React Global State

Lernziele

- [ ] Verständnis von Prop Drilling
- [ ] Kenntnis von Namenskonventionen bei Prop Drilling
- [ ] Verständnis des Konzepts des globalen Zustands (Global State)

State anheben (Lifting State Up)

In vielen Situationen wird ein bereits existierender State in einer Komponente auch in einer anderen Komponente benötigt. Die Komponenten müssen einen gemeinsamen State teilen. Um dies zu lösen, muss der State in der Komponenten-Hierarchie nach oben verschoben werden, bis zur ersten gemeinsamen Elternkomponente. Dies nennt man "Lifting State Up". Von diesem Punkt aus werden State-Variablen oder Funktionen zum Ändern des States an untergeordnete Komponenten über Props weitergegeben.

Weitere Informationen zu diesem Thema findest du im React State 2 Handout.

Prop Drilling

Einzelne Komponenten, die State-Variablen konsumieren, und ihr gemeinsamer Vorfahre, in dem der State definiert ist, können in der Komponenten-Hierarchie weit voneinander entfernt sein. Die Konsequenz ist, dass State-Variablen über mehrere andere Komponenten hinweg via Props weitergereicht werden müssen, bis sie in der Zielkomponente ankommen. Dies wird als "Prop Drilling" bezeichnet.

Betrachte folgendes Beispiel:

```
function App() {
  const [userIsLoggedIn, setUserIsLoggedIn] = useState(false);

  return <ProductsPage userIsLoggedIn={userIsLoggedIn} />;
}

function ProductsPage({ userIsLoggedIn }) {
  return <ProductsList userIsLoggedIn={userIsLoggedIn} />;
}

function ProductsList({ userIsLoggedIn }) {
  return products.map((product) => (
    <ProductCard {...product} userIsLoggedIn={userIsLoggedIn} />
  ));
}

function ProductCard({ userIsLoggedIn }) {
  return <ProductActions userIsLoggedIn={userIsLoggedIn} />;
}
```

```
function ProductActions({ userIsLoggedIn }) {
  return userIsLoggedIn ? (
    <button>One-click Buy</button>
  ) : (
    <button>Add to Basket</button>
  );
}
```

Die State-Variable `userIsLoggedIn` ist in der `App`-Komponente definiert. Sie wird vier Mal weitergereicht, bis sie in der `ProductActions`-Komponente genutzt werden kann.

Stell dir vor, es gäbe noch viele weitere Komponenten in dieser App, von denen einige ebenfalls wissen müssen, ob ein Benutzer eingeloggt ist oder nicht.

Prop Drilling über ein paar Ebenen ist völlig in Ordnung. Wenn der Pfad jedoch länger wird und mehrere State-Variablen über Props weitergegeben werden, steigt die Komplexität und die Wartbarkeit des Codes nimmt ab. Auf dem Weg darf das Weitergeben jedes Props in keiner Komponente vergessen werden.

Namenskonventionen für Props und Funktionen

Im React Props Handout findest du allgemeine Informationen zur Benennung von Variablen und Funktionen, die über Props weitergegeben werden.

Im Kontext von Prop Drilling sei vorsichtig, Props nicht auf halbem Wege umzubenennen. Wenn ein Prop umbenannt wird, kann die logische Referenz verloren gehen, was den Code schwerer verständlich macht.

Obwohl wir empfehlen, Funktionsnamen mit `handle` und entsprechende Props mit `on` zu versehen, musst du nicht in jeder Komponente entlang des Weges die durchgereichten Props umbenennen.

Eine Lösung wie diese ist leichter zu lesen:

```
function App() {
  const [userIsLoggedIn, setUserIsLoggedIn] = useState(false);

  function handleLogIn() {
    setUserIsLoggedIn(true);
  }

  return <Layout handleLogIn={handleLogIn} />;
}

function Layout({ handleLogIn }) {
  return <Header onLogIn={handleLogIn} />;
}

function Header({ onLogIn }) {
  return <button onClick={onLogIn}>Log In</button>;
}
```

State-Management-Bibliotheken

Im React-Ökosystem haben sich verschiedene Bibliotheken entwickelt, um die Handhabung von komplexem State zu vereinfachen. Um starkes Prop Drilling zu vermeiden, ist das Konzept des "globalen States" eine etablierte Lösung.

Die Idee: State wird nicht in einer Komponente definiert und über Props weitergereicht, sondern außerhalb der Komponenten-Hierarchie. Jede Komponente kann auf den globalen State zugreifen.

Es gibt viele Bibliotheken, die Implementierungen für globalen State anbieten. Wir empfehlen "zustand". Die verschiedenen Bibliotheken unterscheiden sich in einigen Details, aber die Idee des globalen States ist gleich.

Mit "zustand" würde das obige Beispiel so aussehen. Das Weiterreichen von `userIsLoggedIn` ist nicht mehr nötig.

```
import { create } from "zustand";

const useUserStore = create((set) => ({
  isLoggedIn: false,
  logIn: () => set(() => ({ isLoggedIn: true })),
  logOut: () => set(() => ({ isLoggedIn: false })),
}));

function App() {
  return <ProductsPage />;
}

function ProductsPage() {
  return <ProductsList />;
}

function ProductsList() {
  return products.map((product) => <ProductCard {...product} />);
}

function ProductCard() {
  return <ProductActions />;
}

function ProductActions() {
  const userIsLoggedIn = useUserStore((state) => state.isLoggedIn);

  return userIsLoggedIn ? (
    <button>One-click Buy</button>
  ) : (
    <button>Add to Basket</button>
  );
}
```

> ! Die Verwendung von globalem State bedeutet nicht, dass du überall in deiner App Prop Drilling vermeiden solltest. Es wäre ein Anti-Pattern, alle Daten im globalen State zu speichern.
>
> Überlege sorgfältig, welche Daten wirklich global für viele Komponenten verfügbar sein müssen. Nur solche Daten sollten im globalen State gespeichert werden.
>
> Wenn sich der State nur auf einen Teil deiner App bezieht, sollte er lokal in einer Komponente definiert und über Props weitergegeben werden.

Resources

- zustand on GitHub