

State of the Art

Zustandsmanagement

in React Anwendungen

Slides: <https://nils.buzz/oose2020-react-state>

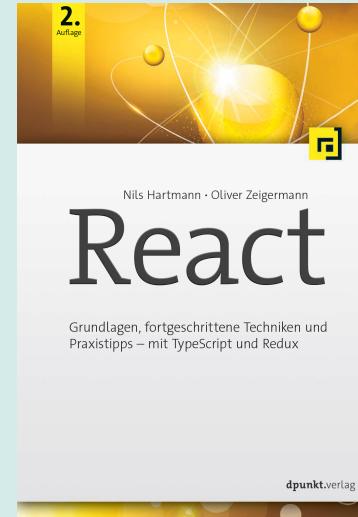
NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings & Workshops



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

NILS HARTMANN

nils@nilshartmann.net

Nächstes öffentliches React-Seminar (online):

25./26. Juni

<https://www.oose.de/seminar/web-apps/>

The screenshot shows a web browser window with the title "React Training Blog". The URL is "localhost:3000". The page displays a list of blog posts:

- 18.04.2020** **Understanding State** Your Post!
Read more 19 Likes
- 17.04.2020** **Increasing React developer experience**
Read more 38 Likes (including me!)
- 02.04.2020** **Using Redux with care** Your Post!
Read more 21 Likes
- 07.01.2020** **Do's and don'ts with React**
Read more 26 Likes

On the right side, there is a sidebar titled "View History" with the following entries:

- 07.01.2020** Do's and don'ts with React
26 Likes
- 18.04.2020** Understanding State
19 Likes
- 17.04.2020** Increasing React developer experience
38 Likes

<https://nils.buzz/react-state-example>

EIN BEISPIEL...

ZUSTAND IN REACT-ANWENDUNGEN

Lokaler vs globaler State

- Lokaler State steht in der Regel **einer einzelnen** Komponente zur Verfügung
- Globaler State steht der gesamten Anwendung zur Verfügung
- Es gibt natürlich Überschneidungen...

The screenshot shows a web browser window with the title "React Training Blog". The URL is "localhost:3000". The top right corner displays "Welcome, Susi Mueller" and a "Logout" link. On the left, there's a button "Add Post". In the center, there's a search bar with the placeholder "Order by Date | Likes (asc / desc)". Below the search bar is a list of blog posts:

- 18.04.2020 **Understanding State** Your Post!
Read more [19 Likes]
- 17.04.2020 **Increasing React developer experience** Your Post!
Read more [38 Likes] (Unread)
- 02.04.2020 **Using Redux with care** Your Post!
Read more [21 Likes]
- 07.01.2020 **Do's and don'ts with React** Your Post!
Read more [26 Likes]

To the right, a sidebar titled "View History" lists previous posts:

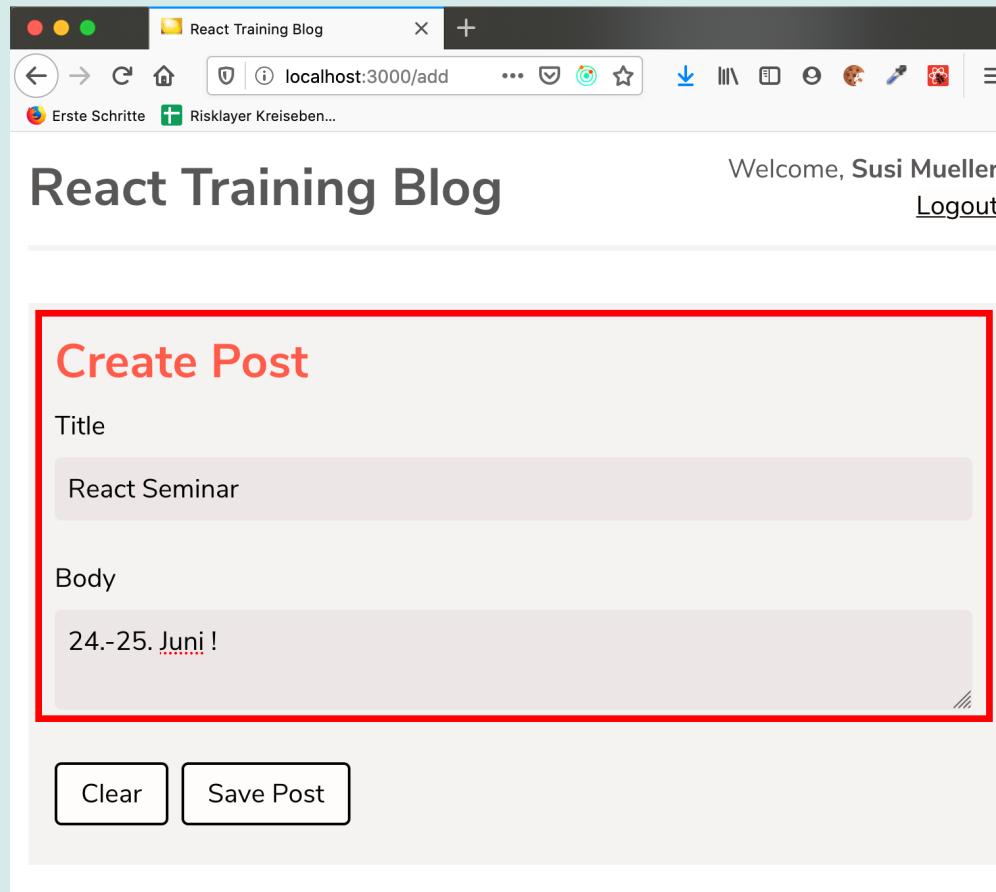
- 07.01.2020 Do's and don'ts with React 26 Likes
- 18.04.2020 Understanding State 19 Likes
- 17.04.2020 Increasing React developer experience 38 Likes

A large orange watermark with a thinking emoji (🤔) and the text "Wo haben wir hier überall State?" is overlaid diagonally across the post list.

EIN BEISPIEL...

BEISPIELE FÜR STATE

Global oder lokal? Formular für neuen Blog Post



BEISPIELE FÜR STATE

Global oder lokal? Angemeldeter Benutzer

The screenshot shows a web browser window for the "React Training Blog" at `localhost:3000`. The page displays a list of blog posts and a user profile section.

User Authentication: The top right corner shows a welcome message "Welcome, Susi Mueller" and a "Logout" link, both enclosed in a red box. The browser's address bar also shows the URL `localhost:3000`.

Blog Posts:

- Post 1:** Date: 18.04.2020, Title: [Understanding State](#), "Your Post!" button, "Read more" link (19 Likes).
- Post 2:** Date: 17.04.2020, Title: [Increasing React developer experience](#), "Read more" link (38 Likes including me!).
- Post 3:** Date: 02.04.2020 (partially visible).

Ordering: A "Order by" dropdown menu is present, with options "Date" (underlined), "Likes (asc)", and "Likes (desc)".

View History: A sidebar on the right lists recent viewed posts:

- 07.01.2020: Do's and don'ts with React, 26 Likes
- 18.04.2020: Understanding State, 19 Likes
- 17.04.2020: Increasing React developer experience, 38 Likes

BEISPIELE FÜR STATE

Global oder lokal? Ein einzelnes Blog-Post

The screenshot shows a web browser window for 'React Training Blog' at localhost:3000/post/P10. The page title is 'React Training Blog'. The top right corner shows a welcome message 'Welcome, Susi Mueller' and a 'Logout' link. A navigation bar includes a 'Home' button. The main content area contains a red-bordered box containing the following text:

Understanding State

Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet. »Wie ein Hund!

Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.

Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte.

»Es ist ein eigentümlicher Apparat«, sagte der Offizier zu dem Forschungsreisenden und überblickte mit einem gewissermaßen bewundernden Blick den ihm doch wohlbekannten Apparat.

To the right of the main content is a sidebar titled 'View History' containing the following list:

- 18.04.2020
Understanding State
19 Likes
- 07.01.2020
Do's and don'ts with React
26 Likes
- 17.04.2020
Increasing React developer experience
38 Likes

BEISPIELE FÜR STATE

Global oder lokal? Liste mit Blog-Posts ("Summaries")

The screenshot shows a web browser window for the "React Training Blog" at `localhost:3000`. The page displays a list of blog posts with a red border around the first four items. Each post card includes the date, title, a "Read more" link, a like count, and a "Your Post!" button.

Date	Title	Actions
18.04.2020	Understanding State	Read more [19 Likes] Your Post!
17.04.2020	Increasing React developer experience	Read more [38 Likes (including me!)]
02.04.2020	Using Redux with care	Read more [21 Likes] Your Post!
07.01.2020	Do's and don'ts with React	Read more [26 Likes]

On the right side, there is a sidebar titled "View History" containing a list of previous posts:

- 07.01.2020
Do's and don'ts with React
26 Likes
- 18.04.2020
Understanding State
19 Likes
- 17.04.2020
Increasing React developer experience
38 Likes

BEISPIELE FÜR STATE

Global oder lokal? Was ist mit den Likes?

The screenshot shows a web browser window for the "React Training Blog" at localhost:3000. The page displays four blog posts with their publication dates, titles, and like counts. The like counts are highlighted with red boxes. A sidebar on the right shows a history of viewed posts.

Date	Title	Likes
18.04.2020	Understanding State	19 Likes
17.04.2020	Increasing React developer experience	38 Likes (including me!)
02.04.2020	Using Redux with care	21 Likes
07.01.2020	Do's and don'ts with React	26 Likes

Welcome, Susi Mueller
[Logout](#)

[Add Post](#)

Order by [Date](#) | [Likes \(asc\)](#) / [desc](#)

View History

- 07.01.2020
Do's and don'ts with React
26 Likes
- 18.04.2020
Understanding State
19 Likes
- 17.04.2020
Increasing React developer experience
38 Likes

BEISPIELE FÜR STATE

Global oder lokal? State für den Filter der Post-Liste

The screenshot shows a web browser window for a blog application at `localhost:3000`. The title bar says "React Training Blog". The page content includes:

- A header with "Welcome, Susi Mueller" and a "Logout" link.
- A "Add Post" button.
- A "View History" sidebar listing previous visits:

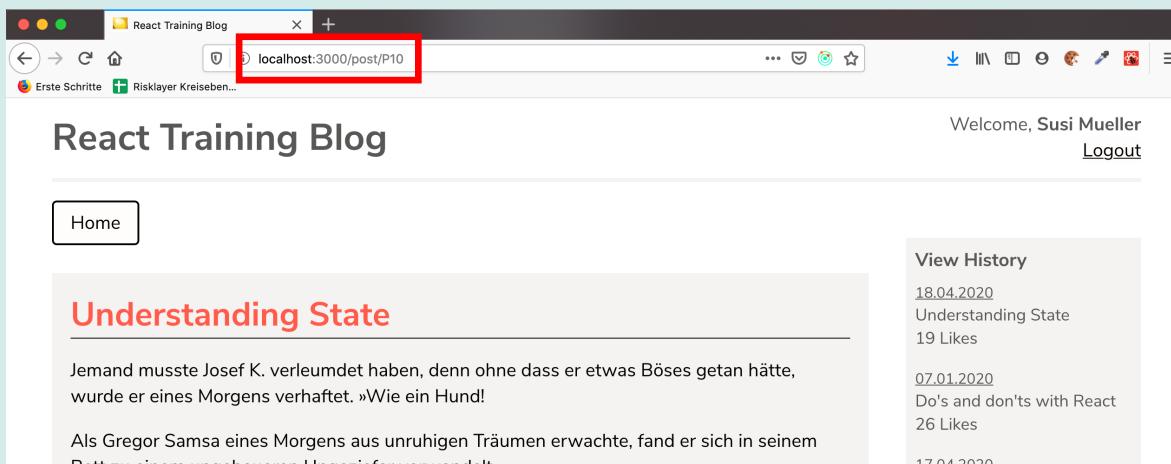
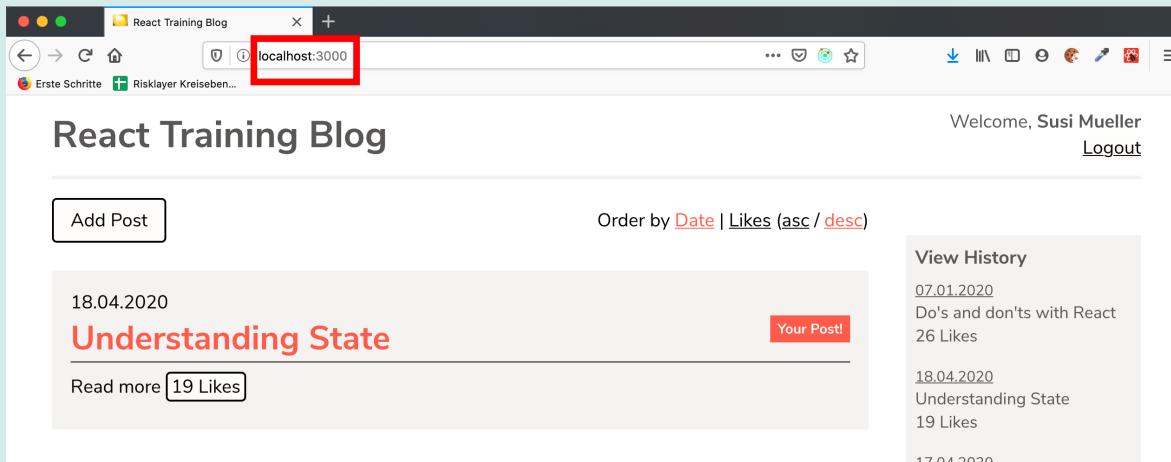
 - 07.01.2020: Do's and don'ts with React (26 Likes)
 - 18.04.2020: Understanding State (19 Likes)
 - 17.04.2020: Increasing React developer experience (38 Likes)

- A main area displaying two posts:
 - Understanding State** (Date: 18.04.2020). It has a "Your Post!" button and a "Read more [19 Likes]" link.
 - Increasing React developer experience** (Date: 17.04.2020). It has a "Read more [38 Likes (including me!)]" link.
- An "Order by Date | Likes (asc / desc)" button, which is highlighted with a red border.

BEISPIELE FÜR STATE

Was ist eigentlich damit? Die URL...

- Welcher Post wird gerade angezeigt?



EIGENSCHAFTEN VON ZUSTAND

Global oder lokal

- Global: Theme, angemeldeter Benutzer, ...
- "Halbglobal": Zustand, der über Komponenten-Wechsel erhalten bleiben soll (z.B. Sortierung)
- Lokal: nur in einer Komponente relevant (z.B. Formular)

EIGENSCHAFTEN VON ZUSTAND

Global oder lokal

- Global: Theme, angemeldeter Benutzer, ...
- "Halbglobal": Zustand, der über Komponenten-Wechsel erhalten bleiben soll (z.B. Sortierung)
- Lokal: nur in einer Komponente relevant (z.B. Formular)

Reiner UI-State vs. "Cache"

- Filter und recently viewed werden nur auf dem Client gehalten, sind nur für die UI relevant
- Blog-Posts und Likes werden auf dem Server gespeichert, müssen als synchronisiert werden

EIGENSCHAFTEN VON ZUSTAND

Global oder lokal

- Global: Theme, angemeldeter Benutzer, ...
- "Halbglobal": Zustand, der über Komponenten-Wechsel erhalten bleiben soll (z.B. Sortierung)
- Lokal: nur in einer Komponente relevant (z.B. Formular)

Reiner UI-State vs. "Cache"

- Filter und recently viewed werden nur auf dem Client gehalten, sind nur für die UI relevant
- Blog-Posts und Likes werden auf dem Server gespeichert, müssen als synchronisiert werden

Stabiler vs. veränderlicher Zustand

- Theme und Benutzer ändern sich nur selten
- Zustand im Eingabefeld ändert sich häufig und schnell
- Was ist mit Daten vom Server? (Beispiel: Klick auf "Like")
- Konsequenz für Performance!

EIGENSCHAFTEN VON ZUSTAND

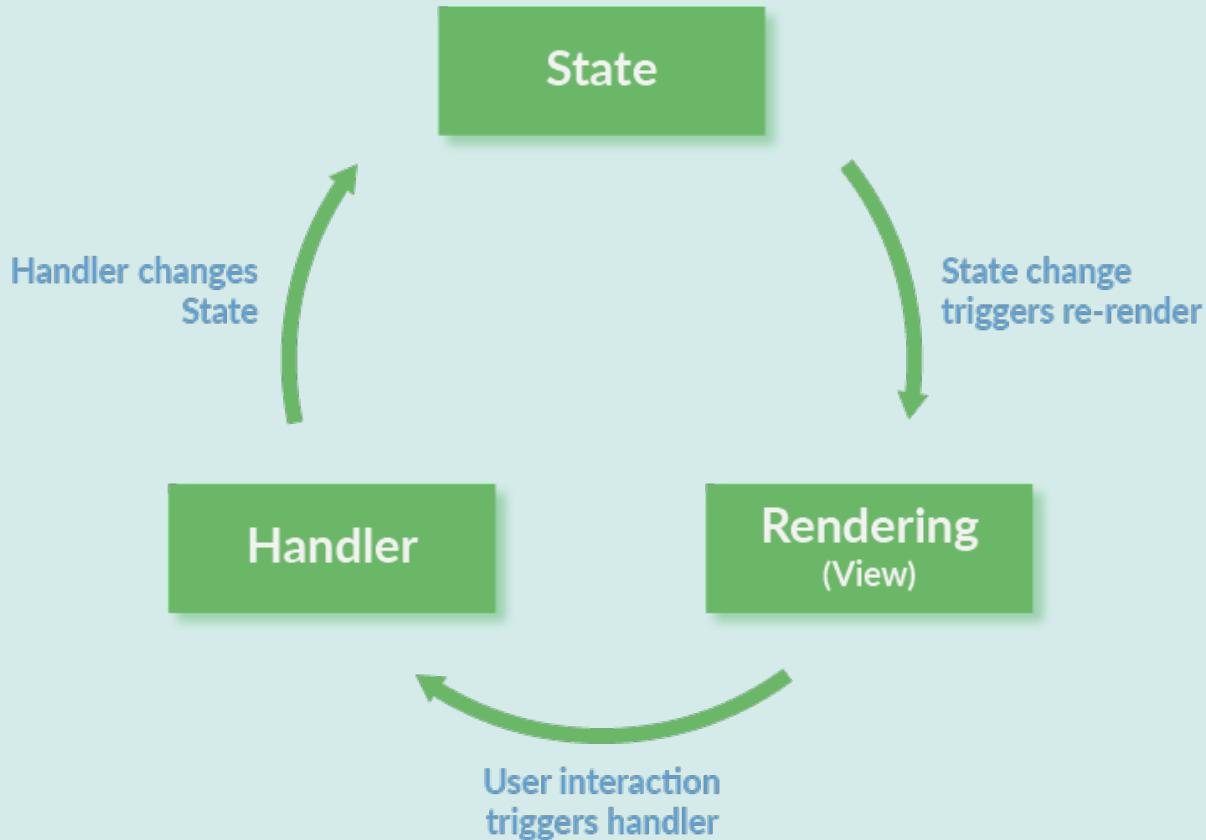
Fragestellungen

- Wo wird der State aufgehängt? Global oder lokal?
- Wie kommen die benötigten Komponenten an den State?
- Wie können Komponenten mit dem State interagieren (verändern)?

EIGENSCHAFTEN VON ZUSTAND

React State

- Grundsätzlicher Datenfluss in React-Anwendungen in eine Richtung
 - Das ändert sich auch mit den gezeigten Ansätzen nicht



Umsetzung

Vorbemerkung: TypeScript

- Funktioniert mit allen gezeigten Ansätzen reibungslos
- Empfehlung: Verwenden
 - (Steht sogar im Redux "Style Guide")

LOKALER STATE

Der Klassiker: lokaler State mit React.useState

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  }  
}
```

Der Klassiker: lokaler State mit React.useState

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e => setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e => setPassword(e.target.value)} />  
  </>);  
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Custom Hooks, um State zu verwalten

- Hooks sind reguläre JavaScript-Funktionen
- Signatur und Rückgabewert können frei gewählt werden
 - (anders als bei Funktionskomponenten)
- Hooks können Hooks verwenden (setState z.B.!)
- Wenn ein Hook State enthält und verändert, wird die verwendende Komponente neu gerendert

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hello World

```
// Hook
function useCounter(initial) {
  const [ value, setValue ] = React.useState(initial);

  return {
    count: value,
    increaseCounter() { setValue(value + 1) }
  }
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hello World

```
// Hook
function useCounter(initial) {
  const [ value, setValue ] = React.useState(initial);

  return {
    count: value,
    increaseCounter() { setValue(value + 1) }
  }
}

// Komponente
function CounterButton() {
  const { count, increaseCounter } = useCounter(10);
  return <button onClick={increaseCounter}>{count}</button>
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

- Verwaltet Daten und Request-Zustand

```
function useApi(url) {  
  const [ loading, setLoading ] = React.useState(false);  
  const [ data, setData ] = React.useState(null);  
  
  return { loading, data };  
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

```
function useApi(url) {  
  const [ loading, setLoading ] = React.useState(false);  
  const [ data, setData ] = React.useState(null);  
  
  React.useEffect( () => {  
    setLoading(true);  
  
    fetch(url) // vereinfacht  
      .then(res => {  
        setLoading(false);  
        setData(res.json());  
      })  
  }, [url]);  
  
  return { loading, data };  
}
```

HOOKS ALS ALTERNATIVE

Beispiel: Hook zum Laden von Daten

- (Wieder-)Verwendung

```
function BlogListPage() {  
  const { loading, data } = useApi("http://api/posts");  
  
  if (loading) {  
    return <LoadingIndicator />  
  }  
  
  return <BlogList posts={data} />  
}
```

HOOKS ALS ALTERNATIVE

Beispiel: Hook zum Laden von Daten

- (Wieder-)Verwendung

```
function BlogListPage() {  
  const { loading, data } = useApi("http://api/posts");  
  
  if (loading) {  
    return <LoadingIndicator />  
  }  
  
  return <BlogList posts={data} />  
}  
  
function BlogPage({postId}) {  
  const { loading, data } = useApi(http://api/posts/ + postId);  
  
  if (loading) {  
    return <LoadingIndicator />  
  }  
  
  return <Blog post={data} />  
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

```
function useApi(url) {  
  const [ loading, setLoading ] = React.useState(false);  
  const [ data, setData ] = React.useState(null);  
  
  React.useEffect( () => {  
    setLoading(true); ← Fehleranfällig!  
    fetch(url) // vereinfacht  
      .then(res => {  
        setLoading(false); ← Kein setData, ... ist das gewollt?  
        setData(res.json());  
      })  
  }, [url]);  
  
  return { loading, data };  
}
```

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

Noch ein Zustand 🤯

```
function useApi(url) {  
  const [ loading, setLoading ] = React.useState(false);  
  const [ error, setError ] = React.useState(null);  
  const [ data, setData ] = React.useState(null);  
  
  React.useEffect( () => {  
    setLoading(true);  
    setError(null); ← Kein setData, ... ist das gewollt?  
    fetch(url) // vereinfacht  
      .then(res => {  
        setLoading(false);  
        setData(res.json()); ← Was passiert, wenn wir error nicht  
      }).catch(e => setError(e)) ← zurücksetzen?  
  }, [url]); ← ...oder hier?  
...  
}
```

Noch komplexer: Fehlerzustand!

Kein setData, ... ist das gewollt?
Was passiert, wenn wir error nicht zurücksetzen?

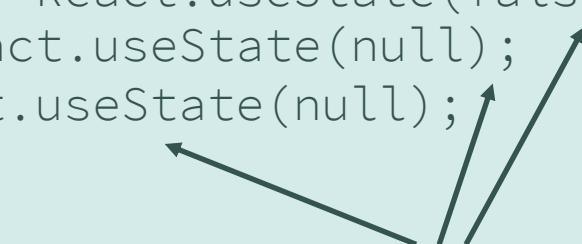
Was passiert, wenn wir vergessen, loading **oder error** zurückzusetzen?

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

Noch ein Zustand 🤯

```
function useApi(url) {  
  const [ loading, setLoading ] = React.useState(false);  
  const [ error, setError ] = React.useState(null);  
  const [ data, setData ] = React.useState(null);  
  
  React.useEffect( () => {  
    setLoading(true);  
    setError(null);  
  
    fetch(url) // vereinfacht  
      .then(res => {  
        setLoading(false);  
        setData(res.json());  
      }).catch(e => setError(e))  
  }, [url]);  
  ...  
}
```



👉 Diese "Teilzustände" sind nicht unabhängig!

CUSTOM HOOKS ZUR VERWALTUNG VON STATE

Beispiel: Hook zum Laden von Daten

Objekte bei "komplexem" Zustand

```
function useApi(url) {  
  const [ apiState, setApiState ] =  
    React.useState({ loading: false, data: null, error: null });  
  
  React.useEffect( () => {  
    setApiState({loading: true});  
  
    fetch(url) // vereinfacht  
      .then(res => setApiState({data: res}))  
      .catch(err => setApiState({error: err}));  
  }, [url]);  
  
  return apiState;  
}
```

Ein "logischer" Zustand

Einfacher State oder komplexer State?

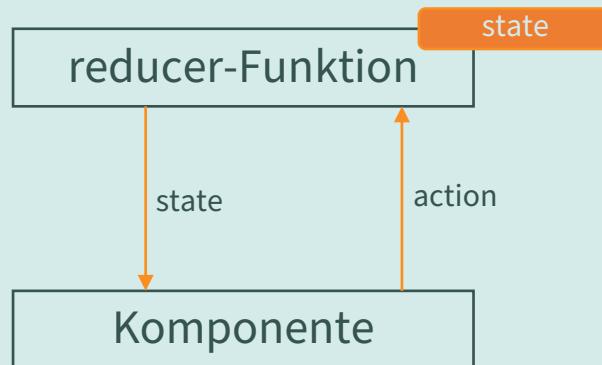
Empfehlung:

- **Einfachen State** für unabhängige Werte verwenden (z.B. Felder im Eingabefeld)
- **Komplexen State** für Werte, die in der Regel gemeinsam geändert werden (*loading* oder *data*)

USERREDUCER HOOK

useReducer

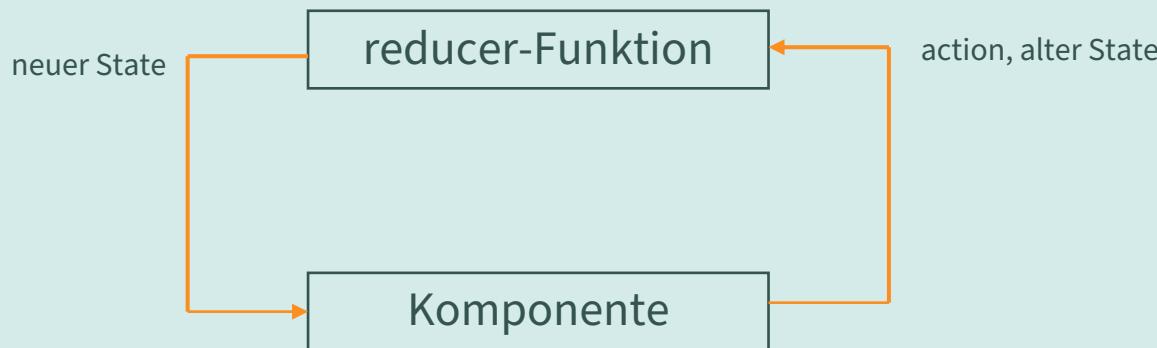
- Logik und Zustand wandern aus Komponente in eine neue Funktion
- Diese Funktion ist React-unabhängig



USERREDUCER HOOK

useReducer

- Logik und Zustand wandern aus Komponente in eine neue Funktion
- Diese Funktion ist React-unabhängig



USERREDUCER HOOK

useReducer: Actions drücken aus, was in der Anwendung passiert ist
Actions sind einfache JavaScript-Objekte (besser: Events? 😊)

```
const action = {  
  type: "LOAD_FINISHED", ----- Type  
  data: "... "      ----- Payload  
}
```

```
const action = {  
  type: "LOAD_FAILED",  
  error: "..."  
}
```

```
const action = {  
  type: "FETCH_START"  
}
```

USERREDUCER HOOK

useReducer: Reducer-Funktion enthält die fachliche Logik

Allgemeine Signatur: (state, action) => newState

```
function apiReducer(oldState, action) {  
  switch (action.type) {  
    case "FETCH_START":  
  
  }  
}
```

USERREDUCER HOOK

useReducer: Reducer-Funktion enthält die fachliche Logik

Allgemeine Signatur: (state, action) => newState

```
function apiReducer(oldState, action) {  
  switch (action.type) {  
    case "FETCH_START":  
      return { ...oldState, loading: true };  
  }  
}
```

USERREDUCER HOOK

useReducer: Reducer-Funktion enthält die fachliche Logik

Allgemeine Signatur: (state, action) => newState

```
function apiReducer(oldState, action) {  
  switch (action.type) {  
    case "FETCH_START":  
      return { ...oldState, loading: true, error: null };  
    case "LOAD_FAILED":  
      return { loading: false, error: action.error };  
  
    case "LOAD_FINISHED":  
      return { data: action.data };  
  
    default:  
      return throw new Error("Invalid action!");  
  }  
}
```

USERREDUCER HOOK

useReducer: Logik wird aus Komponente in reducer-Funktion geschoben
Schritt 2: Verwenden

```
function apiReducer() { ... }

function useApi(url) {
  const [state, dispatch] = React.useReducer(apiReducer);

  React.useEffect( () => {
    dispatch({ type: "FETCH_START" });

    fetch(...)
      .then(res => dispatch({type: "LOAD_FINISHED", data: res }))
  }, []);

  return state;
}
```

USERREDUCER HOOK

useReducer: Konsequenzen

- Reducer Standard-JavaScript-Funktion, d.h. gut test- und wiederverwendbar
- Logik zur Behandlung des Zustandes an einer zentralen Stelle
- dispatch von Actions Code-intensiv

USERREDUCER HOOK

useReducer: Reducer müssen Kopien zurückgeben (immutable)

Kann sehr Code-lastig sein:

```
function blogListOptionsReducer(state, action) {  
  switch (action.type) {  
    case "SET_BLOGLIST_FILTER_BY_LIKES": {  
      return { ...state, likes: action.likes };  
    }  
    case "SET_BLOGLIST_SORT": {  
      return {  
        ...state,  
        sortBy: action.sortBy,  
        order: action.direction,  
      };  
    }  
    default:  
      return state;  
  }  
}
```

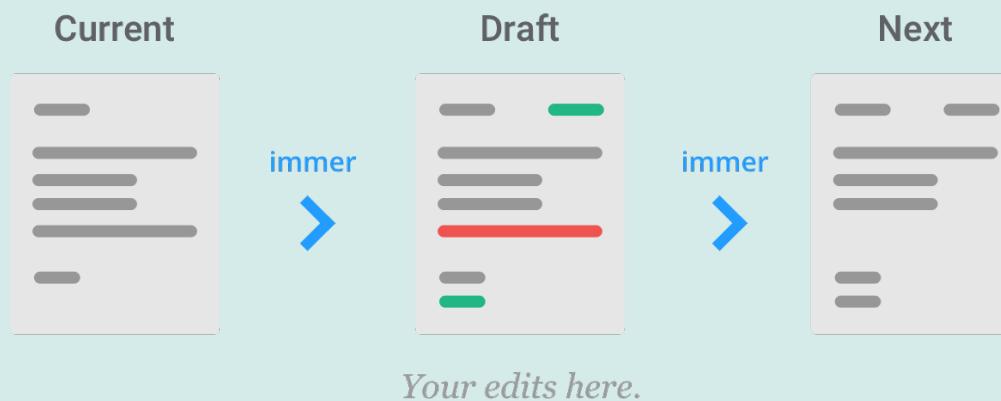
Das ist nur ein
einfaches Beispiel 😱

USERREDUCER HOOK

immer erlaubt mutable Code zu schreiben, der "normal" aussieht

<https://immerjs.github.io/immer/docs/introduction>

Immer (German for: always) is a tiny package that allows you to work with immutable state in a more convenient way. It is based on the copy-on-write mechanism.



USERREDUCER HOOK

Beispiel: reducer-Funktion mit immer

```
import produce from "immer";  
  
function blogListOptionsReducer(oldState, action) {  
  return produce(oldState, state => {  
    switch (action.type) {  
      case "SET_BLOGLIST_FILTER_BY_LIKES": {  
        state.likes = action.likes;  
        return;  
      }  
      case "SET_BLOGLIST_SORT": {  
        state.sortBy = action.sortBy;  
        state.order = action.direction;  
      }  
    }  
  });  
}
```

state ist ein Proxy,
oldState bleibt
unverändert!

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Noch einmal: Die Login-Form

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e => setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e => setPassword(e.target.value)} />  
  
  </>);  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Zustand ist außerhalb der Form erst nach Button-Klick relevant

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e => setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e => setPassword(e.target.value)} />  
  
    <button onClick={  
      () => props.doLogin(username, password)}>Login</button>  
  </>);  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Zustand ist außerhalb der Form erst nach Button-Klick relevant

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e =>  
            setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e =>  
            setPassword(e.target.value)} />  
  
    <button onClick={() => props.doLogin(username, password)}>  
      Login  
    </button>  
  );  
}
```

Sehr einfaches Beispiel:

Zum Beispiel:

- kein Tracking, ob Feld "besucht" wurde
- keine Feld-Validierungen
- keine Gesamt-Validierung

👉 Wiederverwendbare Form-Komponente könnte schlau sein!

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Idee: Wiederverwendbare Form-Komponente

```
function LoginForm(props) {
```

```
    return (<Form> ←
```

Hält Zustand,
Validierungen etc

```
        </Form>);  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Idee: Wiederverwendbare Form-Komponente

```
function LoginForm(props) {  
  
  return (<Form>  
    <input value={???}  
      onChange={e => ???} />  
  
    <input value={???}  
      onChange={e => ???} />  
  
    <button onClick={  
      () => props.doLogin(???)}>Login</button>  
  </Form>);  
}
```

🤔 Wie kommen die Elemente an
ihre Werte und die Callbacks?

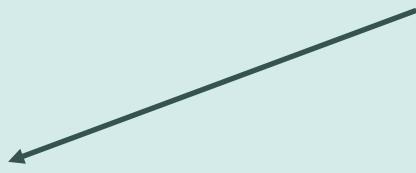
KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Ansatz 1: Render-Property

```
function LoginForm(props) {  
  
    return (<Form>{ formState => (  
  
        )}  
    </Form>);  
}
```

Das Children ist eine Funktion!



KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Ansatz 1: Render-Property

```
function LoginForm(props) {  
  return (<Form>{ formState => (  
    <input value={formstate.username.value}  
      onChange={formstate.username.onChange} />  
  
    <input value={formstate.password.value}  
      onChange={formstate.password.onChange} />  
  
    <button onClick={  
      () => props.doLogin(formstate.username.value,  
        formstate.password.value)}>Login</button>  
  )}  
</Form>);  
}
```

Form-Komponente übergibt z.B.
ein Objekt mit allen Formular-
Teilen

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Die Form-Komponente

```
function Form(props) {  
  const [formState, setFormState] = React.useState({....});  
}  
↑
```

Angenommener State:
für jedes Feld ein Eintrag im Objekt
zu jedem Eintrag gehört Wert, onChange, ...

```
{  
  username: { value: ..., onChange: ... },  
  password: { value: ..., onChange: ... }  
}
```

State über Komponentengrenzen

Die Form-Komponente

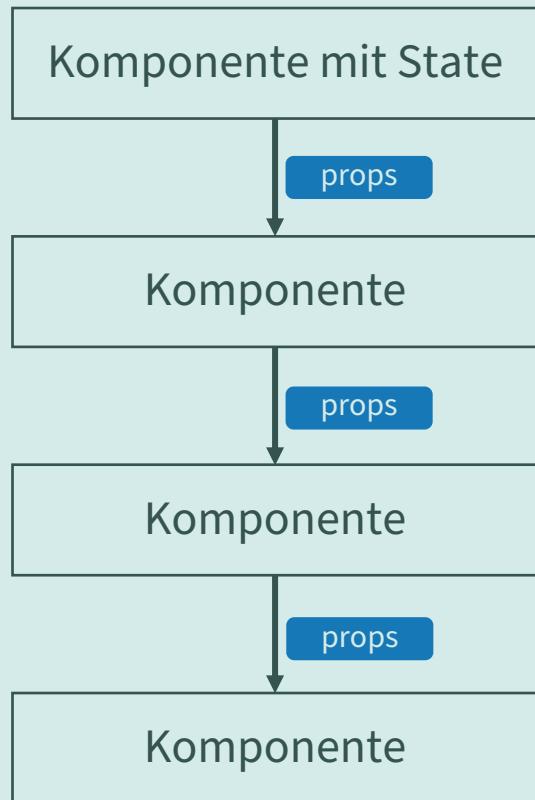
```
function Form(props) {  
  const [formState, setFormState] = React.useState({...});  
  
  return <form>  
    {props.children(formState)}  
  </form>;  
}
```

👉 children ist eine Funktion
und kann aufgerufen werden!



REACT CONTEXT

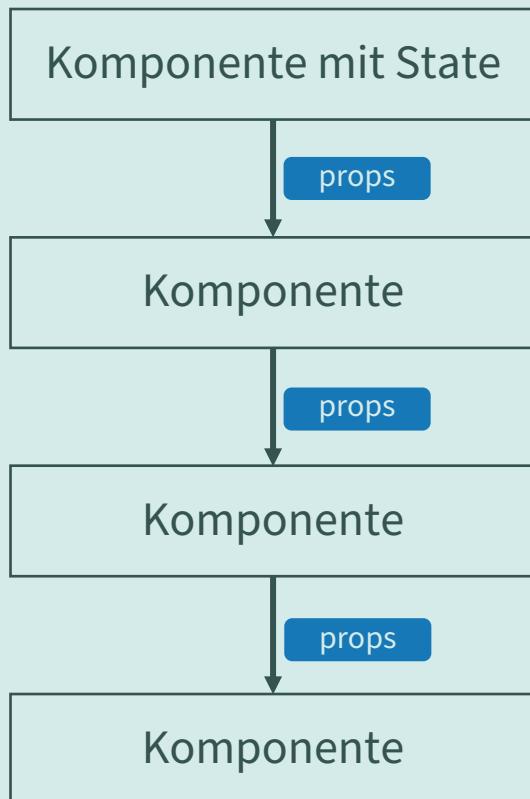
Nächster Ansatz: React Context



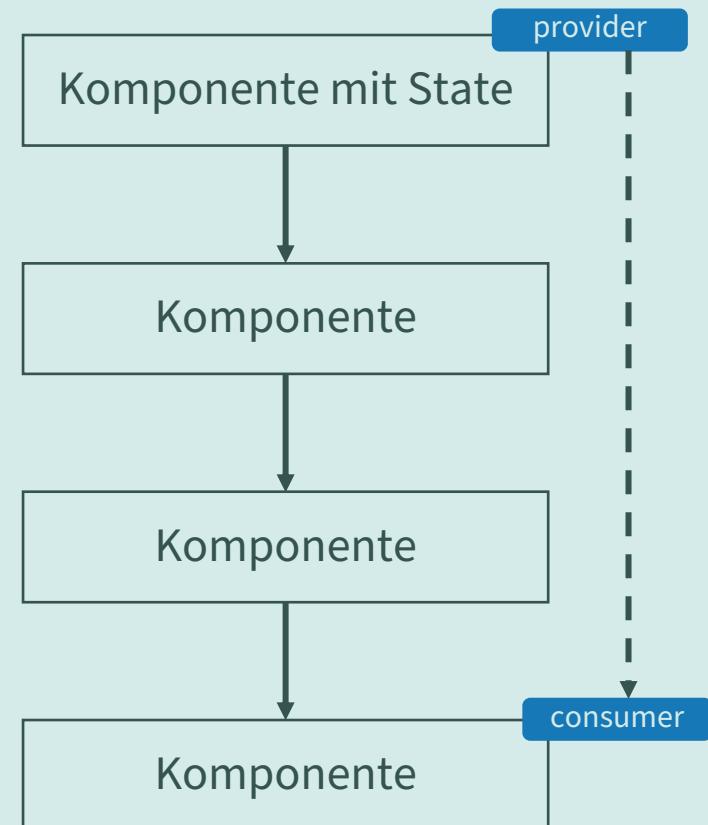
Klassisch: State wird per Properties durchgereicht

REACT CONTEXT

React Context: Stellt Werte innerhalb einer Komponentenhierarchie zur Verfügung



State wird per Properties durchgereicht



State wird mit Context bereitgestellt

REACT CONTEXT

React Context: Stellt Werte innerhalb einer Komponentenhierarchie zur Verfügung

Provider-Komponente

- Stellt Werte für darunterliegende Komponenten zur Verfügung
- Muss dazu ihre Kinder rendern

Provider-Komponente

- Kann die bereitgestellten Werte konsumieren

KOMMUNIKATION VON KOMPONENTEN

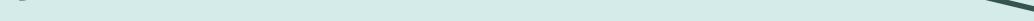
State über Komponentengrenzen

Beispiel Context – Provider-Komponente

```
const FormContext = React.createContext();

function Form(props) {
  const [formState, setFormState] = React.useState({ ... });

  return <FormContext.Provider value={{formState}}>
    {children}
  </form>;
}
```



React-Element als Children ("klassisch")

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Beispiel Context - Zugriff

```
function LoginForm(props) {  
  return <Form> ← Provider  
    <...>  
    <...> ← Consumer  
    <...>  
  </Form>;  
}
```

Alle Komponenten unterhalb
des Providers haben Zugriff auf
das bereitgestellte Objekt

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Beispiel: Ein Consumer

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
}  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Beispiel: Ein Consumer

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Beispiel: Ein 2. Consumer

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}  
  
function Button({children, onButtonClick}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <button  
            onClick={() => onButtonClick(formState)}>{children}</button>;  
}
```

State über Komponentengrenzen

Formular mit Context

```
function LoginForm(props) {  
  return (<Form>  
    <TextField name="username" />  
    <TextField name="password" />  
    <Button onClick={(state) =>  
      props.doLogin(state.username.value, state.password.value)}  
      Login  
    </Button>  
  </Form>);  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Formular mit Context

```
function LoginForm(props) {  
  return (<Form> ←  
    <TextField name="username" />  
    <TextField name="password" />  
    <Button onClick={() =>  
      props.doLogin(state.username.value, state.password.value)}  
      Login  
    </Button>  
  </Form>);  
}
```

Form ist hier eine Art "logische" Komponente, die aus mehreren Teilen besteht.
Alle Teile haben automatisch Zugriff auf den State

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Noch mehr Consumer – alle brauchen den Formzustand für ein Feld

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Noch mehr Consumer – alle brauchen den Formzustand für ein Feld

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}  
  
function TextArea({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  ...  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Noch mehr Consumer – alle brauchen den Formzustand für ein Feld

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}
```

```
function TextArea({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  // formState[name] verwenden  
}
```

```
function MeineEigeneKomponente({name}) {  
  const { formState } = React.useContext(FormContext);  
  // formState[name] verwenden  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Noch mehr Consumer – alle brauchen den Formzustand für ein Feld

```
function TextField({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  return <input value={formState[name].value}  
            onChange={formState[name].onChange} />  
}
```

```
function TextArea({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  // formState[name] verwenden  
}
```

```
function MeineEigeneKomponente({name}) {  
  const { formState } = React.useContext(FormContext);  
  
  // formState[name] verwenden  
}
```

Redundanter Code
Implementierungsdetail
(Context)

State über Komponentengrenzen

Custom Hook für Context

```
function useFieldState(fieldName) {  
  const { formState } = React.useContext(FormContext);  
  
  return formState[fieldName];  
}
```

KOMMUNIKATION VON KOMPONENTEN

State über Komponentengrenzen

Custom Hook für Context

```
function useFieldState(fieldName) {  
  const { formState } = React.useContext(FormContext);  
  
  return formState[fieldName];  
}
```

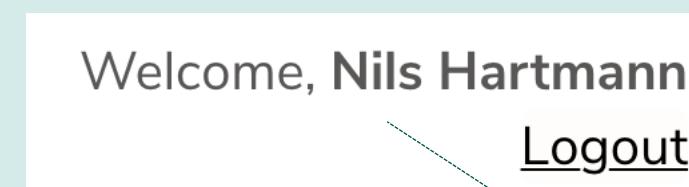
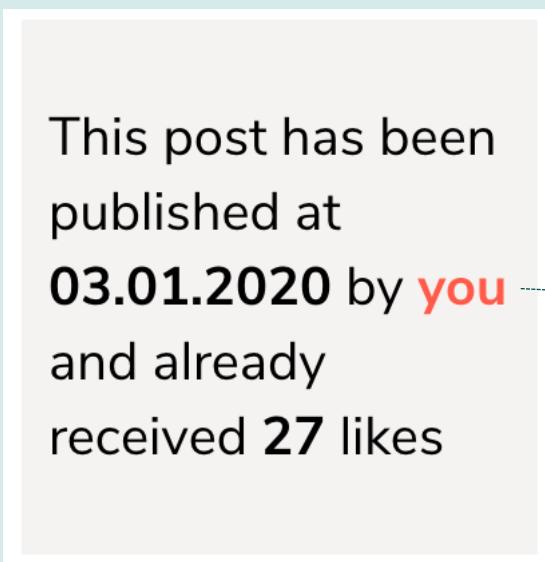
```
function InputField({name}) {  
  const { value, onChange } = useFieldState(name);  
  return <input value={value} onChange={onChange} />  
}
```

```
function TextArea({name}) {  
  const { value, onChange } = useFieldState(name);  
  return <textarea value={value} onChange={onChange} />  
}
```

GLOBALE DATEN

Idee: React Context auch für "echte" globale Daten

Beispiel: angemeldeter Benutzer



User-Objekt mit Daten und Funktionen liegt in einem Context



Provider-Komponente

- Genau wie beim FormContext

```
function AuthContextProvider(props) {  
  
  return (  
    <AuthContext.Provider value={ ... }>  
      { children }  
    </AuthContext.Provider>  
  );  
}
```

Provider-Komponente

- Wird ganz oben in der Hierarchie eingebunden, ganze Anwendung hat darauf Zugriff

```
function AuthContextProvider(props) {  
  
    return (  
        <AuthContext.Provider value={ ... }>  
            { children }  
        </AuthContext.Provider>  
    );  
}  
  
function App(props) {  
  
    return <AuthContextProvider>  
        <BlogPost /> ----- Ebenfalls wie beim FormContext  
    </AuthContextProvider>  
}
```

REACT CONTEXT

Provider-Komponente

- Provider stellt nicht nur Zustand, sondern auch Funktionen zur Verarbeitung Verfügung

```
function AuthContextProvider(props) {  
  const [ user, setUser ] = React.useState(...);  
  
  function login(username, password) {  
    loginViaHttp(...).then(response => setUser(response.user));  
  }  
  
  function logout() { setUser(null); }  
  
  return (...);  
}
```

Provider-Komponente

- Runterreichen von State und Callback-Funktionen

```
function AuthContextProvider(props) {  
  const [ user, setUser ] = React.useState(...);  
  
  function login(username, password) {  
    loginViaHttp(...).then(response => setUser(response.user));  
  }  
  
  function logout() { setUser(null); }  
  
  return (  
    <AuthContext.Provider value={ {  
      user, login, logout ----- Bereitgestellte Werte und Callback-Funktionen  
    } }>  
      { children }  
    </AuthContext.Provider>  
  );  
}
```

REACT CONTEXT

useContext: Verwendung des Contexts mit Hooks

```
function CurrentUser(props) {  
  const { user } = useContext(AuthContext);  
  
  return <div>Welcome, {user} /></div>  
}
```

REACT CONTEXT

useContext: Verwendung des Contexts mit Hooks

```
function LoginForm() {  
  const { login } = useContext(AuthContext);  
  
  return <Form>  
    <TextField name="username" />  
    <TextField name="password" />  
    <Button onClick={ formState =>  
      login(formState.username, formState.password) }>Login  
    </Button>  
  </Form>  
}
```

REACT CONTEXT

useReducer & useContext kombiniert

Wir können den AuthContext mit useReducer implementieren

Bereitgestellte Funktionen dispatchen dann Actions

REACT CONTEXT

useReducer & useContext kombiniert

Wir können den AuthContext mit useReducer implementieren

Bereitgestellte Funktionen dispatchen dann Actions

```
function authReducer(state, action) { ... }

function AuthContextProvider(props) {
  const [state, dispatch] = React.useReducer(authReducer);

  return (
    <AuthContext.Provider value={{
      user: state.user,
      login(u,p) { dispatch({type: "LOGIN", ...}) },
      logout() { dispatch({type: "LOGOUT", ...}) }
    }}>
      { children }
    </AuthContext.Provider>
  );
}
```

REACT CONTEXT

useReducer & useContext kombiniert

Wir können den AuthContext mit useReducer implementieren

Bereitgestellte Funktionen dispatchen dann Actions

- 👍 Globaler Zustand (kann ganz oben in der Hierarchie eingefügt werden)
 - Strukturierung nach Geschmack möglich (Zusand pro Anwendungsteil möglich)
- 👍 Geschäftslogik jetzt aus den Komponenten raus (dank reducer-Funktion)
- 👍 Technik ist Transparent für Consumer (kein dispatch-Aufruf...)

REACT CONTEXT

useReducer & useContext kombiniert

Wir können den AuthContext mit useReducer implementieren
Bereitgestellte Funktionen dispatchen dann Actions

- 👍 Globaler Zustand (kann ganz oben in der Hierarchie eingefügt werden)
Strukturierung nach Geschmack möglich (Zusand pro Anwendungsteil möglich)
- 👍 Geschäftslogik jetzt aus den Komponenten raus (dank reducer-Funktion)
- 👍 Technik ist Transparent für Consumer (kein dispatch-Aufruf...)

- 👎 Performance bei häufigen Änderungen?
- 👎 Actions, die von mehreren Reducern verarbeitet werden sollen?
- 👎 Tooling (Visualisierung der Änderungen am globalen Zustand)

Redux? 😊

Redux

GLOBALE DATEN

Beispiel: angemeldeter Benutzer

Ansatz 2: Redux

This post has been published at **03.01.2020** by **you** and already received **27** likes

Welcome, **Nils Hartmann**
[Logout](#)

User-Objekt mit Daten und Funktionen liegt im **globlen Redux Store**

03.01.2020

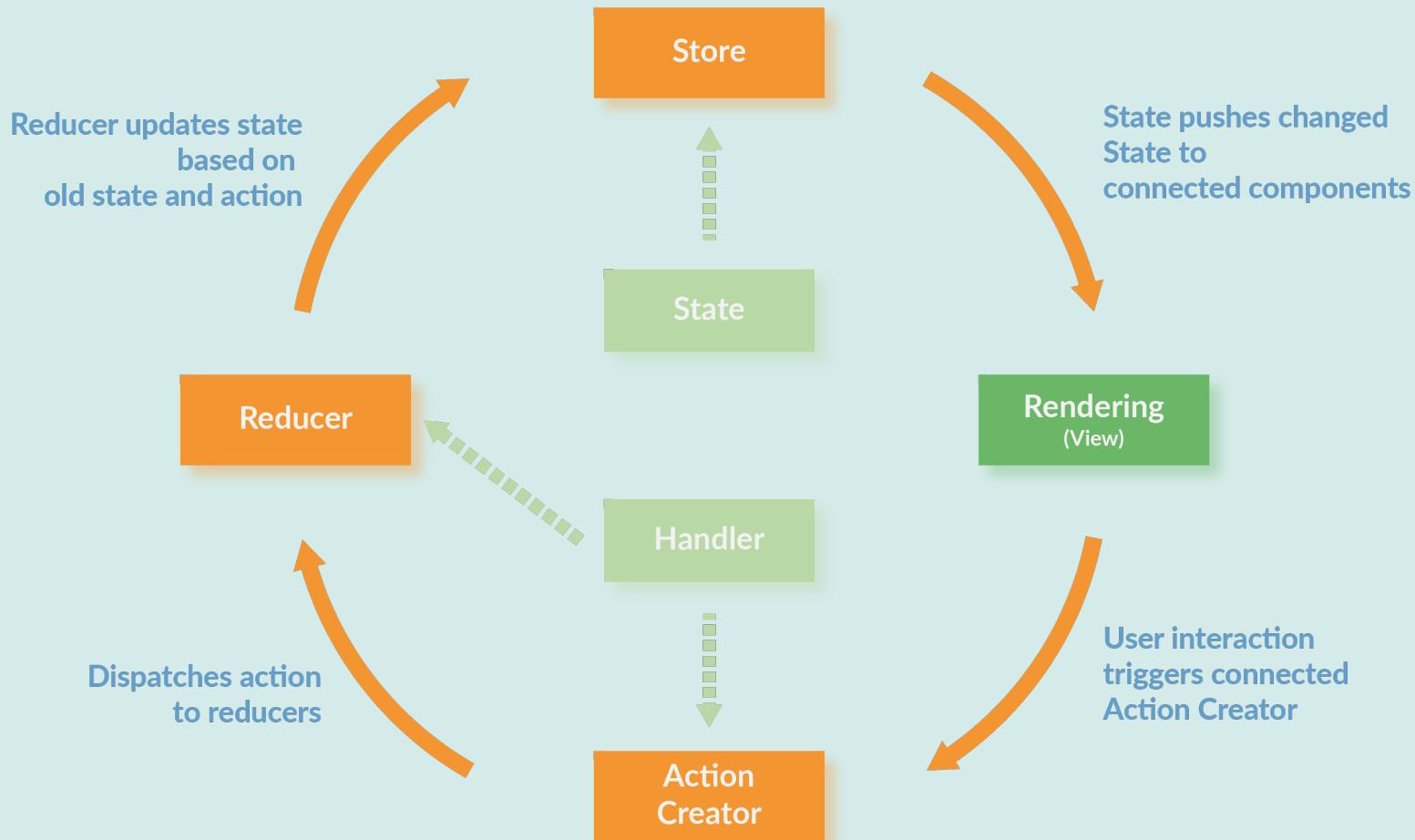
Routing Solutions for React

[Read more](#)

Your Post!

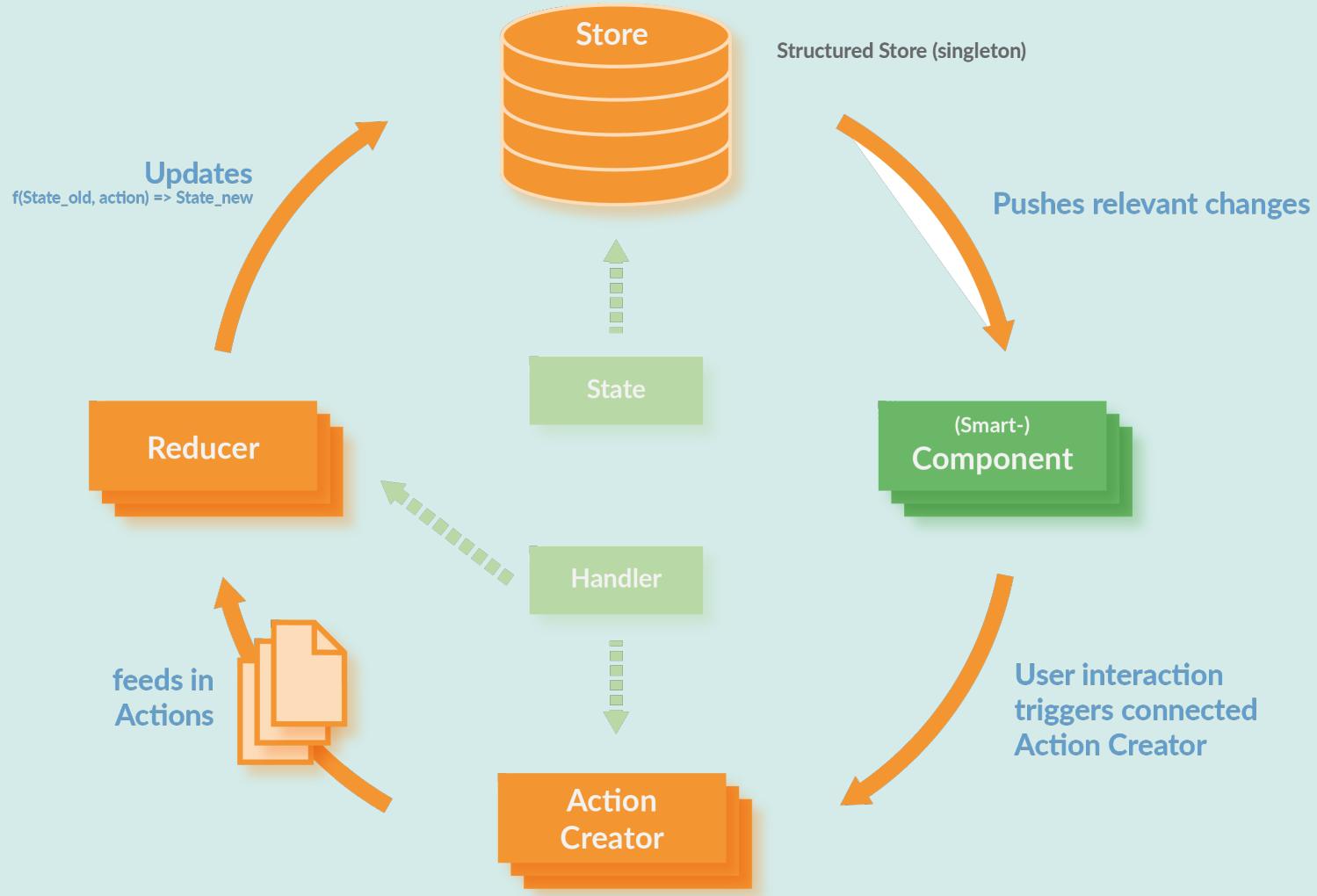
REDUX

Redux verschiebt Verantwortlichkeiten aus der React-Komponente



REDUX

Redux verschiebt Verantwortlichkeiten aus der React-Komponente



Redux verschiebt Verantwortlichkeiten aus der React-Komponente

- Der globale State ("Store") besteht aus mehreren Teilen ("Slices")
- Eine reducer-Funktion verwaltet jeweils einen Slice
- Die slices sind von einander getrennt
- Komponenten haben auf alle Slices zugriff



Beispiel: Redux mit connect-Funktion ("Klassiker")

- Im Gegensatz zu useReducer muss eine Komponente aus dem Store die benötigten Daten abfragen
- Nur wenn sich die abgefragten Daten ändern, wird die Komponente neu gerendert (nicht bei *jedem* Update im Store!)

Beispiel: Redux mit connect-Funktion ("Klassiker")

- Im Gegensatz zu useReducer muss eine Komponente aus dem Store die benötigten Daten abfragen
- Nur wenn sich die abgefragten Daten ändern, wird die Komponente neu gerendert (nicht bei jedem Update im Store!)

```
function UserProfile(props) {  
  return <div>  
    <h1>{props.username}</h1>  
    <button onClick={props.logout}>Logout</button>  
  </div>  
}  
  
export default connect(state => ({  
  username: state.auth.username  
}),  
  dispatch => ({ logout: () => dispatch(logout()) })  
);
```

GLOBALER ZUSTAND

Beispiel: Redux mit Hooks API ("modern")

```
function UserProfile(props) {  
  const username = useSelector(state => state.auth.username) ;  
  const dispatch = useDispatch() ;  
  
  return <div>  
    <h1>{username}</h1>  
    <button onClick={() => dispatch(logout())}>Logout</button>  
  </div>  
}
```

Beispiel: Custom Hook

```
function useUsername() {  
  return useSelector(state => state.auth.username);  
}  
  
function UserProfile(props) {  
  const username = useUsername();  
  
  return <div>  
    <h1>{username}</h1>  
    <button onClick={...}>Logout</button>  
  </div>  
}
```

GLOBALER ZUSTAND

Optimierung: (Komplexe) Auswahl aus dem globalen State

View History

11.12.2019

Styling your Components
23 Likes

02.04.2020

Using Redux with care
21 Likes

18.04.2020

Understanding State
19 Likes

Liste der zuletzt angeklickten Posts
(nur auf dem Client gehalten)

GLOBALER ZUSTAND

(Komplexe) Auswahl aus dem globalen State

View History

11.12.2019

Styling your Components
23 Likes

02.04.2020

Using Redux with care
21 Likes

18.04.2020

Understanding State
19 Likes

Liste der zuletzt angeklickten Posts
(nur auf dem Client gehalten)

```
▼ blog (pin)
  ▼ posts (pin)
    ▶ 0 (pin): { id: "P10", body: "Jemand mus...", likedBy: [], ... }
    ▶ 1 (pin): { id: "P8", likedBy: [...], title: "Increasing...", ... }
    ▶ 2 (pin): { id: "P9", body: "Duis autem...", likedBy: [], ... }
    ▶ 3 (pin): { id: "P4", likedBy: [], title: "Do's and d...", ... }
    ▶ 4 (pin): { id: "P2", likedBy: [], title: "My Story o...", ... }
    ▶ 5 (pin): { id: "P1", likedBy: [], title: "Routing So...", ... }
    ▶ 6 (pin): { id: "P6", body: "Normally, ...", likedBy: [], ... }
    ▶ 7 (pin): { id: "P5", likedBy: [], title: "Something ...", ... }
    ▶ options (pin): { orderBy: "date", direction: "desc" }
  ▶ api (pin): { description: "Loading Po...", loading: false, error: null }
  ▼ viewHistory (pin)
    ▼ postsViewed (pin)
      0 (pin): "P6"
      1 (pin): "P9"
      2 (pin): "P10"
```

Im Store:
Liste mit Ids und Liste mit Posts
("normalisiert")

GLOBALER ZUSTAND

(Komplexe) Auswahl aus dem globalen State

View History

11.12.2019

Styling your Components
23 Likes

02.04.2020

Using Redux with care
21 Likes

18.04.2020

Understanding State
19 Likes



Wann muss die History neu gerendert werden?

```
▼ blog (pin)
  ▼ posts (pin)
    ▶ 0 (pin): { id: "P10", body: "Jemand mus...", likedBy: [], ... }
    ▶ 1 (pin): { id: "P8", likedBy: [...], title: "Increasing...", ... }
    ▶ 2 (pin): { id: "P9", body: "Duis autem...", likedBy: [], ... }
    ▶ 3 (pin): { id: "P4", likedBy: [], title: "Do's and d...", ... }
    ▶ 4 (pin): { id: "P2", likedBy: [], title: "My Story o...", ... }
    ▶ 5 (pin): { id: "P1", likedBy: [], title: "Routing So...", ... }
    ▶ 6 (pin): { id: "P6", body: "Normally, ...", likedBy: [], ... }
    ▶ 7 (pin): { id: "P5", likedBy: [], title: "Something ...", ... }
    ▶ options (pin): { orderBy: "date", direction: "desc" }
  ▶ api (pin): { description: "Loading Po...", loading: false, error: null }
  ▼ viewHistory (pin)
    ▼ postsViewed (pin)
      0 (pin): "P6"
      1 (pin): "P9"
      2 (pin): "P10"
```

Im Store:
Liste mit Ids und Liste mit Posts
("normalisiert")

GLOBALER ZUSTAND

(Komplexe) Auswahl aus dem globalen State

View History

11.12.2019

Styling your Components
23 Likes

02.04.2020

Using Redux with care
21 Likes

18.04.2020

Understanding State
19 Likes



❓ Wann muss die History neu gerendert werden?

Enthaltener Post hat sich geändert

Neuer Post in Liste dazugekommen

```
▼ blog (pin)
  ▼ posts (pin)
    ► 0 (pin): { id: "P10", body: "Jemand mus...", likedBy: [], ... }
    ► 1 (pin): { id: "P8", likedBy: [...], title: "Increasing...", ... }
    ► 2 (pin): { id: "P9", body: "Duis autem...", likedBy: [], ... }
    ► 3 (pin): { id: "P4", likedBy: [], title: "Do's and d...", ... }
    ► 4 (pin): { id: "P2", likedBy: [], title: "My Story o...", ... }
    ► 5 (pin): { id: "P1", likedBy: [], title: "Routing So...", ... }
    ► 6 (pin): { id: "P6", body: "Normally, ...", likedBy: [], ... }
    ► 7 (pin): { id: "P5", likedBy: [], title: "Something ...", ... }
    ► options (pin): { orderBy: "date", direction: "desc" }
  ► api (pin): { description: "Loading Po...", loading: false, error: null }
  ▼ viewHistory (pin)
    ▼ postsViewed (pin)
      0 (pin): "P6"
      1 (pin): "P9"
      2 (pin): "P10"
```

GLOBALER ZUSTAND

Lösungen?

GLOBALER ZUSTAND

Lösungen?

So vielleicht?

```
function Sidebar() {  
  const allPosts = useSelector(state => state.blog.posts);  
  const postIds = useSelector(state => state.viewHistory.postsViewed);  
  
  const postsInSidesBar = allPosts.filter(p => postIds.includes(p.id));  
  
  return ...;  
}
```

GLOBALER ZUSTAND

Lösungen?

So vielleicht?

```
function Sidebar() {  
  const allPosts = useSelector(state => state.blog.posts);  
  const postIds = useSelector(state => state.viewHistory.postsViewed);  
  
  const postsInSidesBar = allPosts.filter(p => postIds.includes(p.id));  
  
  return ...;  
}
```

Wenn sich **irgendein** Post ändert (oder
dazu kommt), wird neu gerendert 😢

GLOBALER ZUSTAND

Lösungen?

So vielleicht?

```
function Sidebar() {  
  const allPosts = useSelector(state => state.blog.posts);  
  const postIds = useSelector(state => state.viewHistory.postsViewed);  
  
  const postsInSidesBar = allPosts.filter(p => postIds.includes(p.id));  
  
  return ...;  
}
```

Bei jedem rendern neu berechnen?

Lösung: reselect-Bibliothek

"Selektoren", die nur ausgeführt werden, wenn sich eine Abhängigkeit ändert

Lösung: reselect-Bibliothek

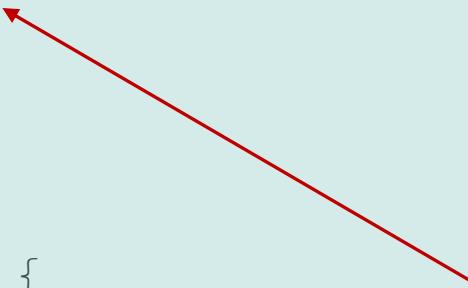
"Selektoren", die nur ausgeführt werden, wenn sich eine Abhängigkeit ändert

```
const selectAllPosts = state => state.blog.posts;
const selectViewedPostIds = state => state.viewHistory.postsViewed;

function Sidebar() {
  const allPosts = useSelector(state => state.blog.posts);
  const postIds = useSelector(state => state.viewHistory.postsViewed);

  const postsInSidesBar = allPosts.filter(p => postIds.include(p.id));

  return ...;
}
```



Lösung: reselect-Bibliothek

"Selektoren", die nur ausgeführt werden, wenn sich eine Abhängigkeit ändert

```
const selectAllPosts = state => state.blog.posts;
const selectViewedPostIds = state => state.viewHistory.postsViewed;

const selectViewedPosts = createSelector(
  [selectAllPosts, selectViewedPostIds],
  (allPosts, viewedPostIds) => // teure Filterlogik hier

function Sidebar() {
  const postsInSidesBar = useSelector(selectViewedPosts);

  return ...;
}
```

Redux oder Context?

- Redux hat mehr Features
 - Middlewares für Logging, Time Travelling etc
 - Developer Tools
 - Architektur-Modell (Reducer, Actions, ...)
- Redux ist global, Context prinzipiell auch für Teil-Anwendungen
- Context einfacher zu bedienen (?)
 - Durch Hooks API ist Redux aber etwas einfacher geworden
- Redux performanter
 - Context nur bei Daten, die sich nicht häufig ändern
- Redux und andere Ansätze widersprechen sich nicht
 - Lokaler Zustand weiterhin erlaubt

AUSBLICK: REDUX

Redux Toolkit <https://redux-toolkit.js.org/>

The official, opinionated, batteries-included toolset for efficient Redux development

- Default-Konfiguration mit TypeScript, Thunk u.a.
- Vereinfachte Reducer und Actions
- Spart viel Boilerplate-Code
- Bringt *reselect* und *immer* mit
- Template für create-react-app

MobX: "Simple, scalable state management"

- <https://github.com/mobxjs/mobx>
- Modelle werden als ES6-Klassen geschrieben und mit Dekoratoren versehen

MobX: "Simple, scalable state management"

- <https://github.com/mobxjs/mobx>
- Modelle werden als ES6-Klassen geschrieben und mit Dekoratoren versehen

```
import { observable, action } from "mobx"

class Auth {
  @observable userId = null;
  @observable userName = null;

}
```

ALTERNATIVEN

MobX: "Simple, scalable state management"

- <https://github.com/mobxjs/mobx>
- Modelle werden als ES6-Klassen geschrieben und mit Dekoratoren versehen

```
import { observable, action } from "mobx"

class Auth {
    @observable userId = null;
    @observable userName = null;

    @action.bound
    login(userId, userName) {
        this.userId = userId;
        this.userName = userName;
    }

    @action.bound
    logout() {
        this.userId = null; this.userName = null;
    }
}
```

MobX: "Simple, scalable state management"

- React Komponenten werden zu "Observern", die automatisch bei Aktualisierungen am Model aktualisiert werden

```
import { observer } from "mobx-react"

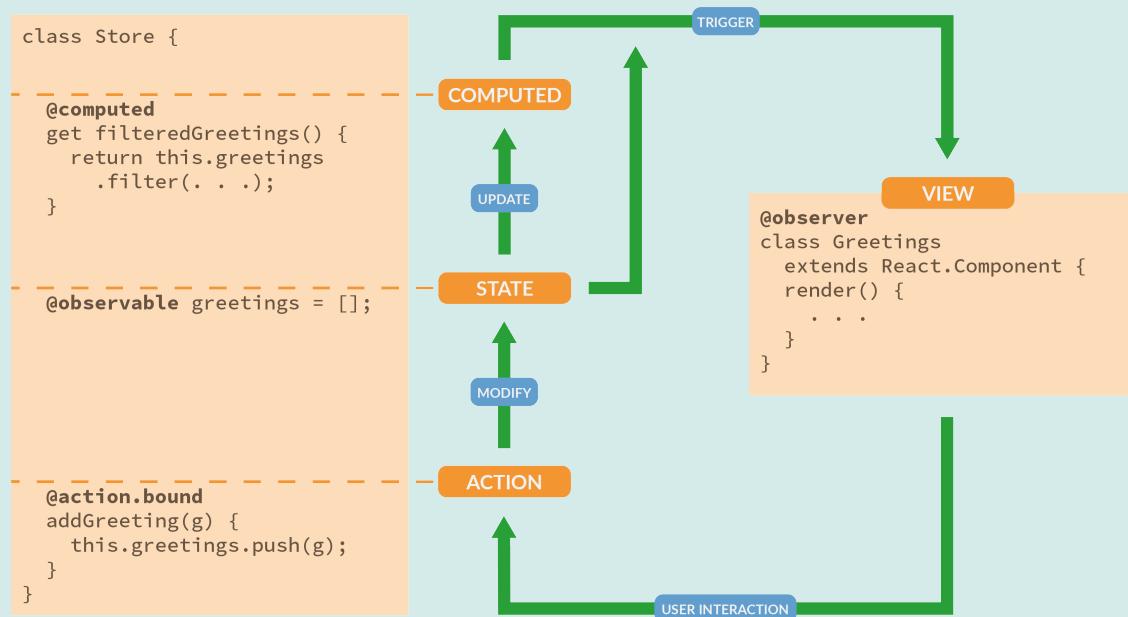
function UserBadge({auth}) {
    return <div>Welcome, {auth.userName}
        <button onclick={auth.logout}>Logout</button>
    </div>
}

export observer(UserBadge);
```

ALTERNATIVEN

MobX: "Simple, scalable state management"

- Datenfluss wie bei Redux in eine Richtung
- Aufteilung in State und Actions, auch ähnlich Redux
- Programmiermodell natürlich ganz anders
- Dev Tools ebenfalls vorhanden



ZUSAMMENFASSUNG

Es ist komplex...

- Viele Möglichkeiten, durch (Custom) Hooks noch mehr als früher
- Lokaler und globaler State schliessen sich nicht aus
 - State immer so nah wie möglich halten
- useReducer + Context != Redux
- Context kann (für einfache Fälle) eine Alternative sein
- Auf jeden Fall TypeScript
- Falls Redux, mit Redux Toolkit starten

NILS HARTMANN

<https://nilshartmann.net>

vielen Dank!

Slides: <https://nils.buzz/oose2020-react-state>

Source Code: <https://github.com/nilshartmann/react-state-example>

Fragen & Kontakt: nils@nilshartmann.net