Sharing State

Birden fazla component'in aynı state'e ihtiyaç duyduğu durumlar -- state'i yukarı taşımak (lifting state up) ve props drilling sorunu, basit örneklerle.

Orta 11 dk

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

---

Sharing State

Şimdiye kadar `useState`'i hep TEK bir component'in içinde kullandık. Bu ders, birden fazla component'in AYNI state'e ihtiyaç duyduğu durumları -- ve bunun beraberinde getirdiği "props drilling" sorununu -- anlatıyor.

İki Component, Aynı State'e İhtiyaç Duyduğunda

Bir arama kutusu (`SearchBox`) ve bir sonuç listesi (`ResultsList`) düşün -- ikisi de AYNI arama metnine ihtiyaç duyar. `query` state'i `SearchBox`'ın içinde tutulursa ne olur?

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

function SearchBox() {
  const [query, setQuery] = useState("");

  return (
    <input
      type="text"
      value={query}
      onChange={(event) => setQuery(event.target.value)}
      placeholder="Search courses..."
    />
  );
}

function ResultsList() {
  // Problem: this component needs to know what the user typed into
  // SearchBox -- but the `query` state is trapped INSIDE SearchBox and
  // there is no way to reach it from here.
  return <p>Type in the search box above -- but this list has no way to know what you typed.
</p>;
}

function SeparateStateProblemExample() {
  return (
    <div>
      <SearchBox />
      <ResultsList />
    </div>
  );
}
```

`ResultsList`'in, `SearchBox`'ın state'ine erişmesinin hiçbir yolu yok -- her component'in kendi state'i kendi İÇİNE hapsolmuştur, kardeş component'ler (sibling'ler) birbirinin state'ini DOĞRUDAN göremez.

State'i Yukarı Taşımak: Lifting State Up

Çözüm, state'i iki component'in de ORTAK atası olan bir yere taşımak:

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

const courses = ["Java", "React", "Spring Boot"];

function SearchBox({ query, onQueryChange }) {
  return (
    <input
      type="text"
      value={query}
      onChange={(event) => onQueryChange(event.target.value)}
      placeholder="Search courses..."
    />
  );
}

function ResultsList({ query }) {
  const filtered = courses.filter((course) =>
    course.toLowerCase().includes(query.toLowerCase()),
  );

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

function LiftingStateUpExample() {
  // The `query` state no longer lives INSIDE SearchBox -- it now lives in
  // this component, the COMMON ancestor of both. This is what we call
  // "lifting state up". Both children receive this state via props.
  const [query, setQuery] = useState("");

  return (
    <div>
      <SearchBox query={query} onQueryChange={setQuery} />
      <ResultsList query={query} />
    </div>
  );
}
```

`query` state'i artık ortak parent'ta yaşıyor; her iki child da bunu props ile alıyor -- `SearchBox`, `query` ve `onQueryChange`'i, `ResultsList` ise yalnızca `query`'i. Bu desene **state'i yukarı taşımak** (lifting state up) denir -- React'te paylaşılan state'i yönetmenin en temel yolu budur.

Aynı Deseni Farklı Bir Senaryoda Görmek

Lifting state up, yalnızca liste filtrelemekle sınırlı değil -- aynı değeri İKİ FARKLI şekilde gösteren component'ler için de geçerli:

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

function SliderInput({ rating, onRatingChange }) {
  return (
    <input
      type="range"
      min="0"
      max="5"
      value={rating}
      onChange={(event) => onRatingChange(Number(event.target.value))}
    />
  );
}

function RatingDisplay({ rating }) {
  return <p>Rating: {rating} / 5</p>;
}

function SyncedSiblingsExample() {
  // Two sibling components with different views (a slider and a text
  // display) represent the SAME value -- since the state has been
  // moved to their common ancestor, both always stay in sync.
  const [rating, setRating] = useState(0);

  return (
    <div>
      <SliderInput rating={rating} onRatingChange={setRating} />
      <RatingDisplay rating={rating} />
    </div>
  );
}
```

Bir slider ve bir metin gösterimi, AYNI `rating` değerini paylaşıyor -- state ortak parent'ta olduğu için, biri değiştiğinde diğeri de anında güncel kalıyor.

Props Drilling: Ara Katmanlardan Geçirmek

Component ağacı derinleştikçe, bir prop'u ihtiyaç duyan component'e ULAŞTIRMAK için, aralarındaki component'lerin de o prop'u almasını gerektirebilir:

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

const courses = ["Java", "React", "Spring Boot"];

function SearchBox({ query, onQueryChange }) {
  return (
    <input
      type="text"
      value={query}
      onChange={(event) => onQueryChange(event.target.value)}
      placeholder="Search courses..."
    />
  );
}

function ResultsList({ query }) {
  const filtered = courses.filter((course) =>
    course.toLowerCase().includes(query.toLowerCase()),
  );

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

function ResultsPanel({ query }) {
  // ResultsPanel never uses `query` ITSELF -- it only receives it to
  // PASS IT ALONG to ResultsList. This is a simple example of what we
  // call "props drilling" (passing props through intermediate layers
  // that don't need them).
  return (
    <div className="panel">
      <ResultsList query={query} />
    </div>
  );
}

function PropsDrillingExample() {
  const [query, setQuery] = useState("");

  return (
    <div>
      <SearchBox query={query} onQueryChange={setQuery} />
      <ResultsPanel query={query} />
    </div>
  );
}
```

`ResultsPanel`, `query`'i KENDİSİ hiç kullanmıyor -- yalnızca `ResultsList`'e iletmek için alıyor. Buna **props drilling** denir -- bir prop'un, kullanmayan ara katmanlardan zorunlu olarak geçirilmesi.

Props Drilling Neden Sorun Yaratır?

Ağaç derinleştikçe bu sorun büyür:

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

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

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

function Level4({ user }) {
  return <p>Logged in as {user}</p>;
}

function WhyPropsDrillingHurtsExample() {
  // NONE of Level1, Level2, or Level3 use `user` -- they just PASS IT
  // ALONG to the next level. Only Level4, at the very bottom, actually
  // uses it. As the tree gets deeper (or new props are added at every
  // level) this becomes tedious to write and error-prone -- in the next
  // lesson (Context API) we'll see an approach that solves this.
  return <Level1 user="Ada" />;
}
```

`Level1`, `Level2`, `Level3`'ün HİÇBİRİ `user`'ı kullanmıyor -- yalnızca aktarıyorlar. Yalnızca en dipteki `Level4` gerçekten kullanıyor. Her yeni seviye, ya da her yeni paylaşılan değer, bu zinciri daha da uzatır -- yazması yorucu ve hataya açık hale gelir. Bir sonraki derste (Context API), bunu çözen bir yöntem göreceğiz.

Özet ve Terimler Sözlüğü

Birden fazla component aynı state'e ihtiyaç duyduğunda, state'i ORTAK ataya taşımak (lifting state up) gerekir -- her child bu state'i props ile alır. Ağaç derinleştikçe, bir prop'u kullanmayan ara component'lerden geçirmek zorunda kalmak "props drilling" sorununu doğurur -- bu, kodu yazması yorucu ve değiştirmesi kırılgan hale getirir.

Terimler Sözlüğü

Lifting State Up (State'i Yukarı Taşımak) — Birden fazla component'in paylaştığı bir state'i, bu component'lerin ORTAK atasına taşımak.

Props Drilling — Bir prop'u, kullanmayan ara katman component'lerden zorunlu olarak geçirmek.