Error Boundaries

React'te render sırasındaki hataları yakalayan class component'ler: getDerivedStateFromError, componentDidCatch, error boundary'lerin kapsamı ve yakalamadıkları, basit örneklerle.

İleri 13 dk

Son güncelleme: 18 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/error-boundaries

---

Error Boundaries

Bu derste, kursta İLK KEZ bir **class component** göreceğiz. React'te error boundary'leri (hata sınırları) hook'larla yazmanın bir yolu yok -- yalnızca class component'ler kullanılarak yazılabiliyor. Şimdiye kadar öğrendiğin her şey (hooks, state, props) fonksiyon component'lerle ilgiliydi; bu, istisnai ve dar kapsamlı bir konu.

Temel Bir Error Boundary Yazmak

Bir component render sırasında hata fırlattığında, React normalde TÜM uygulamayı "unmount" eder (boş bir ekran gösterir). Error boundary, bunu önler:

```
import { Component } from "react";

// IMPORTANT: Error boundaries are the FIRST class component we've seen in
// this course. There is NO way to write error boundaries with hooks
// (function components) in React -- only class components can do this,
// via `static getDerivedStateFromError`. That's why, for this ONE topic,
// we use a class component as an exception.
class BasicErrorBoundaryExample extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  // WHEN a child component THROWS an error during render, React calls
  // this method -- the value it returns becomes the new state.
  static getDerivedStateFromError() {
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      return <p>Something went wrong.</p>;
    }

    return this.props.children;
  }
}
```

`static getDerivedStateFromError()`, bir child hata fırlattığında React tarafından çağrılır -- döndürdüğü değer yeni state olur. `render()` metodu, `hasError` durumuna göre ya normal `children`'ı ya da bir fallback mesajı gösterir.

componentDidCatch ile Hatayı Loglamak

`getDerivedStateFromError`, YALNIZCA fallback UI'ı göstermek için kullanılır -- hatayı bir yere göndermek (loglamak) için ayrı bir metot gerekir:

```
import { Component } from "react";

class ComponentDidCatchExample extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError() {
    return { hasError: true };
  }

  // getDerivedStateFromError is used to SHOW the fallback UI;
  // componentDidCatch is used to SEND the error somewhere (e.g. to a
  // logging service) -- the two can work together.
  componentDidCatch(error, errorInfo) {
    console.error("Caught an error:", error, errorInfo.componentStack);
  }

  render() {
    if (this.state.hasError) {
      return <p>Something went wrong.</p>;
    }

    return this.props.children;
  }
}
```

`componentDidCatch(error, errorInfo)`, hatanın kendisini VE `errorInfo.componentStack`'i (hatanın hangi component'te olduğunu gösteren bir "yığın izi") alır -- gerçek uygulamalarda burada genellikle bir hata izleme servisine (Sentry gibi) bir istek atılır.

Error Boundary Kullanmak

Bir error boundary'i, hata fırlatabilecek component'leri sarmalamak için kullanırız:

```
import { Component, useState } from "react";

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError() {
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      return <p>Something went wrong.</p>;
    }

    return this.props.children;
  }
}

function BuggyCounter({ count }) {
  if (count === 3) {
    // We deliberately throw an error during render -- in a real app this
    // would be an unexpected error, like `undefined.someProperty`.
    throw new Error("Count reached 3!");
  }

  return <p>Count: {count}</p>;
}

function UsingErrorBoundaryExample() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      {/* If any component INSIDE it throws an error during render, the
          ErrorBoundary CATCHES that error and REPLACES the normal render
          with the fallback UI -- BuggyCounter itself doesn't have to
          handle the error. */}
      <ErrorBoundary>
        <BuggyCounter count={count} />
      </ErrorBoundary>
    </div>
  );
}
```

`BuggyCounter`, `count === 3` olduğunda bilinçli olarak bir hata fırlatıyor -- `ErrorBoundary` bunu yakalayıp normal render'ı fallback UI'la değiştiriyor. `BuggyCounter`'ın kendisi hatayı yönetmek zorunda DEĞİL, bu error boundary'nin işi.

Error Boundary'lerin Kapsamı

Birden fazla, KÜÇÜK error boundary kullanmak, tek bir büyük boundary'den genellikle daha iyidir:

```
import { Component } from "react";

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError() {
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      return <p>{this.props.fallbackText}</p>;
    }

    return this.props.children;
  }
}

function BuggyWidget() {
  throw new Error("This widget is broken!");
}

function ErrorBoundaryScopeExample() {
  return (
    <div>
      {/* Two SEPARATE ErrorBoundaries wrap two SEPARATE sections -- even if
          the Sidebar crashes, the MainContent is NOT AFFECTED, the crash
          stays CONTAINED within its own ErrorBoundary. If we used a single
          large ErrorBoundary, any error could turn the ENTIRE page into a
          "Something went wrong" message. */}
      <ErrorBoundary fallbackText="Sidebar failed to load.">
        <BuggyWidget />
      </ErrorBoundary>
      <ErrorBoundary fallbackText="Main content failed to load.">
        <p>Main content (still works fine)</p>
      </ErrorBoundary>
    </div>
  );
}
```

İki ayrı `ErrorBoundary`, iki ayrı bölümü sarmalıyor -- biri çökse bile, diğeri bundan ETKİLENMİYOR. Tek bir büyük boundary kullansaydık, herhangi bir hata TÜM sayfayı fallback mesajına çevirebilirdi.

Error Boundary'lerin Yakalamadığı Hatalar

Error boundary'lerin bir sınırı var -- her tür hatayı yakalamazlar:

```
import { Component, useState } from "react";

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError() {
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      return <p>Something went wrong.</p>;
    }

    return this.props.children;
  }
}

function EventHandlerExample() {
  function handleClick() {
    // Error boundaries only catch errors during RENDER -- an error thrown
    // INSIDE an event handler (like an onClick) is NOT CAUGHT by an error
    // boundary. A regular try/catch is needed for that.
    try {
      throw new Error("Button click failed!");
    } catch (error) {
      console.error("Caught manually:", error.message);
    }
  }

  return <button onClick={handleClick}>Click me</button>;
}

function WhatErrorBoundariesDontCatchExample() {
  const [showInfo, setShowInfo] = useState(false);

  return (
    <div>
      {/* The ErrorBoundary here wraps EventHandlerExample -- but the onClick
          error inside it will still NOT BE CAUGHT, because it happens in
          an event handler, not during render. */}
      <ErrorBoundary>
        <EventHandlerExample />
      </ErrorBoundary>
      <button onClick={() => setShowInfo(true)}>Show limitations</button>
      {showInfo && (
        <p>
          Error boundaries do NOT catch: event handler errors, errors in
          asynchronous code (setTimeout, fetch callbacks), errors during
          server-side rendering, or errors thrown in the boundary itself.
```

```
        </p>
      )}
    </div>
  );
}
```

Error boundary'ler yalnızca RENDER sırasındaki hataları yakalar -- event handler'lardaki (`onClick` gibi), asenkron kod içindeki (`setTimeout` , fetch callback'leri), sunucu tarafı render'daki, ya da boundary'nin KENDİSİNDE fırlatılan hataları YAKALAMAZLAR. Event handler'lardaki hatalar için normal `try` / `catch` kullanılır.

Özet ve Terimler Sözlüğü

Bir error boundary, `static getDerivedStateFromError` (fallback UI göstermek için) ve isteğe bağlı `componentDidCatch` (hatayı loglamak için) tanımlayan bir class component'tir -- React'te bunun hook karşılığı yoktur. Bir error boundary, İÇİNDEKİ herhangi bir component'in render sırasında fırlattığı hatayı yakalar ve normal render'ı bir fallback UI'la değiştirir. Birden fazla küçük boundary kullanmak, bir bölümdeki hatanın diğerlerini etkilemesini önler. Error boundary'ler event handler'lardaki, asenkron koddaki, ya da kendi içindeki hataları YAKALAMAZ.

Terimler Sözlüğü

Error Boundary (Hata Sınırı) — İçindeki component'lerin render sırasında fırlattığı hataları yakalayıp bir fallback UI gösteren class component.

Fallback UI — Bir hata (ya da yükleme durumu) sırasında, normal içerik yerine gösterilen alternatif arayüz.