`React.lazy` and `Suspense` allows for code-splitting, which means you can load components only when they are needed, reducing the initial load time of your application. Additionally, you can use an Error Boundary to catch and handle errors in your React components gracefully.

Here's how you can enhance the previous example to include `React.lazy`, `Suspense`, and an Error Boundary:

Step 1: Set up the project

Make sure you have a React project set up. You can create one using Create React App if you haven't already:

```
npx create-react-app lazy-suspense-example
cd lazy-suspense-example
npm start
```

Step 2: Create an Error Boundary

Create an Error Boundary component to catch errors in any component tree below it.

ErrorBoundary.js:

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

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render shows the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can also log the error to an error reporting service
    console.error("ErrorBoundary caught an error", error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}

export default ErrorBoundary;
```

Step 3: Create the lazy-loaded component

Create a `Counter` component that will be lazy-loaded.

Counter.js:

```
import React from 'react';

const Counter = React.memo(({ increment }) => {
  console.log('Counter component rendered');
  return (
    <button onClick={increment}>Increment</button>
  );
});

export default Counter;
```

Step 4: Update the App component to use `React.lazy` and `Suspense`

Update the `App` component to load the `Counter` component lazily and wrap it with `Suspense`.

App.js:

```
import React, { useState, useCallback, lazy, Suspense } from 'react';
import ErrorBoundary from './ErrorBoundary';

const Counter = lazy(() => import('./Counter'));

function App() {
  const [count, setCount] = useState(0);

  // useCallback to memoize the increment function
  const increment = useCallback(() => {
    setCount(prevCount => prevCount + 1);
  }, []);

  return (
    <div className="App">
      <h1>Count: {count}</h1>
      <ErrorBoundary>
        <Suspense fallback={<div>Loading...</div>}>
          <Counter increment={increment} />
        </Suspense>
      </ErrorBoundary>
    </div>
  );
}

export default App;
```

Step 5: Run the application

Run your application using `npm start`. When you click the "Increment" button, the `Counter` component should be loaded lazily, and the count should increase.

Explanation:

1. **ErrorBoundary**: This component catches JavaScript errors anywhere in its child component tree, logs those errors, and displays a fallback UI instead of the component tree that crashed.
2. **React.lazy**: This function lets you render a dynamic import as a regular component. In this example, the `Counter` component is loaded only when it's needed.
3. **Suspense**: This component is used to wrap the lazy-loaded component and provide a fallback while the component is being loaded.
4. **useCallback**: The `increment` function is memoized to prevent unnecessary re-renders of the `Counter` component.