Reserved File Names and Component Hierarchy

14 min

So far, we've learned how Next.js uses folders and reserved files to build path segments and UIs.

Notice that each reserved file we've seen so far has a single purpose with special functionality, for example:

- page.tsx: Makes a URL segment accessible and displays a unique UI.
- layout.tsx: Creates a shared UI and wraps any nested UIs, maintains state on nested segment navigation.
- template.tsx: Creates a shared UI and wraps any nested UIs, re-instantiates on nested segment navigation.

These are only some of the reserved files Next.js provides. Let's explore three more of the commonly used ones:

- error.tsx: Creates an ErrorBoundary component using the defined UI for the current segment and its nested segments.
- not-found.tsx: Creates an ErrorBoundary component using the defined UI for 404 errors for the current segment and its nested segments.
- loading.tsx: Creates a Suspense component for the current segment and its nested segments (we will explore this fully in a future lesson).

You define one like any other reserved file (create the file in a segment) and export a React component. For example:

```
/* in error.tsx */

'use client' // Must be client components
//receives error object and reset function as props
export default function MyErrorBoundary( { error, reset }: { error: Error; reset: () => void} ) {
  // body
}

/* in not-found.tsx */
export default function MyNotFoundUI() {
  return (
    <>
      <h1>Sorry, I don't recognize this page!</h1>
      {/* rest of components */}
    </>
  )
}
```

```
/* in loading.tsx */
export default function Loading() {
  return <h1>Loading content...</h1>
}
```

In the example, the not-found.tsx and loading.tsx exported components do not receive props. error.tsx receives an Error object named error and a reset callback named reset(). Note that error.tsx components *must* be client components and they *do not* catch errors thrown in layout.tsx or template.tsx (we'll discuss this next).

Lastly, you've probably been wondering how all these files interact with each other. For example, why doesn't error.tsx catch errors in layout.tsx/template.tsx, or can you include a layout.tsx and template.tsx in the same segment? The answers to both can be found in Next.js **component hierarchy**.

The component hierarchy establishes how all the default exported components in the route segment's reserved files are rendered.

```
 ├— app
|    ├— layout.tsx
|    ├— template.tsx
|    ├— loading.tsx
|    ├— settings
|   |    ├— layout.tsx
|   |    ├— template.tsx
|   |    ├— loading.tsx
|   |    ├— error.tsx
|   |    ├— not-found.tsx
```

For an app with the above folder structure,

```
<Layout>
  <Template>
    <Suspense>
      <Layout>
        <Template>
          <ErrorBoundary>
            <Suspense>
              <ErrorBoundary>
                <Page/>
              </ErrorBoundary>
            </Suspense>
```

```
        </ErrorBoundary>
      </Template>
    </Layout>
  </Suspense>
 </Template>
</Layout>
```

and, in this hierarchy:

- <Layout> is the root of a segment.
- <Template> wraps all but <Layout>.
- <ErrorBoundary> wraps <Suspense>, the not-found <ErrorBoundary>, and <Page>.
- <Suspense> wraps not-found <ErrorBoundary/> and <Page>.
- not-found <ErrorBoundary/> wraps <Page>.
- Nested segments hierarchy follows the same hierarchy rules and is wrapped *entirely* within the parent hierarchy.

Understanding the component hierarchy will help us master Next.js routing and understand where certain reserved files are needed. Let's practice using not-found.tsx in our application.

Instructions

1. Checkpoint 1 Passed

1.

Currently, when navigating to an unknown path like /does/not/exist, the application crashes.

Fix this by creating the resolved file that wraps our application in a not-found ErrorBoundary.

Hint

Recall that Next.js provides a reserved file to handle not found errors.

2. Checkpoint 2 Passed

2.

Finally, default export a React component that renders the message Oops, looks like we sent you the wrong way in an <h1> element.

Run the code using npm run dev in the terminal. Once you see a console message indicating "Ready," click the Refresh Browser button in the workspace browser bar.

If successful, you should be able to navigate to a path like /does/not/exist and see our not-found message.

Hint

A not-found ErrorBoundary can be created by defining a not-found.tsx in a segment and default exporting a React component.

Recall that because of the component hierarchy, to catch all unknown URLs not-found errors, the not-found.tsx should be located at the topmost segment.

Enter the command inside the terminal window.

The window can be refreshed when it displays “Ready.”

not-found.tsx

```
export default function MyNotFoundUI() {
  return (
    <>
      <h1>Oops, looks like we sent you the wrong way</h1>
    </>
  )
}
```