React mit Local Storage

Lernziele

- [ ] Das Konzept von persistentem Speicher im Browser verstehen
- [ ] Den Unterschied zwischen `localStorage` und `sessionStorage` kennen
- [ ] Die Methoden `setItem()` und `getItem()` anwenden
- [ ] Eine Bibliothek verwenden, um Local Storage in React-Apps zu nutzen

---

Die Web Storage API

> 💡 Beachte, dass die Web Storage API nicht Teil von React ist. Es handelt sich um eine Browser-API, die in allen modernen Browsern verfügbar ist.

Die Web Storage API bietet zwei Methoden, um Daten auf dem Client zu speichern:

- `localStorage` speichert Daten ohne Ablaufdatum
- `sessionStorage` speichert Daten für eine Sitzung (Daten gehen verloren, wenn der Browser-Tab geschlossen wird)

Daten werden im Browser und pro Domain gespeichert, d.h. alle Daten, die von `example.com` gespeichert werden, können von `www.example.com` und `subdomain.example.com` abgerufen werden, aber nicht von `others.org`.

Dies ermöglicht es, Daten über Seiten-Neuladevorgänge und Browser-Neustarts hinweg sicher zu speichern.

Zum Speichern von Daten verwendet die API Schlüssel-Wert-Paare. Der Schlüssel ist ein String und der Wert kann ein String, eine Zahl oder ein Boolean sein.

> 💡 Alle folgenden Beispiele verwenden `localStorage`, aber das Gleiche gilt für `sessionStorage`.

> 📙 Mehr dazu in den **Web Storage API** auf den mdn web docs.

Daten speichern

Um Daten zu speichern, verwendet man die Methode `setItem()`:

```
localStorage.setItem("name", "Alex");
localStorage.setItem("age", 28);
localStorage.setItem("isOnline", true);
Daten abrufen
Um Daten abzurufen, verwendet man die Methode getItem():
```

```
const name = localStorage.getItem("name"); // → "Alex"
const age = localStorage.getItem("age"); // → 28
```

```
const isOnline = localStorage.getItem("isOnline"); // → true
```

Wenn getItem aufgerufen wird und der Schlüssel nicht existiert, wird null zurückgegeben.

```
const nope = localStorage.getItem("nope"); // → null
```

Daten entfernen

Um Daten zu entfernen, verwendet man die Methode `removeItem()`:

```
localStorage.removeItem("name");
```

Alle Daten löschen

Um alle Daten zu entfernen, verwendet man die Methode `clear()`:

```
localStorage.clear();
```

Komplexe Daten speichern

Die Web Storage API unterstützt nur Strings, Zahlen und Booleans. Um komplexere Daten zu speichern, müssen diese zuerst serialisiert werden. Dies kann mit der Methode `JSON.stringify()` erfolgen:

```
const user = {
  name: "Alex",
  age: 28,
  isOnline: true,
};

localStorage.setItem("user", JSON.stringify(user));
```

Um die Daten abzurufen, muss man sie mit der Methode `JSON.parse()` parsen:

```
const user = JSON.parse(localStorage.getItem("user"));
```

Hilfsfunktionen

Um die Arbeit mit der Web Storage API zu erleichtern, kann man Hilfsfunktionen erstellen, die die Serialisierung und Deserialisierung kapseln:

```
// Daten speichern
function setItem(key, value) {
  localStorage.setItem(key, JSON.stringify(value));
}

// Daten abrufen
function getItem(key) {
  return JSON.parse(localStorage.getItem(key));
}
```

Diese Funktionen funktionieren sowohl mit einfachen Datentypen wie Strings und Zahlen als auch mit komplexen Datentypen:

```
setItem("user", {
  name: "Alex",
  age: 28,
  isOnline: true,
});
setItem("count", 42);

const user = getItem("user");
const count = getItem("count");
```

React mit Local Storage

Du kannst die Web Storage API auch in React verwenden. Am häufigsten möchtest du den Zustand in local storage speichern, damit er Seiten-Neuladevorgänge überlebt.

React bietet verschiedene Möglichkeiten, den Zustand mit local storage zu synchronisieren. Das allgemeine Konzept besteht darin, den Anfangszustand aus local storage abzurufen und den Zustand jedes Mal im local storage zu speichern, wenn er sich ändert.

Da es ziemlich kompliziert wird, all diese verschiedenen Teile selbst korrekt zu verkabeln, solltest du eine Bibliothek verwenden, die dafür einen Hook bereitstellt.

`use-local-storage-state`

Die `use-local-storage-state` Bibliothek bietet einen Hook, der es dir ermöglicht, den Zustand in local storage zu speichern.

Du kannst ihn als Ersatz für den `useState` Hook verwenden (im untenstehenden Beispiel auskommentiert):

```
// import { useState } from "react";
import useLocalStorageState from "use-local-storage-state";

function Counter() {
  // const [count, setCount] = useState(0);
  const [count, setCount] = useLocalStorageState("count", { defaultValue:
```

```
  0 });

    return (
      <div>
        <p>Count: {count}</p>
        <button onClick={() => setCount(count + 1)}>Increment</button>
      </div>
    );
  }
```

> 💡 Beachte, dass das erste Argument des `useLocalStorageState` Hooks der Schlüssel ist, der verwendet wird, um den Zustand im local storage zu speichern. Wenn du denselben Schlüssel für mehrere Komponenten verwendest, teilen sie sich denselben Zustand.

> 💡 Du musst dich bei `use-local-storage-state` nicht selbst um die Serialisierung oder das Parsen von komplexen Daten kümmern. Die Bibliothek erledigt das im Hintergrund für dich.

> 📙 Mehr dazu, **wie man den `use-local-storage-state` Hook verwendet**, findest du in den Dokumenten.

React Custom Hooks

Der `useLocalStorageState` Hook ist nicht Teil von React selbst. Es ist ein benutzerdefinierter Hook, der von einer Drittanbieter-Bibliothek bereitgestellt wird. Genauso wie der Autor von `use-local-storage-state` einen benutzerdefinierten Hook erstellt hat, kannst du deine eigenen benutzerdefinierten Hooks erstellen, um gemeinsame Logik, die andere Hooks verwendet (`useState`, `useEffect`, …), zu abstrahieren.

> 📄 Erfahre mehr über **React Custom Hooks** in dem vorbereiteten Dokument (mit Beispielen).

Resources

- Web Storage API on the mdn web docs
- use-local-storage-state on GitHub
- 📄 React Custom Hooks document