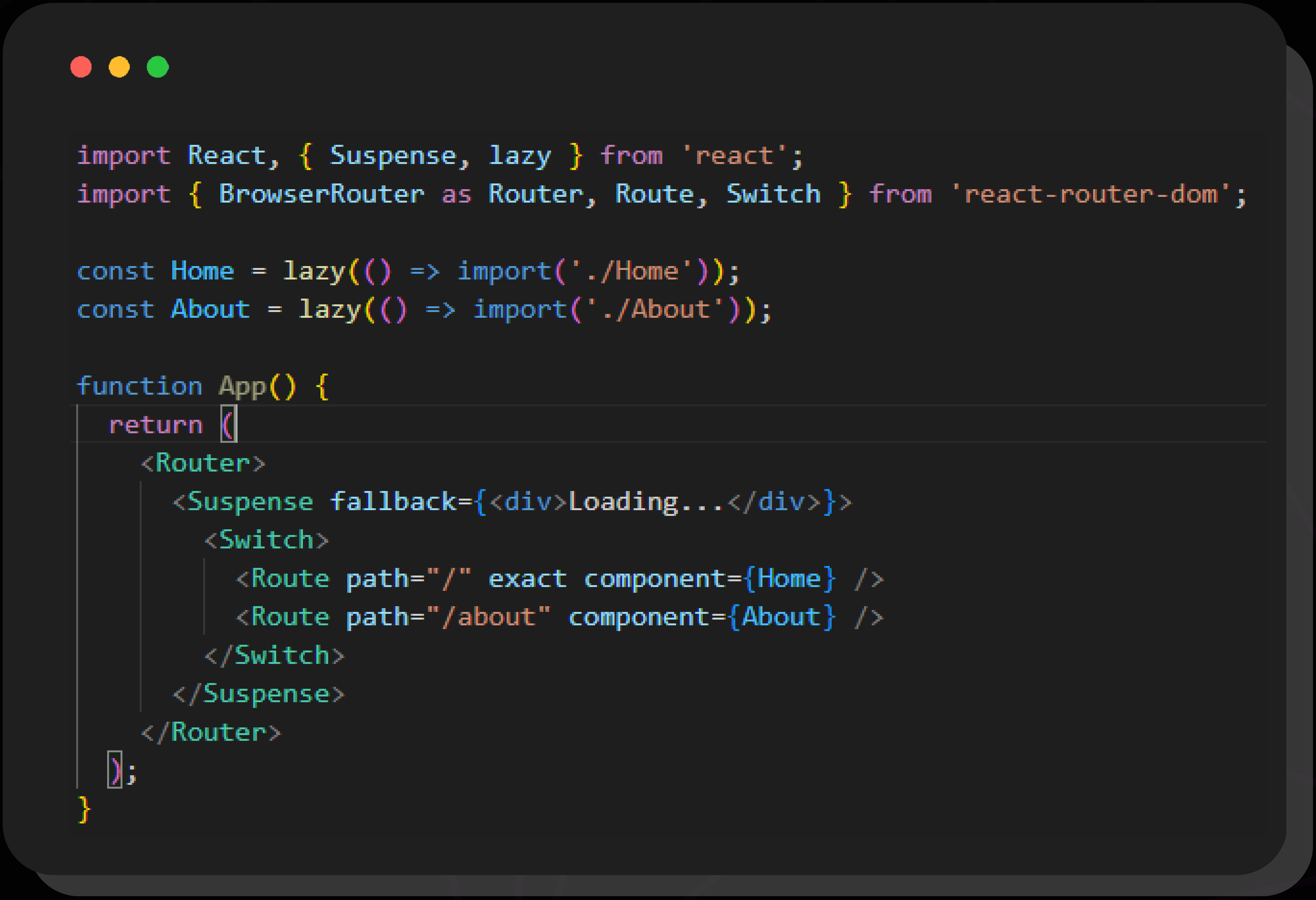
```
<Image
src="path/to/image.webp"
loader={<img src="path/to/placeholder.jpg" />}
alt="description"
/>
```

Route-Based Lazy Loading

Explanation: Load routes and their associated components only when they are needed, reducing the initial load time.

Implementation:

- Use React Router's lazy and Suspense for route-based code splitting.

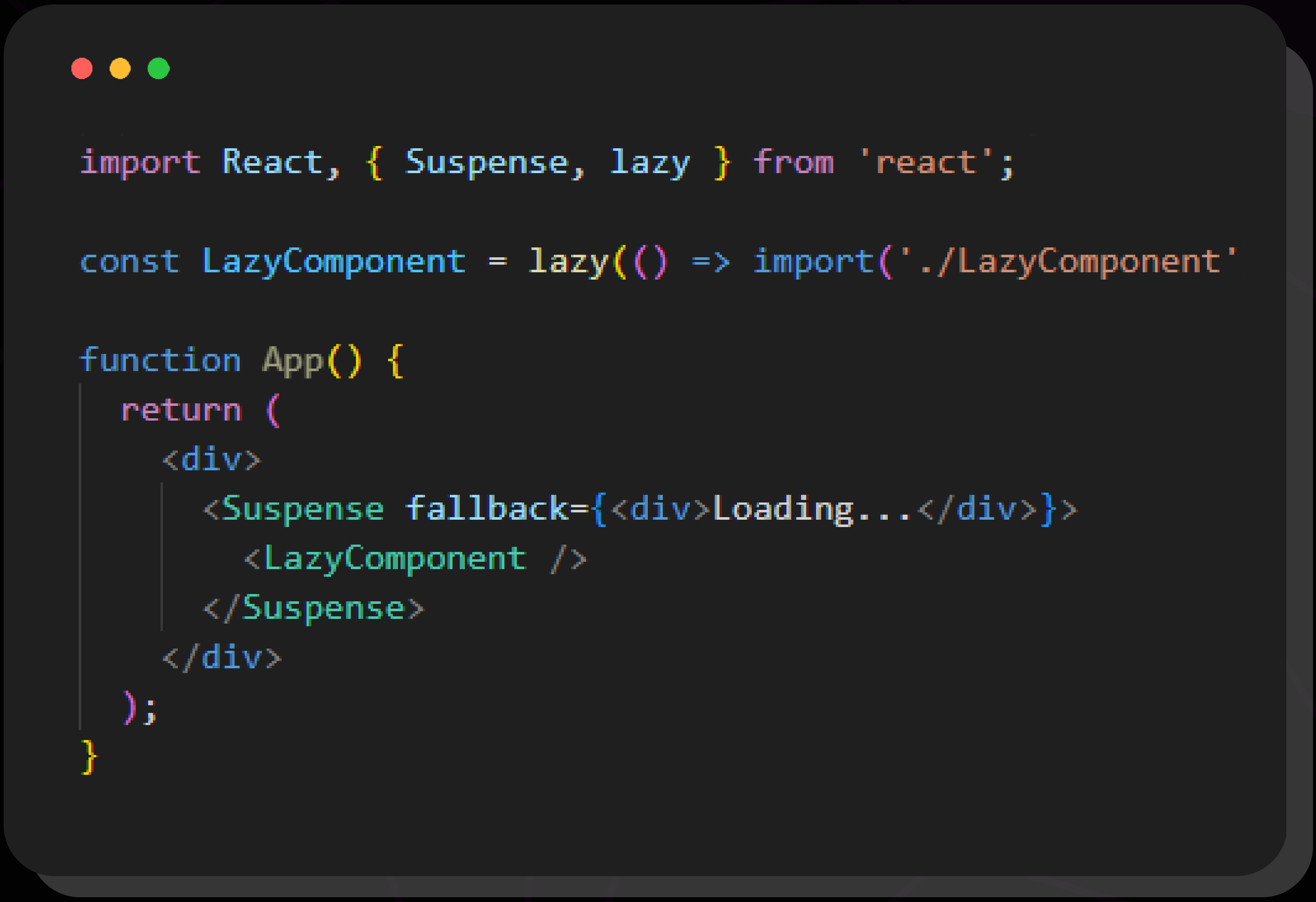


```
import React, { Suspense, lazy } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

const Home = lazy(() => import('./Home'));
const About = lazy(() => import('./About'));

function App() {
  return (
    <Router>
      <Suspense fallback={<div>Loading...</div>}>
        <Switch>
          <Route path="/" exact component={Home} />
          <Route path="/about" component={About} />
        </Switch>
      </Suspense>
    </Router>
  );
}
```

Component Lazy Loading

Explanation: Load components only when they are needed to reduce the initial load time.

Implementation:

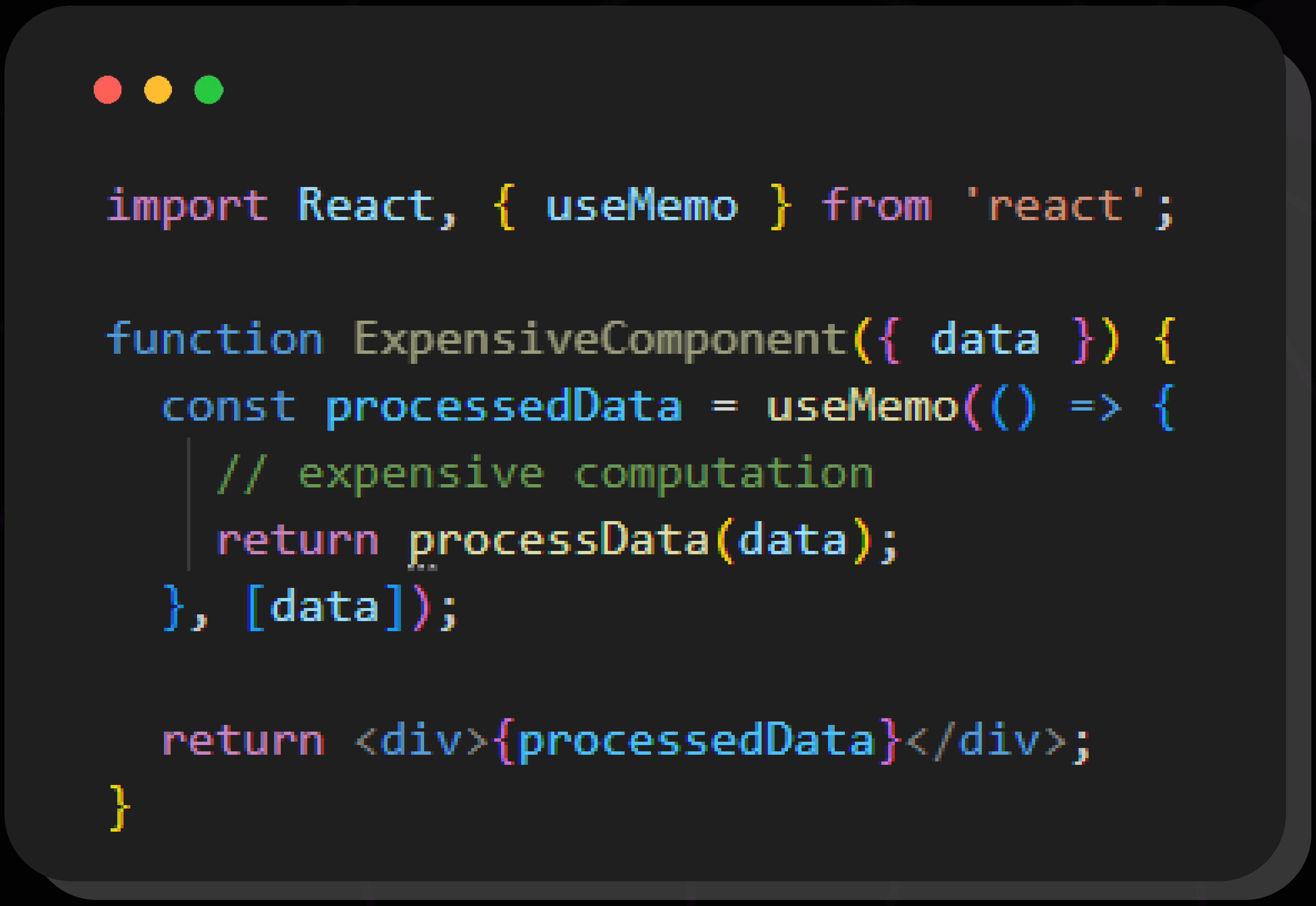


```
import React, { Suspense, lazy } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}
```

useMemo

Explanation: Memoize expensive calculations to avoid recalculating them on every render.

Implementation:

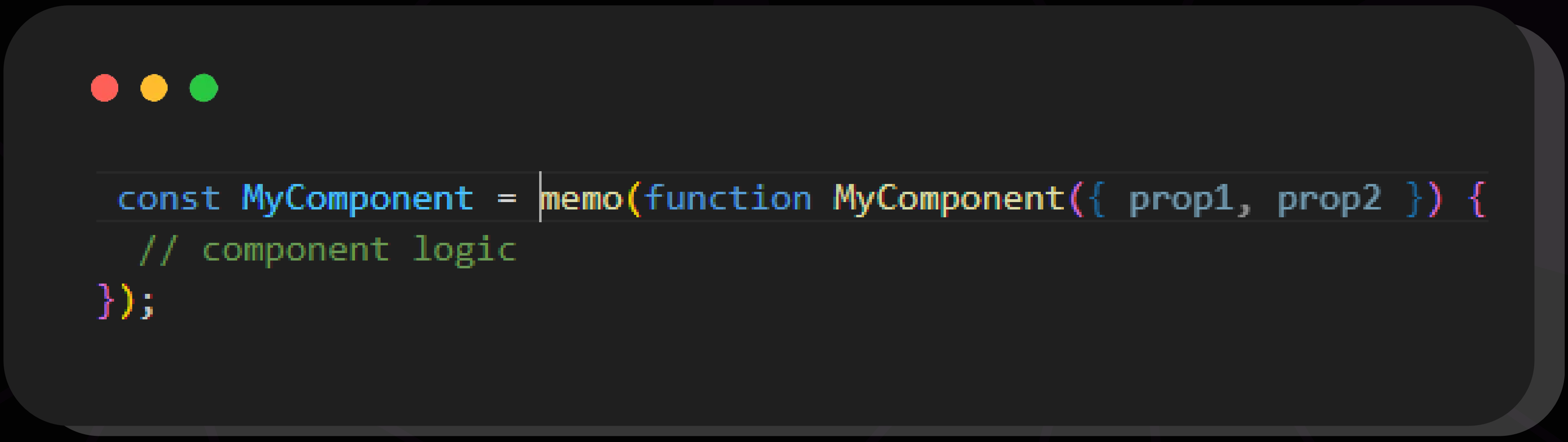


```
import React, { useMemo } from 'react';

function ExpensiveComponent({ data }) {
  const processedData = useMemo(() => {
    // expensive computation
    return processData(data);
  }, [data]);

  return <div>{processedData}</div>;
}
```

React.memo

Explanation: Prevent unnecessary re-renders of functional components by memoizing them.

Implementation:

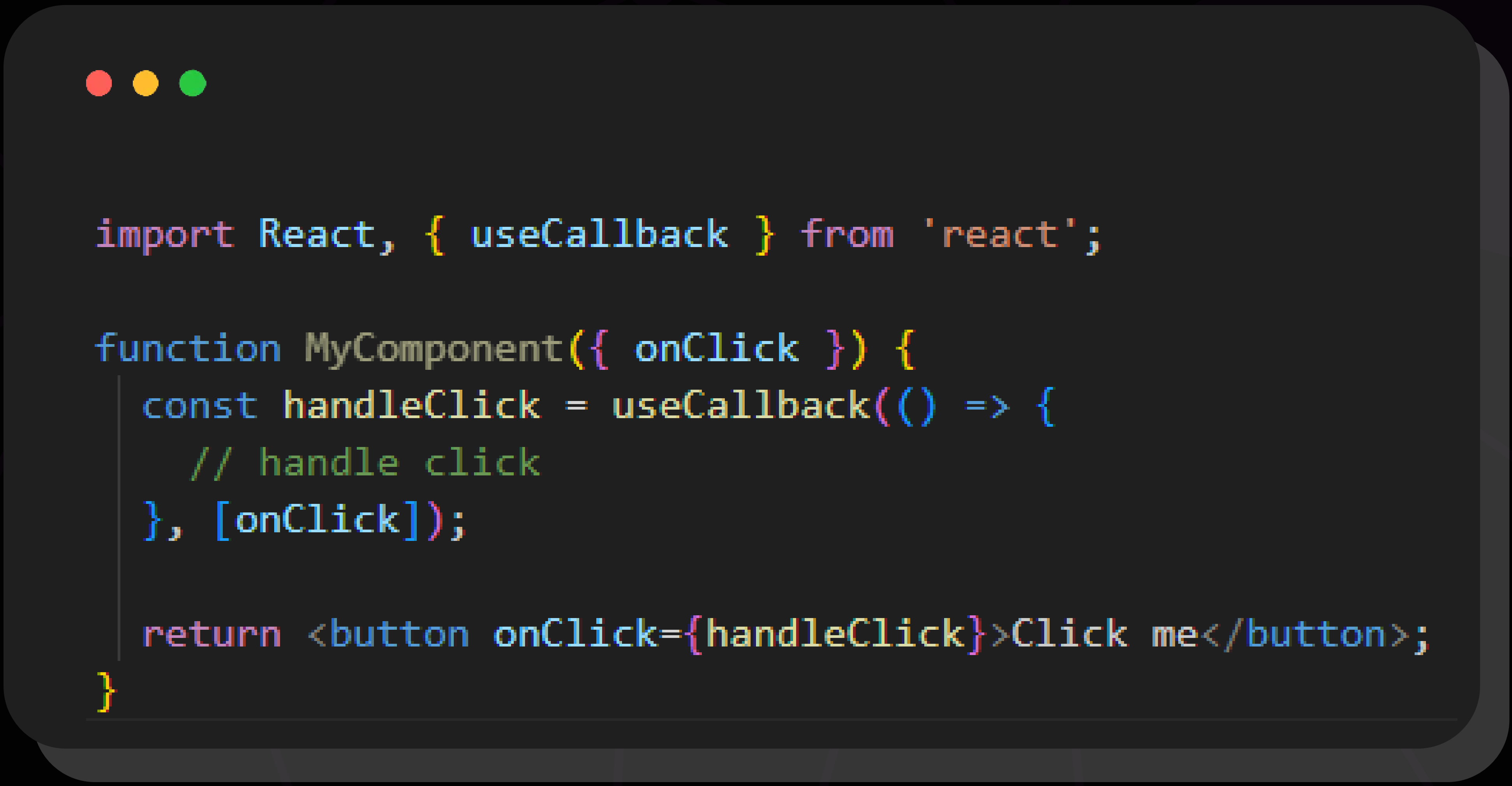


```
const MyComponent = memo(function MyComponent({ prop1, prop2 }) {
  // component logic
});
```

useCallback

Explanation: Memoize functions to prevent them from being re-created on every render.

Implementation:

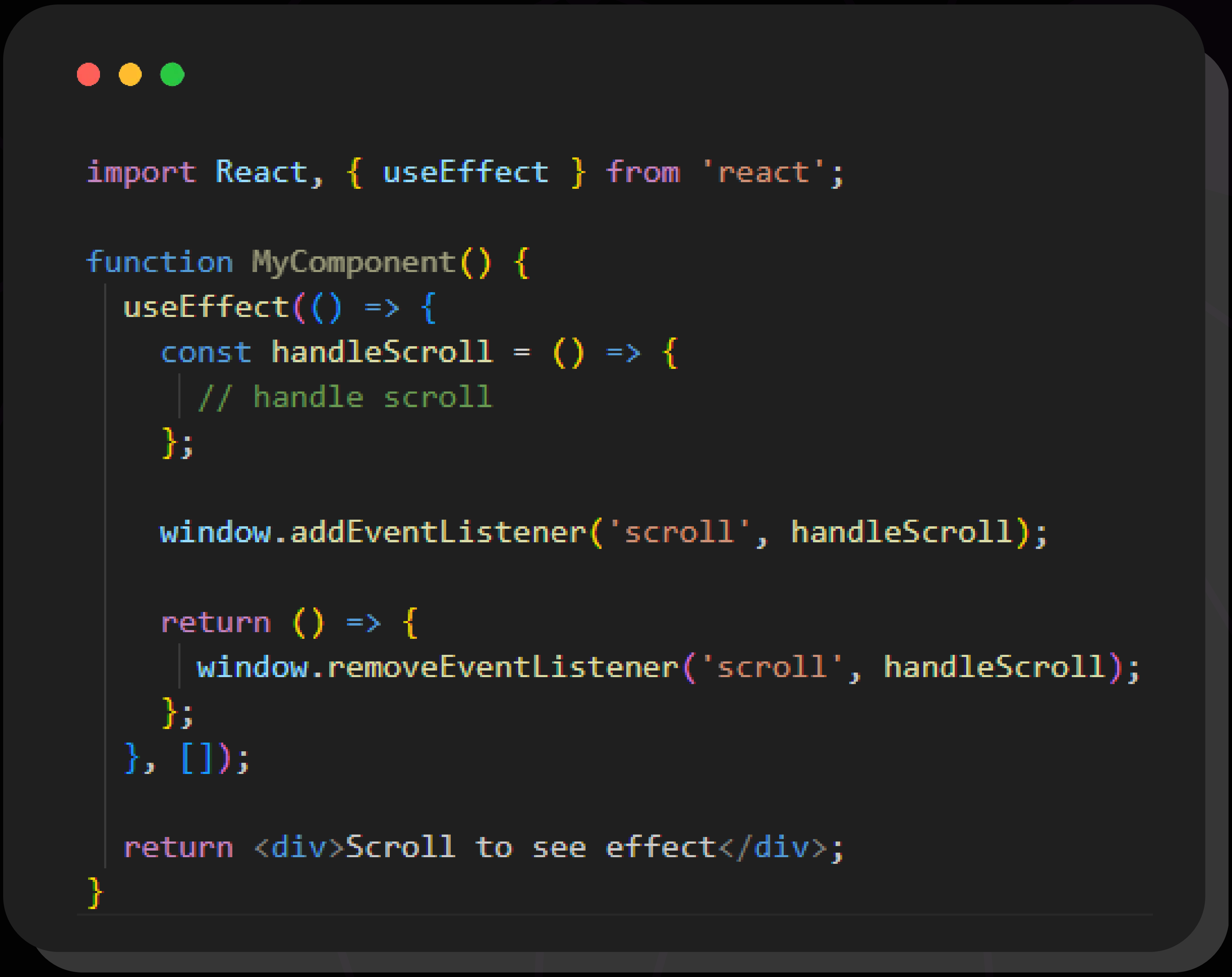


```
import React, { useCallback } from 'react';

function MyComponent({ onClick }) {
  const handleClick = useCallback(() => {
    // handle click
  }, [onClick]);

  return <button onClick={handleClick}>Click me</button>;
}
```

useEffect Cleanup

Explanation: Clean up side effects in useEffect to avoid memory leaks and ensure proper resource management.

Implementation:



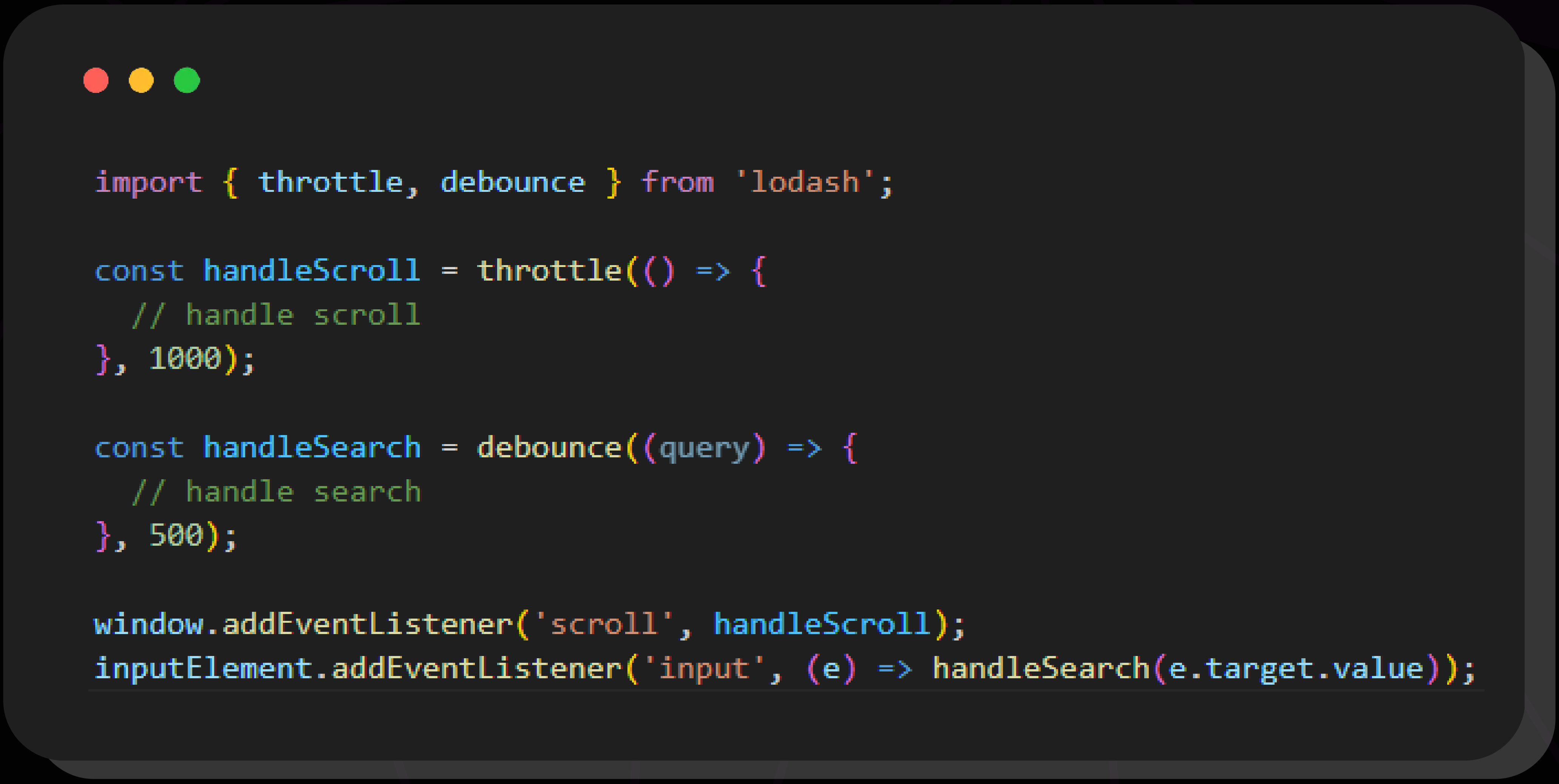
```
import React, { useEffect } from 'react';

function MyComponent() {
  useEffect(() => {
    const handleScroll = () => {
      // handle scroll
    };

    window.addEventListener('scroll', handleScroll);

    return () => {
      window.removeEventListener('scroll', handleScroll);
    };
  }, []);

  return <div>Scroll to see effect</div>;
}
```

Throttling and Debouncing

Explanation: Throttle or debounce expensive operations (e.g., API calls, event handlers) to improve performance.

Implementation:

- Use lodash's throttle and debounce functions.

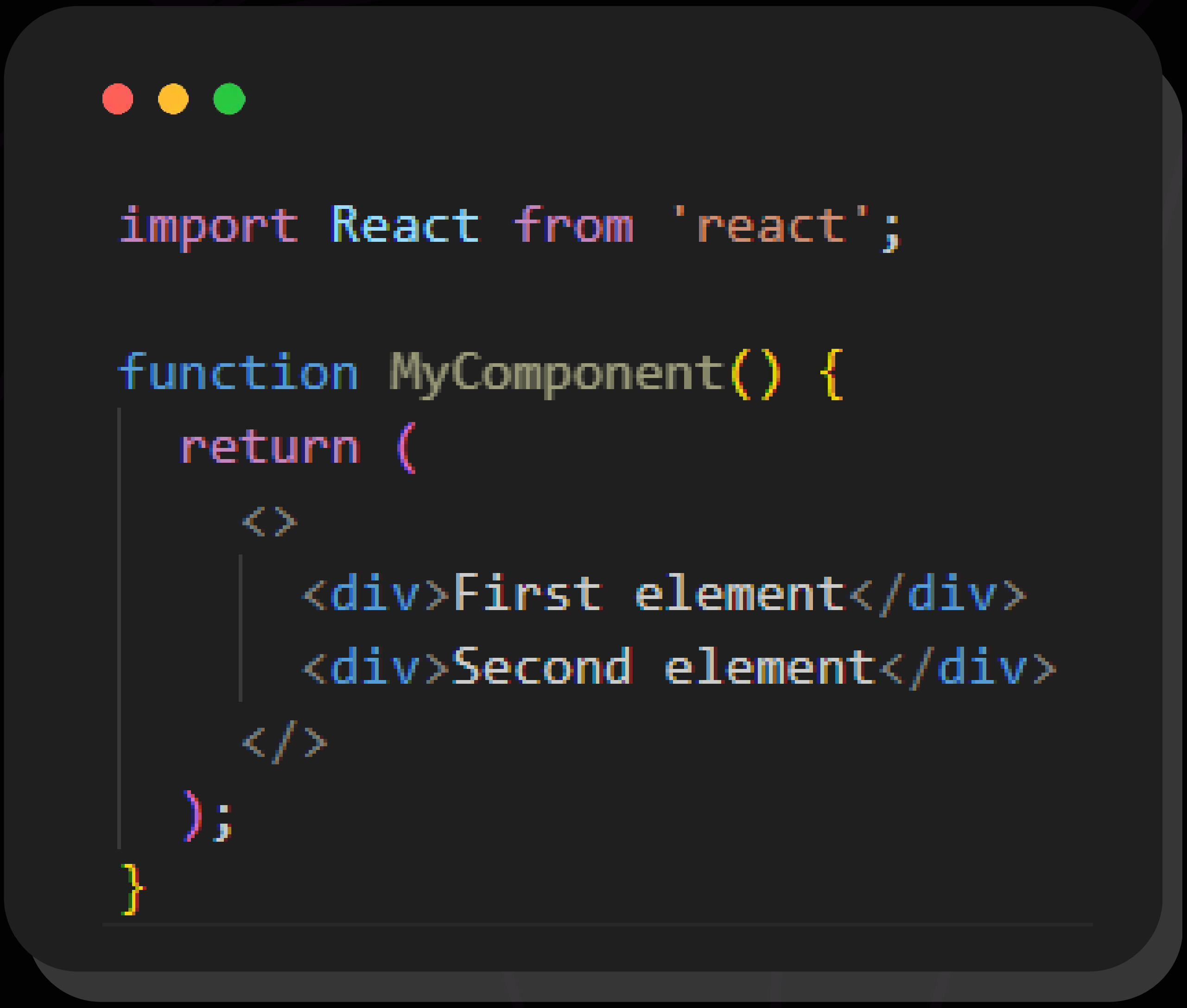


```
import { throttle, debounce } from 'lodash';

const handleScroll = throttle(() => {
  // handle scroll
}, 1000);

const handleSearch = debounce((query) => {
  // handle search
}, 500);

window.addEventListener('scroll', handleScroll);
inputElement.addEventListener('input', (e) => handleSearch(e.target.value));
```

Fragments

Explanation: Use fragments to avoid unnecessary wrapper elements in the DOM, which can reduce the number of nodes and improve rendering performance.

Implementation:

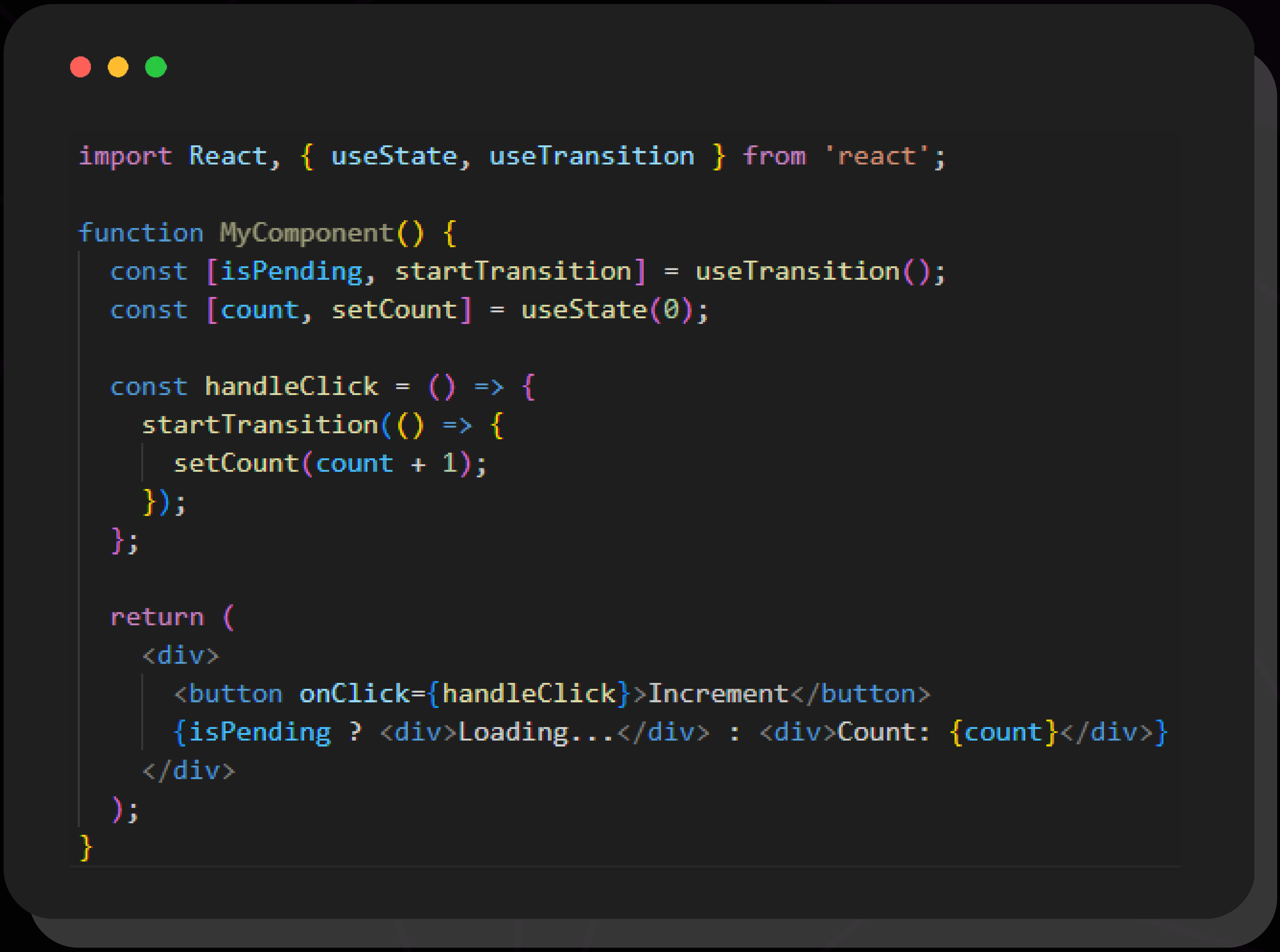


```
import React from 'react';

function MyComponent() {
  return (
    <>
      <div>First element</div>
      <div>Second element</div>
    </>
  );
}
```

useTransition

Explanation: Use useTransition to handle state transitions without blocking the UI, improving the perceived performance.

Implementation:

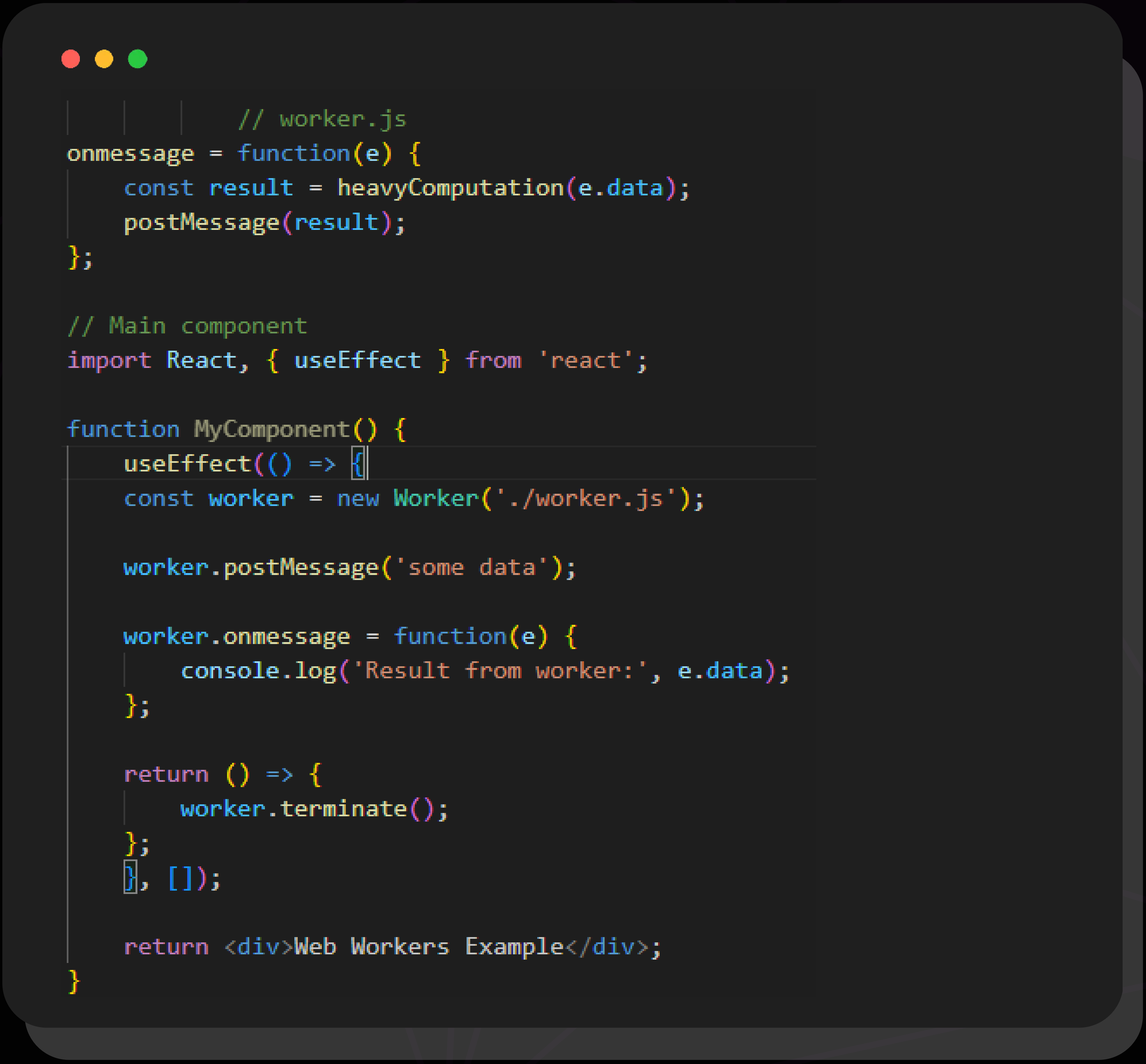


```
import React, { useState, useTransition } from 'react';

function MyComponent() {
  const [isPending, startTransition] = useTransition();
  const [count, setCount] = useState(0);

  const handleClick = () => {
    startTransition(() => {
      setCount(count + 1);
    });
  };

  return (
    <div>
      <button onClick={handleClick}>Increment</button>
      {isPending ? <div>Loading...</div> : <div>Count: {count}</div>}
    </div>
  );
}
```

Web Workers

Explanation: Use web workers to offload heavy computations to a background thread, keeping the UI responsive.

Implementation:

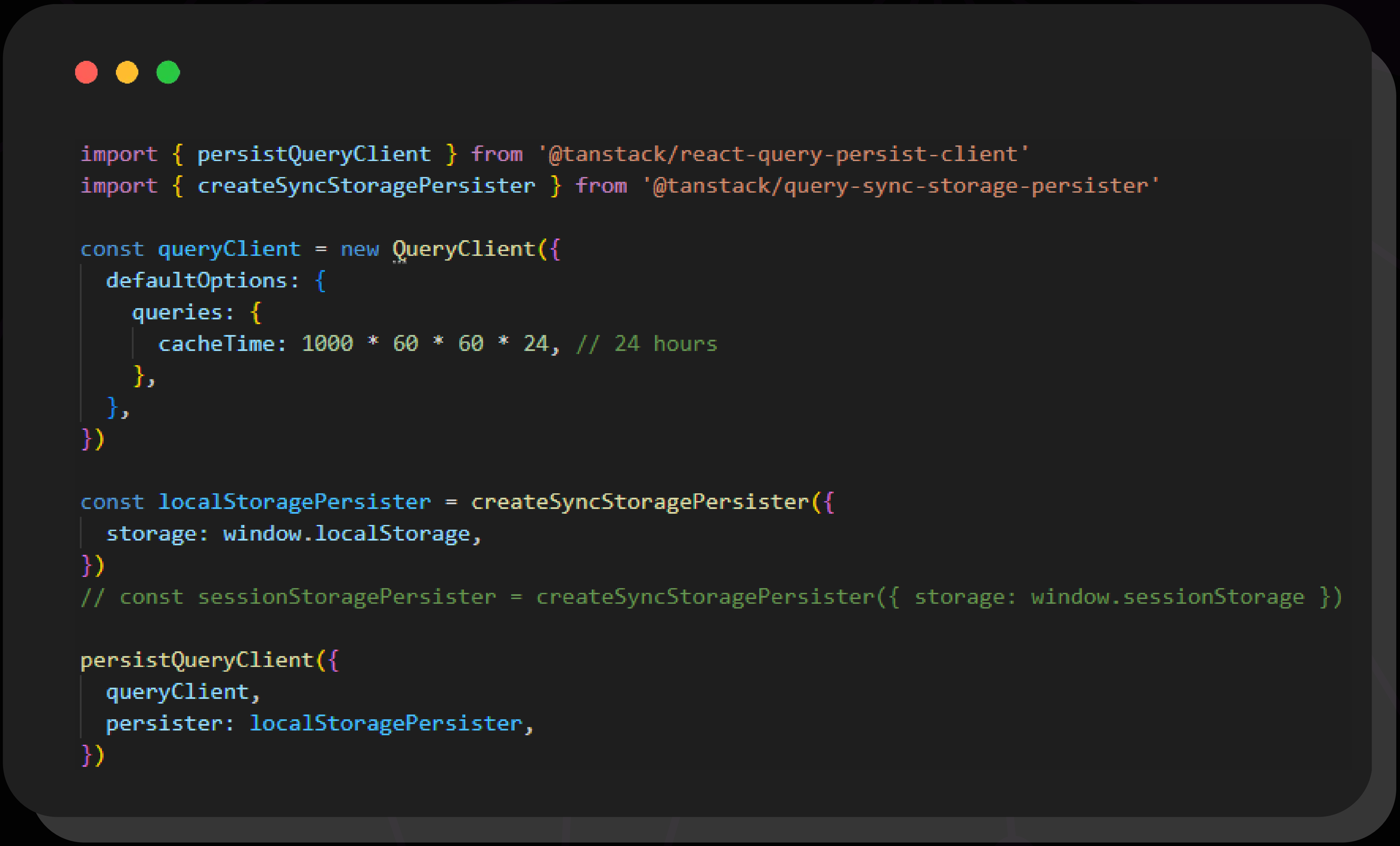


```
            // worker.js
onmessage = function(e) {
    const result = heavyComputation(e.data);
    postMessage(result);
};

// Main component
import React, { useEffect } from 'react';

function MyComponent() {
    useEffect(() => {
    const worker = new Worker('./worker.js');

    worker.postMessage('some data');

    worker.onmessage = function(e) {
        console.log('Result from worker:', e.data);
    };

    return () => {
        worker.terminate();
    };
    }, []);

    return <div>Web Workers Example</div>;
}
```

Caching with React Query

Explanation: React Query helps in fetching, caching, and synchronizing server state in your React applications, reducing network requests and improving performance.

Implementation:



```
import { persistQueryClient } from '@tanstack/react-query-persist-client'
import { createSyncStoragePersister } from '@tanstack/query-sync-storage-persister'

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      cacheTime: 1000 * 60 * 60 * 24, // 24 hours
    },
  },
})

const localStoragePersister = createSyncStoragePersister({
  storage: window.localStorage,
})
// const sessionStoragePersister = createSyncStoragePersister({ storage: window.sessionStorage })

persistQueryClient({
  queryClient,
  persister: localStoragePersister,
})
```

Follow me for more