**ReactJS Error Boundary**

Error Boundaries are a feature in React designed to catch JavaScript errors anywhere in their child component tree, log these errors, and displayed a fallback UI instead of the component tree that crashed.

They help improve the stability of React applications by preventing entire application from crashing due to error in specific components.

Features of error boundaries

Catch errors

Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

Fallback ui

They can display a fallback UI when error occurs, ensuring a better user experience even when things go wrong.

Logging

Errors caught by error boundaries can be logged for debugging purposes.

Use case for error boundaries

Error boundaries are most useful in situation where we want to catch and handle errors that occur during the rendering of a component tree. This can help prevent a entire application from crashing due to errors in specific parts of the UI.

* Component isolation
* Critical ui section
* Feature specific
* Large application

Methods

getDerivedStateFromError(error)

this lifecycle method is used to render a fallback ui after error has been thrown. It sets a state variable to indicate a error has occurred.

componentDidCatch(error, errorInfo)

this lifecycle method is used to log error information. It receives the error and additional error information which can be logged / send to a error reporting service.

Where to catch errors

Error boundaries catch errors in the

* During rendering

Error thrown during the rendering phase of a component

* In lifecycle methods

error thrown in lifecycle methods such as componentDidMount, componentDidUpdate etc.

* In constructors

Error thrown in the constructors of child component

Limitation of error boundaries

Error boundaries do not catch errors for

* Event handlers

Error thrown inside event handlers, we need to handle these error by using try catch blocks.

* Asynchronous code

Error thrown in asynchronous code, like setTimout / requestAnimationFram callback

* Server side rendering

Error thrown during server side rendering

* Errors in error boundaries

Error thrown inside the error boundary itself

Error boundary at page level

Wrap the entire page component to catch errors that occur anywhere on that page.

At the component level

Wrap specific component that are more prone to errors. This provides more granular error handling and allows specific fallback ui for different parts of the application

ReactJS Dynamic import

Dynamic imports in react allow us to load components / modules only when they are needed. We can improve the performance of our application by reducing the initial load tim.this can be achieved using React ‘React.lazy’ and ‘Suspense’ components.

* React.lazy

React.lazy is used to dynamically import a component. It takes a function that returns a promise which resolves to a module with a default export containing the react component.

How it works

code splitting

instead of loading the entire application bundle at once, React.lazy helps in splitting the code at component level.

Dynamic import

It uses the `import()` syntax to dynamically load the component when it is rendered.

* Suspense

Suspense is a component that lets us specify the loading fallback to show while waiting for a lazy loaded component to load.

Fallback ui

displays a fallback ui while the lazy-loaded component is being fetched.

Improve UX

Ensures users see a loading state instead of a blank screen when the component is loading.

Why they required

* Performance optimization

Reduce initial load tim

By spliting our code into smaller chunks and loading on demand, we reduce the initial load tim of our application

Efficient resource utilization

Only the necessary components are loaded, saving bandwidth and improving loading tim for users.

* Improved user experience

Loading state

Users see a loading indicator while components are being fetched, providing feedback that the app is working.

Progressive loading

Parts of the application can be loaded progressively, improving perceived performance.

* Scalability

Manage large application

As our application grows, code splitting becomes essential to manage the size and performance of our application.

Maintainability

Breaking down our application into smaller, lazy loaded component make It easier to manage and maintain.

React Dev tools

React devtools is a powerful browser extension that allows us to inspect the react component hierarchy, debug, and optimize performance.one of the key features of react devTools is the profiler, which helps us measure the performance of react application.