Suspense

Suspense'in fallback prop'u, iç içe Suspense sınırları, React 19'un use() hook'u ile Suspense entegrasyonu, ve Suspense'in otomatik yapmadıkları -- basit örneklerle.

İleri 11 dk

Son güncelleme: 6 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/suspense

---

Suspense

Lazy Loading dersinde `Suspense`'i yalnızca `lazy()` ile birlikte gördük. Bu ders, `Suspense`'in kendisine -- ne yaptığına, nasıl iç içe kullanılabileceğine, ve neyi OTOMATİK OLARAK yapmadığına -- daha yakından bakıyor.

fallback Prop'u

`Suspense`, İÇİNDEKİ bir şey HENÜZ hazır değilken bir `fallback` gösterir:

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

const CourseDetails = lazy(() => import("./CourseDetails.jsx"));

function SuspenseFallbackExample() {
  return (
    // `fallback` can be any JSX -- not just text, but also a spinner, a
    // skeleton screen, or another component. In the Lazy Loading lesson
    // we only saw Suspense used together with lazy() -- this lesson
    // takes a closer look at Suspense itself.
    <Suspense fallback={<p className="spinner">Loading course...</p>}>
      <CourseDetails />
    </Suspense>
  );
}
```

`fallback`, yalnızca bir metin değil, HERHANGİ bir JSX olabilir -- bir spinner, bir iskelet (skeleton) ekran, ya da başka bir component. İçindeki component (burada `CourseDetails`) hazır olduğunda, `fallback` otomatik olarak gerçek içerikle DEĞİŞTİRİLİR.

İç İçe Suspense Sınırları

Birden fazla `Suspense`, farklı seviyelerde iç içe kullanılabilir:

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

const CourseHeader = lazy(() => import("./CourseHeader.jsx"));
const CourseReviews = lazy(() => import("./CourseReviews.jsx"));

function NestedSuspenseExample() {
  return (
    // The outer Suspense shows a fallback for the WHOLE page until
    // CourseHeader loads. Once CourseHeader appears, the INNER Suspense
    // only takes the place of CourseReviews -- the rest of the page
    // (including CourseHeader) does NOT go back to a "loading" state.
    <Suspense fallback={<p>Loading page...</p>}>
      <CourseHeader />
      <Suspense fallback={<p>Loading reviews...</p>}>
        <CourseReviews />
      </Suspense>
    </Suspense>
  );
}
```

Dıştaki `Suspense`, `CourseHeader` yüklenene kadar TÜM sayfa için bir fallback gösterir. `CourseHeader` göründükten sonra, İÇTEKİ `Suspense` yalnızca `CourseReviews`'un yerini kaplar -- sayfanın geri kalanı tekrar "loading" durumuna DÖNMEZ. Bu, kullanıcıya daha akıcı bir deneyim sunar: her şeyin birden kaybolup gelmesi yerine, yalnızca gerçekten bekleyen kısım "loading" gösterir.

use() Hook'u ile Suspense

React 19'daki `use()` hook'u, bir Promise'i doğrudan Suspense ile entegre EDEBİLİR:

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

function fetchCourse() {
  return fetch("http://localhost:3000/courses/1").then((response) => response.json());
}

// To avoid creating a NEW Promise on every render, we call this once
// OUTSIDE the component, while the module loads.
const coursePromise = fetchCourse();

function CourseName() {
  // Unlike normal hooks, use() can also be called CONDITIONALLY. When
  // given a Promise, if the Promise has NOT resolved yet, use() tells
  // React "I need to wait" -- this shows the fallback of the nearest
  // Suspense; once the Promise resolves, it returns the actual value.
  const course = use(coursePromise);
  return <p>Course: {course.name}</p>;
}

function UsePromiseWithSuspenseExample() {
  return (
    <Suspense fallback={<p>Loading course...</p>}>
      <CourseName />
    </Suspense>
  );
}
```

`use()`, diğer hook'ların aksine KOŞULLU olarak da çağrılabilir. Bir Promise verildiğinde, henüz ÇÖZÜLMEDİYSE React'e "beklemem gerekiyor" der -- bu, en yakın `Suspense`'in `fallback`'ini gösterir; Promise çözülünce `use()` gerçek değeri döner ve component normal render edilir.

Suspense'in Otomatik Yapmadıkları

Önemli bir gotcha: her asenkron işlem, Suspense'i otomatik tetiklemez:

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

function CourseListWithEffect() {
  const [courses, setCourses] = useState(null);

  // IMPORTANT: useEffect + fetch (the pattern from the API & Data
  // Fetching lesson) does NOT AUTOMATICALLY TRIGGER Suspense -- Suspense
  // only works with a Promise source that React DIRECTLY recognizes,
  // like use(). That's why, while the `courses` state is null here, the
  // outer Suspense's fallback does NOT show -- the component renders
  // `null`, and Suspense never even finds out.
  useEffect(() => {
    fetch("http://localhost:3000/courses")
      .then((response) => response.json())
      .then((data) => setCourses(data));
  }, []);

  if (!courses) {
    // Not Suspense's fallback -- our own manual loading check.
    return null;
  }

  return (
    <ul>
      {courses.map((course) => (
        <li key={course.id}>{course.name}</li>
      ))}
    </ul>
  );
}

function SuspenseLimitationsExample() {
  return (
    <Suspense fallback={<p>This fallback never shows for CourseListWithEffect.</p>}>
      <CourseListWithEffect />
    </Suspense>
  );
}
```

API ve Veri Çekme kategorisindeki `useEffect` + `fetch` deseni, Suspense'i OTOMATİK OLARAK tetiklemez -- Suspense yalnızca `use()` gibi, React'in DOĞRUDAN tanıdığı bir Promise kaynağıyla çalışır. `useEffect` içinde `fetch` kullanan bir component, kendi `loading` state'ini KENDİSİ yönetmeye devam etmeli.

Özet ve Terimler Sözlüğü

`Suspense`, içindeki bir şey henüz hazır değilken bir `fallback` gösterir; içerik hazır olunca otomatik olarak değiştirilir. Birden fazla `Suspense`, farklı granülaritede yükleme durumları göstermek için iç içe kullanılabilir. React 19'daki `use()` hook'u, bir Promise'i Suspense ile entegre eder. Ama `useEffect` + `fetch` gibi "klasik" veri çekme desenleri Suspense'i OTOMATİK tetiklemez -- yalnızca `use()` gibi React'in doğrudan desteklediği kaynaklar tetikler.

Terimler Sözlüğü

Suspense — İçindeki bir kaynak (lazy component, Promise) henüz hazır değilken bir fallback UI gösteren React component'i.

Suspense Boundary (Suspense Sınırı) — Bir `<Suspense>` component'inin kapsadığı, kendi fallback'ine sahip alan.