Lazy Loading & Code Splitting

React.lazy ile component'leri sonradan yüklemek, route bazlı code splitting, named export'larla lazy kullanmak, ve koşullu lazy loading -- basit örneklerle.

İleri 11 dk

Son güncelleme: 18 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/lazy-loading-code-splitting

---

Lazy Loading & Code Splitting

Şimdiye kadar yazdığımız tüm component'ler, uygulamanın İLK yüklenen JavaScript paketine (bundle) dahildi. Uygulama büyüdükçe bu paket de büyür -- kullanıcı, belki hiç ziyaret etmeyeceği sayfaların kodunu bile indirmek zorunda kalır. Bu ders, bunu önlemenin yolunu anlatıyor.

React.lazy ile Bir Component'i Sonradan Yüklemek

`lazy()`, bir component'in kodunu normal bir `import` yerine, ihtiyaç duyulduğunda indirilen AYRI bir dosya haline getirir:

```
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"))`, `CourseDetails`'in kodunu uygulamanın ilk paketinden ÇIKARIR -- yalnızca `showDetails` `true` olduğunda indirilir. `Suspense`, bu indirme sırasında bir `fallback` göstermek için gereklidir (bir sonraki derste Suspense'e daha yakından bakacağız).

Route Bazlı Code Splitting

`lazy()`'nin en yaygın kullanımı, Routing dersindeki sayfaları ayrı paketlere bölmek:

```
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>
  );
}
```

Her sayfa (`CoursesPage`, `AboutPage`) kendi ayrı dosyası -- kullanıcı `/about`'a hiç gitmezse, o sayfanın kodu hiç indirilmez. Bu desene **code splitting** (kod bölme) denir: uygulamanın tek bir dev paket yerine, birden fazla küçük parçaya bölünmesi.

Named Export'larla lazy Kullanmak

`lazy()`, `import()`'un bir DEFAULT export döndürmesini bekler -- named export'lu bir component için küçük bir uyarlama gerekir:

```
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 }))`, named export'u (`CourseChart`) `lazy`'nin beklediği `{ default: ... }` şekline DÖNÜŞTÜRÜYOR.

Koşullu Lazy Loading

`lazy()`, yalnızca sayfalar için değil, nadiren kullanılan HERHANGİ bir component için faydalı:

```
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`'ın kodu, kullanıcı `showPicker`'ı İLK KEZ `true` yapana kadar hiç indirilmez -- kullanıcıların çoğu belki hiç kullanmaz, o zaman kodunu hiç indirmemiş oluruz.

Özet ve Terimler Sözlüğü

`lazy()`, bir component'in kodunu ayrı bir dosyaya (chunk) bölüp, yalnızca gerçekten gerektiğinde indirir -- bu, uygulamanın ilk yüklenen JavaScript miktarını AZALTIR. En yaygın kullanımı, sayfaları (route'lar) ya da nadiren kullanılan component'leri (modal'lar, emoji picker'lar gibi) bölmek. `lazy()` her zaman bir `Suspense` ile birlikte kullanılır -- kod indirilirken gösterilecek bir fallback gerekir.

Terimler Sözlüğü

Code Splitting (Kod Bölme) — Bir uygulamanın JavaScript'ini tek bir büyük paket yerine, ihtiyaç duyuldukça indirilen küçük parçalara bölme tekniği.

Bundle (Paket) — Bir uygulamanın, tarayıcıya gönderilmek üzere birleştirilmiş JavaScript dosyası.

Chunk (Parça) — Code splitting sonucu oluşan, ayrı ayrı indirilebilen küçük bir JavaScript dosyası.