React Nesting

Lernziele

- Verstehen des Konzepts des Verschachtelns
- Erstellen mehrerer benutzerdefinierter Komponenten, um eine Hierarchie zu erstellen
- Verwenden der `children`-Prop, um JSX vom übergeordneten Component zu rendern
- Verstehen von Komposition als eine Möglichkeit, komplexe Komponenten zu erstellen

---

JSX als Props übergeben

Von JSX erstellte Elemente sind lediglich Objekte. Sie können wie jedes andere Objekt weitergegeben werden: zum Beispiel als Props.

```
function UserCard({ avatar }) {
  return <div className="card">{avatar}</div>;
}
```

```
function App() {
  return <UserCard avatar={<Avatar />} />;
}
```

Die `children`-Prop

Sie sind bereits mit dem Verschachteln von eingebauten Browser-Tags vertraut:

```
<div>
  <img />
</div>
```

Oftmals möchten Sie, dass Ihre eigenen Komponenten ebenfalls verschachtelbar sind.

```
<UserCard>
  <Avatar />
</UserCard>
```

Wenn Sie eine Komponente in eine andere Komponente verschachteln, wird die verschachtelte Komponente als Prop an die übergeordnete Komponente übergeben. Diese spezielle Prop wird `children` genannt.

```
function UserCard({ children }) {
  return <div className="card">{children}</div>;
}
```

Diese Komponente wird das/die verschachtelte(n) Element(e) als Kind des `div`-Elements rendern.

> 💡 Das/die verschachtelte(n) Element(e) kann/können ein einzelnes Element, mehrere Elemente oder sogar eine Zeichenkette oder Zahl sein.

> 📙 Lesen Sie mehr über **JSX als Kinder übergeben** in den React Docs.

Fragments

Manchmal möchten Sie mehrere Elemente aus einer Component-Funktion zurückgeben, ohne sie in ein `div` oder ein anderes Element zu verpacken. Dafür können Sie ein `Fragment` (`<></>` oder `<Fragment></Fragment>`) verwenden.

Dies ist notwendig, da React-Komponenten nur ein einziges Element aus einer Component-Funktion zurückgeben können.

```
function UserList() {
  return (
    <>
      <UserCard>
        <Avatar />
      </UserCard>
      <UserCard>
        <Avatar />
      </UserCard>
    </>
  );
}
```

Dies ist äquivalent zu folgendem, aber die obige Kurzversion wird allgemein bevorzugt.

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

function UserList() {
  return (
    <Fragment>
      <UserCard>
        <Avatar />
      </UserCard>
      <UserCard>
        <Avatar />
      </UserCard>
    </Fragment>
```

```
    );
}
```

> 💡 Die `<Fragment></Fragment>`-Syntax ist nur dann notwendig, wenn Sie die spezielle `key`-Prop an das Fragment übergeben möchten, was wichtig wird, wenn Sie mit Listen arbeiten.

> 💡 Bei der Recherche könnten Sie manchmal `<React.Fragment></React.Fragment>` sehen, was dasselbe ist.

> 📙 Lesen Sie mehr über **Fragment (<>...</>)** in den React Docs.

Komposition

Wenn wir React-Anwendungen erstellen, möchten wir oft komplexe Komponenten aus einfacheren Komponenten aufbauen. Dies wird als Komposition bezeichnet.

Dazu müssen Sie Ihre Anwendung in Komponenten unterteilen. Diese Komponenten können dann zusammengesetzt werden, um komplexere Komponenten zu erstellen.

Es ist wichtig herauszufinden, welche Komponenten Sie benötigen und wie sie zusammengesetzt werden sollten. Dies wird als Anwendungsdesign bezeichnet.

> 📙 Lesen Sie **Thinking in React** in den React Docs bis einschließlich Schritt 2. Spätere Schritte erfordern State, den wir in einer zukünftigen Sitzung behandeln werden.

Resources

- Passing JSX as children in the React Docs
- Fragment (<>...</>) in the React Docs
- Thinking in React in the React Docs