React State 3

Lernziele

- [ ] Wissen, wie man Arrays und Objekte im State behandelt
- [ ] Verstehen, was State-Mutation ist und wie man sie vermeidet

---

Vermeidung von State-Mutation

Unabhängig davon, wie komplex der State in deiner Anwendung ist (Objekt, Array, Array von Objekten), musst du den State immer als unveränderlich behandeln. Das bedeutet, dass du den State nicht direkt mutieren solltest, z.B. durch das Zuweisen eines neuen Wertes.

Um eine Mutation bei der Aktualisierung des States zu vermeiden, musst du

1. ein neues Objekt / Array erstellen (oder eine Kopie des vorhandenen erstellen) und
2. die Setter-Funktion mit der neu erstellten / aktualisierten Kopie verwenden, um ein Re-Rendern auszulösen.

Aktualisierung von Objekten im State

Um eine Kopie eines Objekts zu erstellen und nur einige Eigenschaften zu ändern, kannst du die Spread-Syntax verwenden:

```
const [person, setPerson] = useState({
  firstName: "John",
  lastName: "Doe",
});

function handleChangeFirstName(firstName) {
  setPerson({ ...person, firstName });
}

// Irgendwo anders:
handleChangeFirstName("Jane");
```

> ! Wenn du einen neuen Wert direkt zuweisen würdest, würdest du den State mutieren. Das ist schlecht, weil wir den State als unveränderlich behandeln müssen. Dadurch können schwer zu findende Bugs entstehen.

```
// ⚠ DAS NIEMALS TUN
function handleChangeFirstName(firstName) {
  person.firstName = firstName;
  setPerson(person);
}
```

> 📙 Erfahre mehr über **Aktualisierung von Objekten im State** in den React Docs.

Aktualisierung von Arrays im State

Wie du weißt, gibt es mehrere Möglichkeiten, Arrays zu aktualisieren. Einige davon mutieren das Array, andere nicht.

	vermeiden (mutiert das Array)	bevorzugen (gibt ein neues Array zurück)
adding	`push`, `unshift`	`[...arr]` spread syntax
removing	`pop`, `shift`, `splice`	`filter`
replacing	`splice`, `arr[i] = ...` assignment	`map`
sorting	`reverse`, `sort`	copy the array first

> 💡 Es spielt keine Rolle, ob dein Array-State nur primitive Werte oder andere Objekte / Arrays enthält; in allen Fällen solltest du nur die bevorzugten Array-Methoden verwenden.

Hinzufügen zu einem Array

Um ein Element zu einem Array hinzuzufügen, kannst du die Spread-Syntax verwenden:

```
const [numbers, setNumbers] = useState([0, 1, 2]);

function handleAppendNumber(number) {
  setNumbers([...numbers, number]);
}

// Irgendwo anders:
handleAppendNumber(3);

function handlePrependNumber(number) {
  setNumbers([number, ...numbers]);
}

// Irgendwo anders:
handlePrependNumber(-1);
```

Entfernen aus einem Array

Um ein Element aus einem Array zu entfernen, kannst du die `filter`-Methode verwenden:

```
const [numbers, setNumbers] = useState([0, 1, 2]);

function handleRemoveNumber(numberToRemove) {
  setNumbers(numbers.filter((number) => number !== numberToRemove));
```

```
}

// Irgendwo anders:
handleRemoveNumber(1);
```

Ersetzen eines Array-Elements

Um ein Element in einem Array zu ersetzen, kannst du die `map`-Methode verwenden:

```
const [numbers, setNumbers] = useState([0, 1, 2]);

function handleReplaceNumber(oldNumber, newNumber) {
  setNumbers(
    numbers.map((number) => {
      if (number === oldNumber) return newNumber;
      return number;
    })
  );
}

// Irgendwo anders:
handleReplaceNumber(1, 1337);
```

Sortieren eines Arrays

Um ein Array zu sortieren, kannst du die Spread-Syntax verwenden, um zuerst eine Kopie des Arrays zu erstellen, und dann die Kopie sortieren:

```
const [numbers, setNumbers] = useState([12, 2, 42, 4]);

function handleSortNumbers() {
  setNumbers([...numbers].sort());
}
```

> Erfahre mehr über **Aktualisierung von Arrays ohne Mutation** in den React Docs.

Aktualisierung von Arrays von Objekten im State

Meistens wirst du in deinem State auf Arrays von Objekten stoßen.

Hinzufügen eines neuen Objekts

Du kannst ein neues Objekt zum State-Array hinzufügen, indem du die Spread-Syntax verwendest:

```
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
```

```
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

function handleAddTree(tree) {
  setTrees([...trees, tree]);
}

// Irgendwo anders:
handleAddTree({ id: 3, name: "Spruce", height: 13 });
```

Entfernen eines Objekts

Um ein Objekt zu entfernen, kannst du das Array nach einem eindeutigen Bezeichner `filtern`. In den meisten Fällen ist dieser Bezeichner im Geltungsbereich, da das betreffende Objekt über `map` gerendert wird.

```
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

function handleRemoveTree(idToRemove) {
  setTrees(trees.filter((tree) => tree.id !== idToRemove));
}

// Irgendwo anders:
handleRemoveTree(0);
```

Ersetzen eines Objekts

Um ein Objekt zu ersetzen, kannst du `map` verwenden, um ein neues Array mit dem aktualisierten Objekt zu erstellen. Denke daran, zuerst eine Kopie des Objekts zu erstellen, da du sonst den State mutierst.

```
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

function handleSetNewHeightForTree(id, height) {
  setTrees(
    trees.map((tree) => {
      if (tree.id === id) return { ...tree, height };
      return tree;
    })
  );
}
```

```
// Irgendwo anders:
handleSetNewHeightForTree(0, 8);
```

Sortieren eines Arrays von Objekten

Um ein Array von Objekten zu sortieren, kannst du `sort` auf einer *Kopie des Arrays* mit einer benutzerdefinierten Vergleichsfunktion verwenden. Die Vergleichsfunktion gibt eine Zahl zurück, die zur Bestimmung der Reihenfolge der Elemente verwendet wird.

```
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

function handleSortTreesByHeight() {
  setTrees([...trees].sort((a, b) => a.height - b.height));
}
```

> 📙 Erfahre mehr über **Aktualisierung von Objekten innerhalb von Arrays** in den React Docs.

Auswahl der State-Struktur

Es gibt einige häufige Stolperfallen bei der Auswahl deiner State-Struktur.

Gruppiere verwandten State

Wenn du State hast, der zusammengehört (und zusammen aktualisiert wird), gruppiere ihn in einem einzigen Objekt. Das macht es einfacher, den State zu aktualisieren.

```
// ❌ MEH
const [userName, setUserName] = useState("Alex");
const [userAge, setUserAge] = useState(28);

// ✅ BESSER
const [user, setUser] = useState({ name: "Alex", age: 28 });
```

Vermeide redundanten State

Wenn du einen Wert hast, der von einem State abgeleitet ist, solltest du ihn nicht im State speichern. Verwende stattdessen eine normale Variable.

Das Problem mit redundantem State ist, dass er nicht mehr synchron mit der Quelle der Wahrheit ist, wenn du vergisst, ihn korrekt zu aktualisieren.

```
// ❌ SCHLECHT
const [user, setUser] = useState({ name: "Alex", age: 28 });
const [isAdult, setIsAdult] = useState(user.age >= 18);

// ✅ GUT
const [user, setUser] = useState({ name: "Alex", age: 28 });
const isAdult = user.age >= 18;
```

Vermeide Duplikationen im State

Vermeide es, denselben Wert an mehreren Stellen im State zu speichern. Dies kann zu Bugs führen und erschwert das Aktualisieren des States.

```
// ❌ SCHLECHT
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

const [selectedTree, setSelectedTree] = useState(
  trees.find((tree) => tree.id === 0)
);

// Irgendwo anders:
setSelectedTree(trees.find((tree) => tree.id === 2));

// ✅ GUT
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

const [selectedTreeId, setSelectedTreeId] = useState(0);

const selectedTree = trees.find((tree) => tree.id === selectedTreeId);

// Irgendwo anders:
setSelectedTreeId(2);
```

Vermeide doppelte Listen im State

Wenn du eine Liste von Elementen im State hast, solltest du vermeiden, eine abgeleitete Version davon in einer anderen State-Variable zu speichern. Dies ist ein häufiger Fehler, wenn du eine gefilterte Version der Liste anzeigen möchtest.

```
// ❌ SCHLECHT
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

const [filteredTrees, setFilteredTrees] = useState(
  trees.filter((tree) => tree.height > 7)
);

// Irgendwo anders:
setFilteredTrees(trees.filter((tree) => tree.height > 9));

// ✅ GUT
const [trees, setTrees] = useState([
  { id: 0, name: "Oak", height: 7.5 },
  { id: 1, name: "Beech", height: 6 },
  { id: 2, name: "Pine", height: 10 },
]);

const [minHeight, setMinHeight] = useState(7);

const filteredTrees = trees.filter((tree) => tree.height > minHeight);

// Irgendwo anders:
setMinHeight(9);
```

> 📙 Erfahre mehr über **Auswahl der State-Struktur** in den React Docs. Die Dokumentation enthält viele weitere Beispiele und Erklärungen.

Resources

- Updating Objects in State (React Docs)
- Updating Arrays in State (React Docs)