React mit Arrays

Lernziele

- [ ] Wissen, wie man `.map()` verwendet, um Listen in JSX zu rendern
- [ ] Verstehen, wie man Elemente aus einem Array von Objekten rendert
- [ ] Wissen, dass man eine eindeutige `key`-Eigenschaft für Listenelemente hinzufügen muss

---

Arrays in JSX

Um Elemente aus einem Array in React zu rendern, verwenden wir die Array-Methode `.map()`.

Die Array-Methode `.map()` wird verwendet, um eine Transformation auf alle Elemente eines Arrays anzuwenden. Beim Rendern eines Arrays zu JSX möchten wir genau dies tun. Wir möchten jedes Element eines Arrays in ein JSX-Tag transformieren. Aus diesem Grund verwenden wir `.map()`.

```
function Drinks() {
  const drinks = ["water", "lemonade", "coffee", "tee"];

  return (
    <ul>
      {drinks.map((drink) => (
        <li>{drink}</li>
      ))}
    </ul>
  );
}
```

Key-Eigenschaft

Das obige Beispiel fehlt ein kleines, aber sehr wichtiges Detail: die `key`-Eigenschaft!

Ohne die `key`-Eigenschaft wirst du eine Fehlermeldung in der Konsole sehen:

```
Warning: Each child in a list should have a unique "key" prop.
```

Beim Rendern eines Arrays in JSX musst du einen **eindeutigen Bezeichner(unique identifier)** als Wert für die `key`-Eigenschaft des ersten JSX-Tags, das in `.map()` zurückgegeben wird, übergeben. Dies ist wichtig, damit React die Änderungen, die bei einem erneuten Rendern an den Daten vorgenommen werden, nachverfolgen kann.

Daher musst du immer sicherstellen, dass dein Array eine eindeutige ID pro Element enthält. Dies kannst du gewährleisten, indem du Objekte zur Definition der Daten in deinen Arrays verwendest.

```
function Drinks() {
  const drinks = [
    { id: 0, name: "water" },
    { id: 1, name: "lemonade" },
    { id: 2, name: "coffee" },
    { id: 3, name: "tea" },
  ];

  return (
    <ul>
      {drinks.map(({ id, name }) => (
        <li key={id}>{name}</li>
      ))}
    </ul>
  );
}
```

📙 Wenn du daran interessiert bist, die Details dahinter zu verstehen, kannst du in den React-Dokumenten mehr über **die key-Eigenschaft lesen.

💡 Wenn du die `key`-Eigenschaft an eine Komponente übergibst, kannst du in der Komponente nicht darauf zugreifen. Es ist eine spezielle Eigenschaft, die React nur intern >verwendet.

```
function Drink({ name, key }) {
  console.log(key); // → undefined
  return <li>{name}</li>;
}

function Drinks() {
  const drinks = [
    { id: 0, name: "water" },
    { id: 1, name: "lemonade" },
    { id: 2, name: "coffee" },
    { id: 3, name: "tea" },
  ];

  return (
    <ul>
      {drinks.map(({ id, name }) => (
        <Drink key={id} name={name} />
      ))}
    </ul>
  );
}
```

Wenn du in diesem Beispiel auf die `id` zugreifen möchtest, kannst du sie erneut als Prop übergeben: `<Drink key={id} id={id} name={name} />`.

Keyed Fragments

Wenn du eine Liste von Elementen renderst, die nicht in einem einzelnen JSX-Tag eingeschlossen sind, kannst du ein `<Fragment>` verwenden, um die Elemente zu umschließen.

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

function Drinks() {
  const drinks = [
    { id: 0, name: "water", description: "very wet" },
    { id: 1, name: "lemonade", description: "quite sweet" },
    { id: 2, name: "coffee", description: "cold brew" },
    { id: 3, name: "tea", description: "earl grey, hot" },
  ];

  return (
    <dl>
      {drinks.map(({ id, name, description }) => (
        <Fragment key={id}>
          <dt>{name}</dt>
          <dd>{description}</dd>
        </Fragment>
      ))}
    </dl>
  );
}
```

> 💡 Hier kannst du nicht die Kurzsyntax (`<>…</>`) für das `<Fragment>` verwenden, weil du die `key`-Eigenschaft an das `<Fragment>` übergeben musst. Die Kurzsyntax erlaubt es nicht, > Props zu übergeben.

Resources

- React Docs: Rendering Lists