

# Custom Hooks

Custom hook nedir, use öneki ve isimlendirme kuralı, aynı hook'u birden fazla kez kullanmak, ve bir useFetch örneği -- basit örneklerle.

Orta 10 dk

Oluşturulma tarihi: 5 Ekim 2026

Çevrimiçi oku: <https://www.learnforgex.com/tr/topics/custom-hooks>

## Custom Hooks

Bu kategoride `useState`, `useEffect`, `useRef`, `useMemo`, `useCallback` gibi React'in hazır hook'larını gördük. Bu son ders, kendi hook'unu yazmayı -- bir custom hook -- anlatıyor.

### Custom Hook Nedir?

Birden fazla component'te aynı state+effect mantığını tekrar tekrar yazdığını fark edersen, bu mantığı bir **custom hook**'a çıkarabilirsin. Custom hook, React'in kendi hook'larını ( `useState`, `useEffect` gibi) içeride kullanan, senin yazdığın normal bir fonksiyondur.

### use Öneki ve İsimlendirme Kuralı

Bir fonksiyonun custom hook sayılması için tek şart, isminin `use` ile başlaması:

```

import { useState } from "react";

// A custom hook is just a NORMAL JavaScript function that starts with "use" --
// inside it, you can call React's own hooks (like useState, useEffect).
function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  function increment() {
    setCount((prev) => prev + 1);
  }

  function decrement() {
    setCount((prev) => prev - 1);
  }

  return { count, increment, decrement };
}

function CustomHookNamingExample() {
  const { count, increment, decrement } = useCounter(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+1</button>
      <button onClick={decrement}>-1</button>
    </div>
  );
}

```

`useCounter`, içinde `useState` çağıran, senin yazdığın bir fonksiyon -- React'in kendi hook'larından biri değil. `use` öneki, hem React'e (Rules of Hooks kontrolleri için) hem de seni okuyan diğer geliştiricilere "bu bir hook, içinde başka hook'lar çağırabilir" sinyalini veriyor.

## Aynı Hook'u Birden Fazla Kez Kullanmak

Bir custom hook'u aynı component içinde birden fazla kez çağırabilirsin -- her çağrı kendi BAĞIMSIZ state'ine sahip olur:

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

function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);
  return {
    count,
    increment: () => setCount((prev) => prev + 1),
  };
}

function ReusingCustomHookExample() {
  // We call the same hook twice -- each call has its own INDEPENDENT state.
  const apples = useCounter(0);
  const oranges = useCounter(10);

  return (
    <div>
      <p>Apples: {apples.count}</p>
      <button onClick={apples.increment}>Add Apple</button>

      <p>Oranges: {oranges.count}</p>
      <button onClick={oranges.increment}>Add Orange</button>
    </div>
  );
}
```

`apples` ve `oranges`, aynı `useCounter` hook'unu kullanıyor, ama birbirinden tamamen bağımsız iki ayrı state. Bu, custom hook'ların gücü: state'i YÖNETME MANTIĞINI yeniden kullanıyorsun, state'in KENDİSİNİ değil.

## Örnek: useFetch ile Veri Çekme

Gerçek projelerde sık görülen bir custom hook örneği, veri çekme mantığını sarmalayan `useFetch`:

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

// Instead of writing useEffect + useState over and over, we extract the
// "data fetching" logic into a reusable custom hook. (A simplified
// example -- topics like error handling and race conditions will be
// covered in more detail in the "API & Data Fetching" category.)
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch(url)
      .then((response) => response.json())
      .then((json) => {
        setData(json);
        setLoading(false);
      });
  }, [url]);

  return { data, loading };
}

function UseFetchExample() {
  const { data, loading } = useFetch("https://api.example.com/users");

  if (loading) {
    return <p>Loading...</p>;
  }

  return <p>{data.length} users found.</p>;
}
```

`useFetch`, `useState` (veri ve yükleniyor durumu için) ile `useEffect` i (veriyi çekmek için) bir araya getiriyor ve bunları her component'te tekrar yazmak yerine tek bir yerde topluyor. Bu, basitleştirilmiş bir örnek -- hata yönetimi gibi konuları ileride "API & Data Fetching" kategorisinde daha detaylı göreceğiz.

## Özet ve Terimler Sözlüğü

Custom hook, `use` ile başlayan, içinde React'in kendi hook'larını kullanabilen, senin yazdığın normal bir fonksiyondur. Tekrar eden state+effect mantığını bir custom hook'a çıkarmak, aynı mantığı birden fazla component'te (ya da aynı component içinde birden fazla kez) bağımsız state'lerle yeniden kullanmanı sağlar.

### Terimler Sözlüğü

**Custom Hook** — `use` ile başlayan, içinde React'in kendi hook'larını kullanabilen, geliştiricinin kendi yazdığı fonksiyon.

**Kod Yeniden Kullanımı (Code Reuse)** — Aynı mantığı birden fazla yerde tekrar yazmak yerine, tek bir yerden (burada bir custom hook'tan) paylaşma.

## Pratik Proje

Bu kategoride (What Are Hooks?, useEffect, useRef, useMemo & useCallback, Custom Hooks) öğrendiğimiz kavramları bir arada kullanan, gerçek ve çalıştırılabilir bir örnek proje var: [Hooks Demo](#) -- tur (lap) kaydı yapabilen basit bir kronometre uygulaması.

Proje; bir custom hook ( `useStopwatch` ) içinde `useEffect` +`cleanup` ile `setInterval` yönetimini, `useRef` ile hem bir DOM elementine erişip otomatik kaydırma yapmayı hem de render'ı tetiklemeyen kalıcı bir sayaç tutmayı, `useMemo` ile en iyi turu yalnızca gerektiğinde yeniden hesaplamayı, ve `useCallback` ile bir fonksiyon referansını sabit tutmayı bir arada gösteriyor. Bilgisayarına indirip çalıştırabilir, kodunu satır satır inceleyebilirsin:

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects
npm install
cd projects/hooks
npm run dev
```

`react-course-projects` deposu **npm workspaces** kullanır -- `npm install` yalnızca bir kez, depo kökünde çalıştırılır ve tüm proje klasörleri ortak bağımlılıkları paylaşır (her klasörde ayrı bir `node_modules` oluşmaz). Kök dizinde `npm install` 'ı daha önce çalıştırdıysan, doğrudan `cd react-course-projects/projects/hooks` yapıp `npm run dev` demen yeterli.