# React Workshop - Handout

Philip Neumann, Max Tamási, Nico Scherer, Jens Zangerle

## Was ist React?

React ist eine JavaScript-Bibliothek zur GUI-Programmierung, die basierend auf Komponenten und Virtual DOM für effiziente Aktualisierungen sorgt

komponent:

- wiederverwendbarer UI-Baustein (z.B. Button, Menü, Kommentar)
- enthalten eigene Daten und Logik
- Datenänderung betrifft nur den spezifischen komponent

Virtual DOM (Document Object Model):

- Interne baumartige darstellung einer website
- Ist die Schnittstelle damit JS HTML Inhalte ändern kann
- VDOM ist eine kopie des DOM die schnelleren zugriff ermöglicht
- React vergleicht VDOM und DOM und übergibt veränderungen

## Warum React?

React wird eingesetzt, um Webanwendungen modular, dynamisch und performant zu gestalten. Die Wiederverwendbarkeit der Komponente und Effiziens des VDOMs sind zentrale Gründe

## Vorteile

- Modular und wiederverwendbar
- Performant
- Klare trennung von UI und Logk

## Nachteile

- deckt nur die UI ab
- JSX Lernkurve

## Vergleich HTML und JSX

HTML

```html
<div class="box">
  <label for="name">Name:</label>
  <input type="text" />
</div>
```

```jsx
<div className="box">
  <label htmlFor="name">Name:</label>
  <input type="text" />
</div>
```

## State (useState)

State ist das Gedächtnis einer React-Komponente. Normale Variablen werden nach einem Render vergessen, State bleibt erhalten.

Wichtig:

- State enthält veränderbare Daten
- Bei einer Änderung rendert React die Komponente neu

Beispiel:

```jsx
function Counter() {
  const [count, setCount] = useState(0);

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

setCount teilt React mit, dass sich der State geändert hat und die Komponente neu gerendert werden muss.

## Bedingtes Rendern

Beim bedingten Rendern passt sich die Benutzeroberfläche an bestimmte Bedingungen an.

Typische Varianten:

If (frühes return)

```jsx
if (isLoggedIn) {
  return <p>Willkommen</p>;
}
```

Ternary Operator (entweder / oder)

```
{isLoggedIn ? <p>Willkommen</p> : <p>Bitte einloggen</p>}
```

&& (anzeigen oder nichts)

```
{isLoggedIn && <p>Willkommen</p>}
```

## Controlled Inputs

Bei Controlled Inputs wird der Inhalt von Eingabefeldern vollständig durch React gesteuert.

- Der Wert kommt aus dem State
- Jeder Tastendruck aktualisiert den State
- Das Input besitzt keinen eigenen Zustand mehr

Beispiel:

```
function NameInput() {
  const [name, setName] = useState("");

  return (
    <>
      <input
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <p>Hallo {name}</p>
    </>
  );
}
```

## Props

Props werden genutzt, um Daten von einer Elternkomponente an eine Kindkomponente weiterzugeben.

- Datenfluss: Eltern → Kind
- Props sind read-only
- Komponenten werden wiederverwendbar

Beispiel:

Kindkomponente

```
function Greeting({ name }) {
  return <h2>Hallo {name}</h2>;
}
```

Elternkomponente

```
function App() {
  return <Greeting name="Max" />;
}
```

## Listen & Keys

Listen werden in React mit **`.map()`** gerendert und benötigen eindeutige, stabile Keys, damit React Änderungen korrekt erkennen kann.
Zufällige Werte wie **`Math.random()`** sind ungeeignet, und der Index sollte nur verwendet werden, wenn sich die Liste niemals verändert.
Keys dienen ausschließlich React intern und stehen nicht als Prop zur Verfügung.

## Events & Handler

Events werden in React in camelCase geschrieben, zum Beispiel **`onClick`** oder **`onSubmit`**.
Event-Handler werden als Funktionsreferenz übergeben und nicht direkt ausgeführt, bei Parameterübergabe nutzt man eine Wrapper-Funktion.
Auf das Event-Objekt greift man über **`e.target.value`** zu, wobei React mit Synthetic Events und zentraler Event-Verarbeitung arbeitet.

| | |
|---|---|
| Ohne Parameter: | onClick={handleClick} |
| Mit Parameter: | onClick={() => handleDelete(id)} |
| Falsch: | onClick={handleClick()} |

## useEffect & Side Effects

**`useEffect`** führt Code nach dem Rendern aus und wird für Side Effects wie das laden von Daten, Timer, Event-Listener oder Zugriffe auf den Local Storage verwendet.
Über das Abhängigkeits-Array wird gesteuert, wann der Effekt erneut ausgeführt wird.

## Variante 1: Nach jedem Render (sollte vermieden werden)

```
useEffect(() => {
console.log("läuft immer");
});
```

## Variante 2: Nur beim ersten Render

```
useEffect(() => {
console.log("nur beim ersten Mal");
}, []);
```

## Variante 3: Wenn bestimmte Werte sich ändern

```
useEffect(() => {
console.log("count hat sich geändert:", count);
}, [count]);
```

**Cleanup**
Alles, was geöffnet wird, muss im Cleanup wieder geschlossen werden, um Memory Leaks oder doppelte Listener zu verhindern.

**Interval**:

```
useEffect(() => {
  const id = setInterval(() => {
    console.log("tick");
  }, 1000);

  return () => {
    clearInterval(id);   // Cleanup
  };
}, []);
```

## Event Listener:

```
useEffect(() => {
  const onResize = () => {
    console.log(window.innerWidth);
  };

  window.addEventListener("resize", onResize);

  return () => {
    window.removeEventListener("resize",
onResize);   // Cleanup
  };
}, []);
```

## Derived State

Berechnete Werte sollten idealerweise nicht doppelt im Code gespeichert werden, sondern nur abhängig vom Wert, von dem abgeleitet wird, berechnet werden. Best Practice dafür ist es, an der Stelle, an der diese benötigt werden, diese zu berechnen.

```
// state
const [namen, setNamen] = useState(["Anna", "Ben"]);
//derived state
const [anzahl, setAnzahl] = namen.length;
//kein derived state
const [anzahl, setAnzahl] = useState(2);
```

## Commands

- **npm create-react-app** - Erstellt React Vorlage
- **npm start** - Startet Projekt
- **npm run dev** - startet Projekt im Entwicklermodus
- **npm install** - Lädt Abhängigkeiten runter

## Projektstruktur

- **src/** - enthält gesamten Quellcode der Anwendung
- **App.jsx** - Hauptkomponente der App, hier werden andere Komponenten zusammengesetzt
- **main.jsx** - Der Einstiegspunkt – hier wird die App gestartet.
- **components/** - Kann angelegt werden, um wiederverwendbare Bausteine als Components anzulegen
- Best Practise: Jede logische Funktion eine Datei also ein Komponente