Lazy Loading & Code Splitting

Loading components later with React.lazy, route-based code splitting, using lazy with named exports, and conditional lazy loading -- with simple examples.

Advanced 11 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/lazy-loading-code-splitting

---

Lazy Loading & Code Splitting

Every component we've written so far has been part of the application's FIRST loaded JavaScript bundle. As an application grows, so does this bundle -- users may end up downloading code for pages they'll never even visit. This lesson covers how to avoid that.

Loading a Component Later with React.lazy

`lazy()` turns a component's code into a SEPARATE file that's downloaded when needed, instead of a regular `import`:

```
import { lazy, Suspense, useState } from "react";

// lazy() loads the component's CODE with a dynamic import() instead of
// a regular import -- this file is NOT INCLUDED in the app's initial
// bundle, it's only downloaded as a separate file when actually needed.
// This is what we call "code splitting".
const CourseDetails = lazy(() => import("./CourseDetails.jsx"));

function ReactLazyBasicExample() {
  const [showDetails, setShowDetails] = useState(false);

  return (
    <div>
      <button onClick={() => setShowDetails(true)}>Show Details</button>
      {showDetails && (
        // Suspense specifies a fallback UI to show WHILE the lazy
        // component's code is LOADING -- `fallback` is shown until the
        // code is downloaded, then the real component is rendered.
        <Suspense fallback={<p>Loading...</p>}>
          <CourseDetails />
        </Suspense>
      )}
    </div>
  );
}
```

`lazy(() => import("./CourseDetails.jsx"))` REMOVES `CourseDetails`'s code from the app's initial bundle -- it's only downloaded once `showDetails` becomes `true`. `Suspense` is required to show a `fallback` during this download (we'll take a closer look at Suspense in the next lesson).

Route-Based Code Splitting

The most common use of `lazy()` is splitting the pages from the Routing lesson into separate bundles:

```
import { lazy, Suspense } from "react";
import { BrowserRouter, Routes, Route } from "react-router";

// In the Routing lesson we loaded every page with a regular import -- in
// a real app, we don't want to download the code for pages the user
// might never visit BEFORE they navigate to that page. With lazy() we can
// turn each page into a separate bundle and only download it when that
// route is actually visited.
const CoursesPage = lazy(() => import("./CoursesPage.jsx"));
const AboutPage = lazy(() => import("./AboutPage.jsx"));

function RouteBasedCodeSplittingExample() {
  return (
    <BrowserRouter>
      {/* Suspense wraps AROUND Routes -- no matter which page is visited,
          a SINGLE fallback is shown until that page's code is loaded. */}
      <Suspense fallback={<p>Loading page...</p>}>
        <Routes>
          <Route path="/courses" element={<CoursesPage />} />
          <Route path="/about" element={<AboutPage />} />
        </Routes>
      </Suspense>
    </BrowserRouter>
  );
}
```

Each page (`CoursesPage` , `AboutPage`) is its own separate file -- if a user never visits `/about` , that page's code is never downloaded. This pattern is called **code splitting**: breaking an application into multiple small pieces instead of one giant bundle.

Using lazy with Named Exports

`lazy()` expects `import()` to resolve to a DEFAULT export -- a component with a named export needs a small adaptation:

```
import { lazy, Suspense } from "react";

// lazy() expects the import() function to return a DEFAULT export --
// if CourseChart is only a named export (`export function
// CourseChart() {}`, NOT `export default`), we need to wrap it into a
// "default" field using .then().
const CourseChart = lazy(() =>
  import("./CourseChart.jsx").then((module) => ({ default: module.CourseChart })),
);

function NamedExportLazyExample() {
  return (
    <Suspense fallback={<p>Loading chart...</p>}>
      <CourseChart />
    </Suspense>
  );
}
```

`.then((module) => ({ default: module.CourseChart }))` CONVERTS the named export (`CourseChart`) into the `{ default: ... }` shape that `lazy` expects.

Conditional Lazy Loading

`lazy()` is useful not just for pages, but for ANY rarely-used component:

```
import { lazy, Suspense, useState } from "react";

// Making a large, rarely used component like EmojiPicker lazy is
// especially useful -- most users may never open it, in which case we
// never download its code at all.
const EmojiPicker = lazy(() => import("./EmojiPicker.jsx"));

function ConditionalLazyLoadExample() {
  const [showPicker, setShowPicker] = useState(false);

  return (
    <div>
      <button onClick={() => setShowPicker(!showPicker)}>
        {showPicker ? "Hide" : "Show"} Emoji Picker
      </button>
      {/* EmojiPicker's code is NEVER downloaded until `showPicker` becomes
          true for the FIRST time -- it's only loaded if it will actually
          be used. */}
      {showPicker && (
        <Suspense fallback={<p>Loading emoji picker...</p>}>
          <EmojiPicker />
        </Suspense>
      )}
    </div>
  );
}
```

`EmojiPicker`'s code is never downloaded until the user makes `showPicker` `true` for the FIRST time -- most users may never use it, in which case we never download its code at all.

Summary and Glossary

`lazy()` splits a component's code into a separate file (chunk), downloading it only when it's actually needed -- this REDUCES the amount of JavaScript loaded initially. Its most common uses are splitting pages (routes) or rarely-used components (modals, emoji pickers). `lazy()` is always used together with `Suspense` -- a fallback is needed while the code downloads.

Glossary

Code Splitting — The technique of breaking an application's JavaScript into small pieces that are downloaded as needed, instead of one large bundle.

Bundle — An application's JavaScript files combined together to be sent to the browser.

Chunk — A small, separately downloadable JavaScript file produced by code splitting.