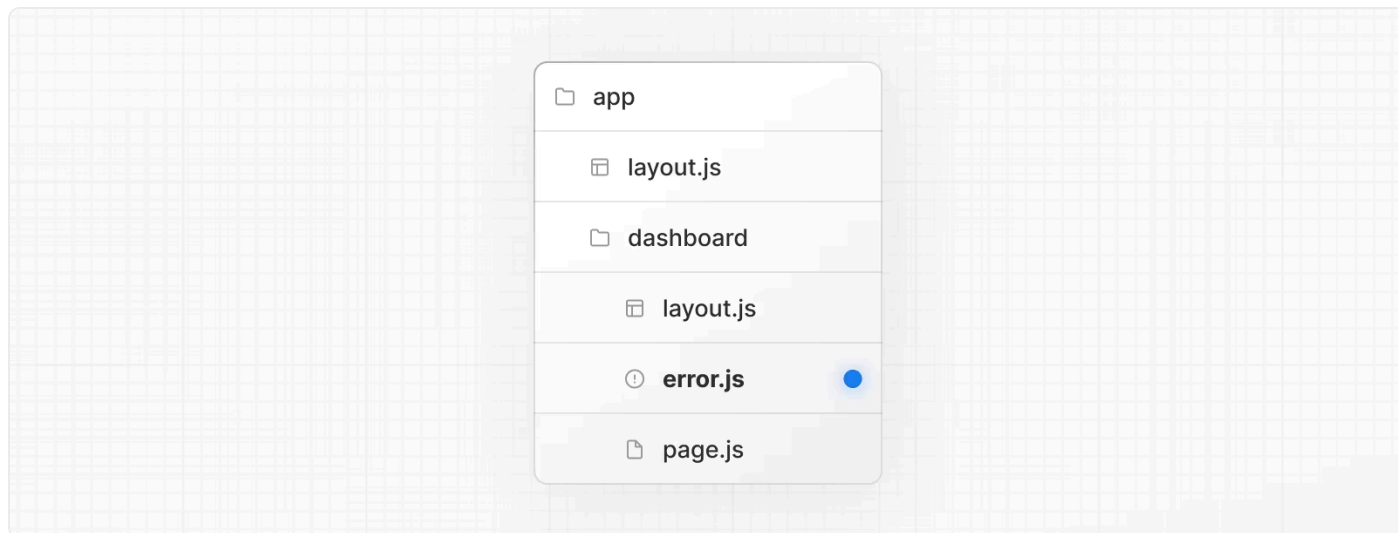


[> Menu](#)

# error.js

An **error** file allows you to handle unexpected runtime errors and display fallback UI.



TS app/dashboard/error.tsx

TypeScript ▾

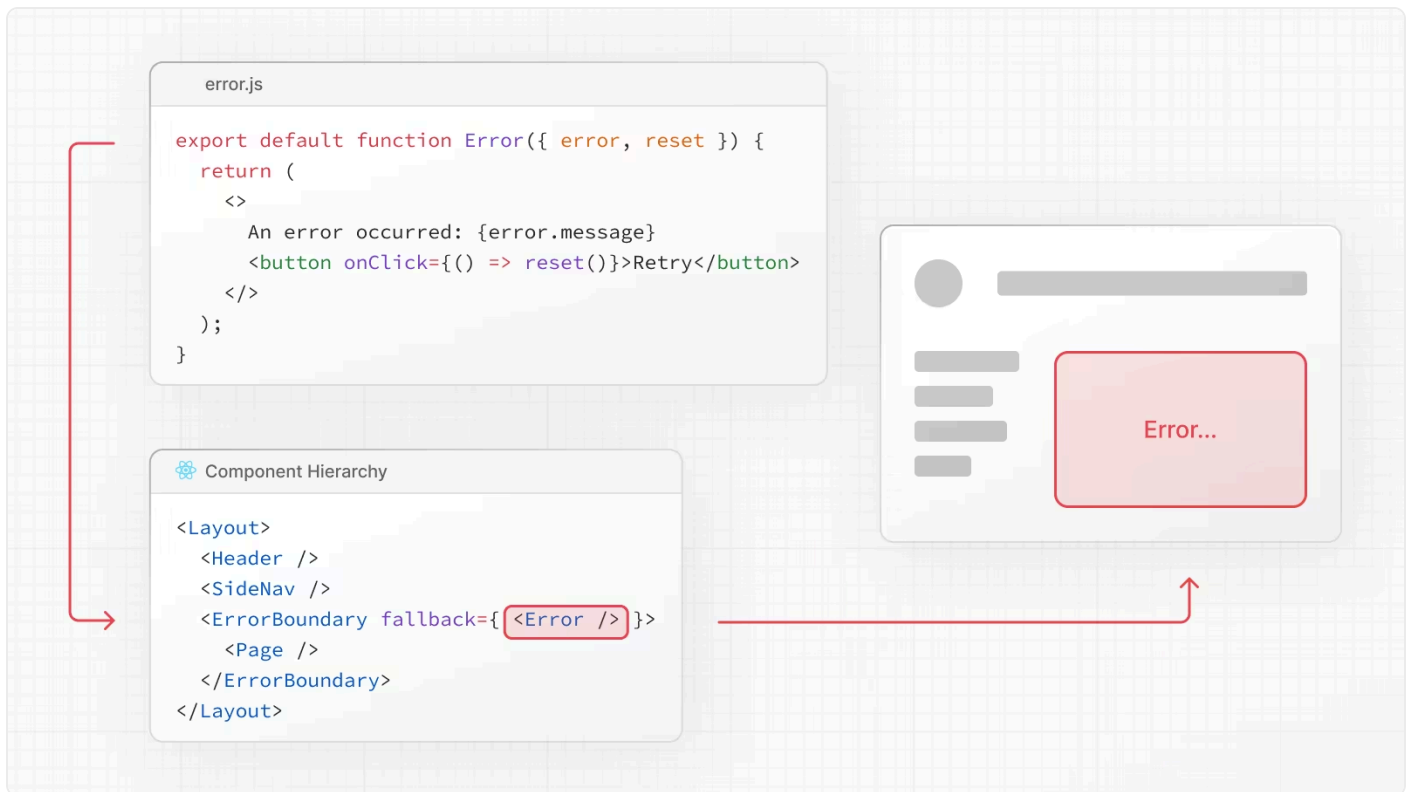


```
1  'use client' // Error boundaries must be Client Components
2
3  import { useEffect } from 'react'
4
5  export default function Error({
6    error,
7    reset,
8  }: {
9    error: Error & { digest?: string }
10   reset: () => void
11 }) {
12   useEffect(() => {
13     // Log the error to an error reporting service
```

```
14     console.error(error)
15   }, [error])
16
17   return (
18     <div>
19       <h2>Something went wrong!</h2>
20       <button
21         onClick={
22           // Attempt to recover by trying to re-render the segment
23           () => reset()
24         }
25       >
26         Try again
27       </button>
28     </div>
29   )
30 }
```

## How `error.js` Works

`error.js` wraps a route segment and its nested children in a [React Error Boundary](#). When an error throws within the boundary, the `error` component shows as the fallback UI.



#### Good to know:

- The [React DevTools](#) allow you to toggle error boundaries to test error states.

## Props

### `error`

An instance of an `Error` object forwarded to the `error.js` Client Component.

**Good to know:** During development, the `Error` object forwarded to the client will be serialized and include the `message` of the original error for easier debugging. However, **this behavior is different in production** to avoid leaking potentially sensitive details included in the error to the client.

### `error.message`

- Errors forwarded from Client Components show the original `Error` message.

- Errors forwarded from Server Components show a generic message with an identifier. This is to prevent leaking sensitive details. You can use the identifier, under `errors.digest`, to match the corresponding server-side logs.

### `error.digest`

An automatically generated hash of the error thrown. It can be used to match the corresponding error in server-side logs.

### `reset`

The cause of an error can sometimes be temporary. In these cases, trying again might resolve the issue.

An error component can use the `reset()` function to prompt the user to attempt to recover from the error. When executed, the function will try to re-render the error boundary's contents. If successful, the fallback error component is replaced with the result of the re-render.

TS app/dashboard/error.tsx

TypeScript ▾



```
1  'use client' // Error boundaries must be Client Components
2
3  export default function Error({
4    error,
5    reset,
6  }: {
7    error: Error & { digest?: string }
8    reset: () => void
9  }) {
10   return (
11     <div>
12       <h2>Something went wrong!</h2>
13       <button onClick={() => reset()}>Try again</button>
14     </div>
15   )
16 }
```

## global-error.js

While less common, you can handle errors in the root layout or template using `app/global-error.js`, located in the root `app` directory, even when leveraging [internationalization](#). Global error UI must define its own `<html>` and `<body>` tags. This file replaces the root layout or template when active.

TS app/global-error.tsx

TypeScript ▾



```
1  'use client' // Error boundaries must be Client Components
2
3  export default function GlobalError({
4    error,
5    reset,
6  }: {
7    error: Error & { digest?: string }
8    reset: () => void
9  }) {
10    return (
11      // global-error must include html and body tags
12      <html>
13        <body>
14          <h2>Something went wrong!</h2>
15          <button onClick={() => reset()}>Try again</button>
16        </body>
17      </html>
18    )
19  }
```

### Good to know:

- `global-error.js` is only enabled in production. In development, our error overlay will show instead.

## not-found.js

The `not-found` [↗](#) file shows UI when calling the `notFound()` function within a route segment.

# Version History

Version	Changes
v13.1.0	<code>global-error</code> introduced.
v13.0.0	<code>error</code> introduced.

# Learn more about error handling

App Router > ... > Routing

Error Handling





Learn how to display expected errors and handle uncaught exceptions.

Previous

< default.js

Next

instrumentation.js >

Was this helpful?    

- [Blog](#)
- [Releases](#)
- [X](#)
- [Analytics](#)
- [Telemetry](#)
- [Next.js Conf](#)
- [Governance](#)
- [Previews](#)

Subscribe to our newsletter

Stay updated on new releases and features, guides, and case studies.

you@domain.com

Subscribe

© 2024 Vercel, Inc.

