useEffect

Side effect nedir, useEffect ile nasıl çalıştırılır, dependency array'in üç kullanımı, cleanup fonksiyonu ve yaygın infinite loop hatası -- basit örneklerle.

Orta 14 dk

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

---

useEffect

What Are Hooks? dersinde hook kavramını gördük. Bu ders, en sık kullanılan hook'lardan biriyle -- `useEffect` -- component'inin dışına çıkıp bir şeyler yapmasını (bir side effect) sağlamayı anlatıyor.

Side Effect (Yan Etki) Nedir?

Bir **side effect**, component'in kendi render çıktısı (yani döndürdüğü JSX) DIŞINDA yaptığı bir şeydir: tarayıcının sekme başlığını değiştirmek, bir zamanlayıcı (timer) kurmak, veri çekmek, `localStorage`'a yazmak gibi. `useEffect`, bu tür işleri güvenli bir şekilde yapmanı sağlayan hook.

Temel useEffect Kullanımı

`useEffect`'e bir fonksiyon verirsin; React bu fonksiyonu render bittikten SONRA çalıştırır:

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

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

  // A SIDE EFFECT: something a component does outside of its own render
  // output (JSX) -- here we're changing the browser's tab title.
  useEffect(() => {
    document.title = `Count: ${count}`;
  });

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

Burada her render'dan sonra sekme başlığını güncelliyoruz -- bu, JSX'in kendisinin yapamayacağı bir iş, çünkü `document.title`, ekrana çizilen component ağacının dışında, tarayıcının kendi bir özelliği.

Dependency Array: Boş Dizi []

`useEffect`'in ikinci parametresi, isteğe bağlı bir **dependency array** (bağımlılık dizisi). Boş bir dizi `[]` verirsen, effect yalnızca component ilk kez ekrana geldiğinde (mount olduğunda) bir kez çalışır:

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

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

  useEffect(() => {
    console.log("Component appeared on screen for the first time (mounted).");
  }, []); // Empty dependency array: runs only once, after the FIRST render.

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

Sonraki render'larda (state her değiştiğinde) bu effect BİR DAHA ÇALIŞMAZ -- yalnızca ilk render'dan sonra.

Dependency Array: Belirli Değerler

Dependency array'e belirli değerler koyarsan, effect yalnızca O DEĞERLER değiştiğinde çalışır:

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

function DependencyArrayExample() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("Alice");

  useEffect(() => {
    console.log("count changed:", count);
  }, [count]); // Only runs when count changes -- not when name changes.

  return (
    <div>
      <p>
        {name}, count: {count}
      </p>
      <button onClick={() => setCount(count + 1)}>Increment Count</button>
      <button onClick={() => setName(name === "Alice" ? "Bob" : "Alice")}>
        Change Name
      </button>
    </div>
  );
}
```

`[count]` yazdığımız için effect yalnızca `count` değiştiğinde çalışır -- `name` değişse bile effect tetiklenmez. React, her render'da dependency array'deki değerleri bir önceki render'la karşılaştırır; en az biri değiştiyse effect'i çalıştırır.

Cleanup Fonksiyonu

Bazı effect'ler (zamanlayıcı kurmak, bir event listener eklemek gibi) "temizlenmesi" gereken bir şey bırakır. `useEffect`'in verdiği fonksiyon bir CLEANUP fonksiyonu döndürebilir:

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

function CleanupFunctionExample() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);

    // Cleanup function: React automatically calls this when the
    // component is removed from the screen (unmount), or RIGHT BEFORE
    // the effect runs again. If we don't clear the interval here, it
    // keeps running in the background even after the component is
    // removed from the screen -- a "memory leak".
    return () => {
      clearInterval(intervalId);
    };
  }, []);

  return <p>Elapsed time: {seconds} seconds</p>;
}
```

React, cleanup fonksiyonunu component ekrandan kalktığında (unmount) ya da effect yeniden çalışmadan HEMEN ÖNCE otomatik çağırır. Burada `clearInterval` ile zamanlayıcıyı durdurmazsak, component ekrandan kalktıktan sonra bile arka planda çalışmaya devam eder.

Yaygın Hata: Infinite Loop

En yaygın `useEffect` hatası, dependency array'i unutup effect içinde state güncellemektir:

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

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

  // WRONG: a useEffect without a dependency array runs after EVERY
  // render. If state is updated inside it, that update triggers a new
  // render, which runs the effect again -- an INFINITE LOOP.
  // useEffect(() => {
  //   setCount(count + 1);
  // });

  // RIGHT: make the dependency array [] so it only runs on the first render.
  useEffect(() => {
    setCount((prev) => prev + 1);
  }, []);

  return <p>Count (should only increase once): {count}</p>;
}
```

Dependency array'siz bir `useEffect`, HER render'dan sonra çalışır. İçinde state güncellenirse, bu güncelleme yeni bir render tetikler, o render yine effect'i çalıştırır -- SONSUZ DÖNGÜ. Çözüm, dependency array'i doğru kurmak: yalnızca bir kez çalışması gerekiyorsa `[]`, belirli bir değere bağlıysa o değeri diziye koymak.

Özet ve Terimler Sözlüğü

`useEffect`, component'in render çıktısı dışında bir şey yapması (side effect) gerektiğinde kullanılır. Dependency array, effect'in NE ZAMAN çalışacağını belirler: `[]` yalnızca ilk render'da, `[deger]` yalnızca o değer değiştiğinde, hiç dependency array olmaması ise HER render'dan sonra. Cleanup fonksiyonu, effect'in bıraktığı şeyleri (timer, event listener gibi) temizler. Dependency array'i unutup effect içinde state güncellemek, en yaygın hata olan infinite loop'a yol açar.

Terimler Sözlüğü

Side Effect (Yan Etki) — Bir component'in render çıktısı dışında yaptığı bir şey (DOM'u doğrudan değiştirmek, veri çekmek gibi).

`useEffect` — Component'e side effect çalıştırma yeteneği ekleyen hook.

Dependency Array (Bağımlılık Dizisi) — `useEffect`'in ikinci parametresi; effect'in hangi değerler değiştiğinde yeniden çalışacağını belirler.

Cleanup Fonksiyonu — `useEffect`'in verdiği fonksiyondan döndürülen, effect'in bıraktığı şeyleri temizleyen fonksiyon; component unmount olduğunda ya da effect yeniden çalışmadan önce çağrılır.

Mount / Unmount — Bir component'in ekrana ilk kez gelmesi (mount) ve ekrandan tamamen kaldırılması (unmount).