Context API

React'in props drilling'e yerleşik çözümü Context API -- createContext, useContext, Provider, ve context'i bir custom hook'a sarmalamak, basit örneklerle.

Orta 13 dk

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

---

Context API

Sharing State dersinde, derin bir component ağacında props drilling'in nasıl bir soruna dönüştüğünü gördük. Bu ders, React'in bu soruna yerleşik çözümünü -- **Context API**'yi -- anlatıyor.

createContext ile Bir Context Oluşturmak

Context, ağacın herhangi bir yerinden, props geçirmeden okunabilecek bir değer taşıyan bir "kutu"dur:

```
import { createContext, useContext } from "react";

// createContext() creates a "box" -- it carries a value that can be read
// from anywhere in the tree, without passing props. The "light" in the
// parentheses is the DEFAULT value used when no Provider exists.
const ThemeContext = createContext("light");

function ThemedButton() {
  // useContext(ThemeContext) reads the `value` of the nearest
  // ThemeContext.Provider -- ThemedButton reads this value directly,
  // WITHOUT receiving it as a prop.
  const theme = useContext(ThemeContext);

  return <button className={theme}>Click me</button>;
}

function CreateContextExample() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedButton />
    </ThemeContext.Provider>
  );
}
```

`createContext("light")`, bir `ThemeContext` oluşturur -- parantez içindeki değer, bir Provider bulunmadığında kullanılacak VARSAYILAN değerdir. `<ThemeContext.Provider value="dark">`, içindeki tüm ağaç için bu değeri `"dark"` olarak EZER (override eder). `useContext(ThemeContext)`, en yakın Provider'ın değerini okur.

Provider Yokken: Varsayılan Değer

Provider her zaman gerekli değil -- bir Provider yoksa, `useContext` `createContext`'e verilen varsayılan değeri döner:

```
import { createContext, useContext } from "react";

const ThemeContext = createContext("light");

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>Click me</button>;
}

function DefaultValueExample() {
  // There's NO Provider here at all -- when ThemedButton calls
  // useContext(ThemeContext), it gets the DEFAULT value ("light") given
  // by createContext("light"). A Provider is only needed when we want to
  // OVERRIDE that value.
  return <ThemedButton />;
}
```

Bu örnekte hiç `Provider` yok; `ThemedButton`, `createContext("light")` ile verilen `"light"` değerini alıyor. Provider, yalnızca o varsayılanı DEĞİŞTİRMEK istediğinde gerekiyor.

Props Drilling'i Context ile Çözmek

Sharing State dersindeki derin ağaç örneğini, bu sefer Context ile yeniden yazalım:

```
import { createContext, useContext } from "react";

const UserContext = createContext(null);

function Level1() {
  return <Level2 />;
}

function Level2() {
  return <Level3 />;
}

function Level3() {
  return <Level4 />;
}

function Level4() {
  // Unlike the example in the Sharing State lesson, NONE of Level1/Level2/
  // Level3 even knows about the `user` prop -- only Level4, at the bottom,
  // reads DIRECTLY from UserContext.
  const user = useContext(UserContext);
  return <p>Logged in as {user}</p>;
}

function AvoidingPropsDrillingExample() {
  return (
    <UserContext.Provider value="Ada">
      <Level1 />
    </UserContext.Provider>
  );
}
```

`Level1`, `Level2`, `Level3`'ün HİÇBİRİ artık `user` prop'unu bilmiyor bile -- yalnızca en dipteki `Level4`, `UserContext`'ten DOĞRUDAN okuyor. Ara katmanlardan hiçbir şey geçirmemize gerek kalmadı.

Context İçinde State Taşımak

Context yalnızca sabit bir değer taşımaz -- state'in kendisini VE onu güncelleyen fonksiyonu birlikte taşıyabilir:

```
import { createContext, useContext, useState } from "react";

const CartContext = createContext(null);

function AddItemButton() {
  const { addItem } = useContext(CartContext);
  return <button onClick={() => addItem("Course")}>Add to Cart</button>;
}

function CartCount() {
  const { items } = useContext(CartContext);
  return <p>Items in cart: {items.length}</p>;
}

function ContextWithStateExample() {
  // Context doesn't only carry a FIXED value -- here we're giving `value`
  // an object: both the state itself (`items`) and the function that
  // UPDATES it (`addItem`). This is the most common way Context is used
  // in real applications.
  const [items, setItems] = useState([]);

  function addItem(item) {
    setItems([...items, item]);
  }

  return (
    <CartContext.Provider value={{ items, addItem }}>
      <AddItemButton />
      <CartCount />
    </CartContext.Provider>
  );
}
```

`value={{ items, addItem }}` ile Provider'a bir NESNE veriyoruz -- hem güncel `items` listesini hem de onu güncelleyen `addItem` fonksiyonunu. Bu, gerçek uygulamalarda en sık görülen Context kullanım biçimi.

Context'i Bir Custom Hook'a Sarmalamak

Hooks dersindeki custom hook deseni, Context ile birlikte çok yaygın kullanılan bir kalıp oluşturuyor:

```
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext(null);

function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

// The pattern from the Custom Hooks lesson continues here: instead of
// spreading useContext across components, we wrap it in a separate hook
// (`useTheme`). This way, the consuming component just calls `useTheme()`
// without ever dealing with the "context" concept -- just like `useState`.
function useTheme() {
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error("useTheme must be used within a ThemeProvider");
  }
  return context;
}

function ThemeToggleButton() {
  const { theme, setTheme } = useTheme();

  return (
    <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
      Current theme: {theme}
    </button>
  );
}

function CustomContextHookExample() {
  return (
    <ThemeProvider>
      <ThemeToggleButton />
    </ThemeProvider>
  );
}
```

`useTheme()`, `useContext(ThemeContext)`'i SARMALAYIP daha temiz bir API sunuyor -- kullanan component, "context" kavramıyla hiç uğraşmadan `useTheme()` çağırıyor, tıpkı yerleşik bir hook gibi. Provider dışında kullanılırsa hata fırlatması, yanlış kullanımı erken yakalamamızı sağlıyor.

Özet ve Terimler Sözlüğü

`createContext()`, ağacın herhangi bir yerinden okunabilecek bir değer oluşturur; `Provider`, bu değeri belirli bir alt ağaç için EZER; `useContext()`, en yakın Provider'ın değerini okur.

Provider yoksa, `createContext`'e verilen varsayılan değer kullanılır. Context, sabit bir değer taşıyabildiği gibi, state VE onu güncelleyen fonksiyonları birlikte de taşıyabilir -- bu, `useContext`'i bir custom hook'a sarmalamakla (`useTheme()` gibi) birleştiğinde, React'te "global" sayılabilecek state'i yönetmenin en yaygın yoludur.

Terimler Sözlüğü

Context — Ağacın herhangi bir yerinden, props geçirmeden okunabilen bir değer taşıyan React yapısı.

Provider — Bir Context'in değerini, kendi altındaki tüm ağaç için belirleyen component.

Pratik Proje

Bu kategoride (Sharing State, Context API) öğrendiğimiz kavramları bir arada kullanan, gerçek ve çalıştırılabilir bir örnek proje var: **State Management Demo** -- arama ile filtrelenebilen, favorilere eklenebilen bir kurs listesi uygulaması.

Proje; arama metnini ortak parent'ta tutup (lifting state up) hem arama kutusuna hem sonuç listesine props ile aktarmayı, favori kursları bir `FavoritesContext` ile (createContext + Provider + custom hook) HİÇBİR props drilling olmadan ağacın farklı yerlerinden (kurs listesi VE başlıktaki favori sayacı) okumayı 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/state-management
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/state-management` yapıp `npm run dev` demen yeterli.