

NILS HARTMANN

Moderne

# React

Pattern

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

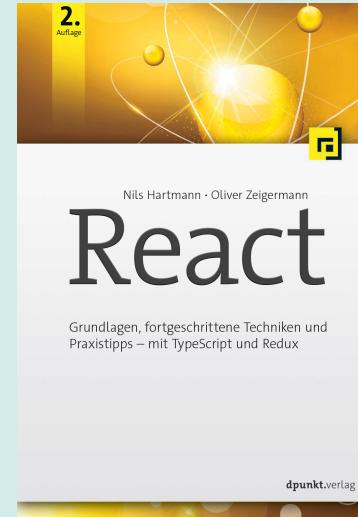
# NILS HARTMANN

nils@nilshartmann.net

**Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg**

Java  
JavaScript, TypeScript  
React  
GraphQL

**Trainings & Workshops**



<https://reactbuch.de>

**HTTPS://NILSHARTMANN.NET**

The screenshot shows a web browser window with the title "React Training Blog". The URL bar indicates the site is running on "localhost:3000". The top right corner shows a welcome message "Welcome, Nils Hartmann" and a "Logout" link. On the left, there's a button labeled "Add Post". The main content area displays four blog posts:

- Keep calm and learn GraphQL!** (Published 10.01.2020) - A red link to the post.
- Do's and don'ts with React** (Published 07.01.2020) - A red link to the post.
- My Story on JavaScript** (Published 04.01.2020) - A red link to the post.
- Routing Solutions for React** (Published 03.01.2020) - A red link to the post. To the right of this post is a small red button labeled "Your Post!".

Below each post is a "Read more" link. To the right of the posts is a sidebar containing a vertical list of links:

- [09.12.2019 Something to remember when learning new tech](#)
- [01.01.2020 Increasing React developer experience](#)
- [10.01.2020 Keep calm and learn GraphQL!](#)
- [03.01.2020 Routing Solutions for React](#)
- [07.01.2020 Do's and don'ts with React](#)

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

EIN BEISPIEL...

## HINTERGRUND: RENDER PROPERTIES

**Pattern: Render Properties**

## HINTERGRUND: RENDER PROPERTIES

**Render Properties:** Ein Property, das JSX-Elemente entgegen nimmt

- Genau wie children-Property, nur selbst definiert
- Beispiel: Layout-Komponente

```
function Layout(props) {  
  return <div className="Layout">  
    <div className="Left">{props.left}</div>  
    <div className="Right">{props.right}</div>  
  }  
}
```

## HINTERGRUND: RENDER PROPERTIES

**Render Properties:** Ein Property, das JSX-Elemente entgegen nimmt

- Genau wie children-Property, nur selbst definiert
- Beispiel: Layout-Komponente

```
function Layout(props) {  
  return <div className="Layout">  
    <div className="Left">{props.left}</div>  
    <div className="Right">{props.right}</div>  
  }  
}
```

```
function BlogListPage(props) {  
  return <Layout left={<BlogList />}  
                right={BlogListSidebar />}>  
}
```

## RENDER PROPERTIES

### Beispiel: Laden von Daten

- In unserer Anwendung werden an diversen Stellen Daten geladen
- Wir wollen gemeinsame Behandlung der Funktionalität
  - Error Handling, Lade Zustände etc

```
function BlogPage() {  
  // Daten laden  
  // Loading Indicator  
  // Error Handling  
  return // Daten rendern  
}
```

```
function BlogListPage() {  
  // Daten laden  
  // Loading Indicator  
  // Error Handling  
  return // Daten rendern  
}
```

## RENDER PROPERTIES

**Beispiel:** Generische DataLoader-Komponente

- „Infrastruktur“-Komponente, die Daten laden implementiert

```
class DataLoader extends React.Component {  
  state = { loading: true, data: null };  
  
  componentDidMount() {  
    fetch(this.props.url)  
      .then(data => setState({data, loading: false}));  
  }  
  
}  
}
```

## RENDER PROPERTIES

**Beispiel:** Generische DataLoader-Komponente

- „Infrastruktur“-Komponente, die Daten laden implementiert

```
class DataLoader extends React.Component {  
  state = { loading: true, data: null };  
  
  componentDidMount() {  
    fetch(this.props.url)  
      .then(data => setState({data, loading: false}));  
  }  
  
  render() {  
    // Kind-Komponente rendern und Properties (loading, data)  
    // übergeben....  
    return .... ? // WIE?  
  }  
}
```

# RENDER PROPERTIES

## Render Property als Funktion

- Function-as-a-Child (statt statischer Komponente!)
- Callback-Funktion liefert dann die Komponente zurück

```
class DataLoader extends React.Component {  
  state = { loading: true, data: null };  
  
  componentDidMount() {  
    fetch(this.props.url)  
      .then(data => this.setState({data, loading: false}));  
  }  
  
  render() {  
    // this.props.children ist eine Funktion!  
    return this.props.children({  
      loading: state.loading,  
      data: state.data  
    })  
  }  
}
```

## RENDER PROPERTIES

### Beispiel: DataLoader-Komponente - Verwendung

- Function-as-a-Child (statt statischer Komponente!)
- Callback-Funktion liefert dann die Komponente zurück

```
function BlogListPage(props) {  
  
  return <DataLoader url="http://api/posts">  
    {  
      ({ loading, data }) =>  
        loading ? <LoadingIndicator /> :  
          <BlogList posts={data}>  
    }  
  </DataLoader>  
}
```

## RENDER PROPERTIES

**Beispiel:** Mehrere Funktionen als Kind-Elemente

Lesbarkeit? 😬

Verständlichkeit (des Konzeptes)? 😳

```
function BlogListPage(props) {  
  
  return  
    <ApiConfiguration>  
      { config =>  
        <DataLoader url={config.url + "/posts"}>  
          { ({ loading, data }) =>  
            loading ? <LoadingIndicator /> :  
              <BlogList posts={data}>  
            }  
          </DataLoader>  
        }  
    </ApiConfiguration>  
}
```

## **HOOKS ALS ALTERNATIVE**

# React Hooks

(Ein Jahr! 🎂)

HOOKS ALS ALTERNATIVE

# Wer kennt die Hooks API nicht?



## HINTERGRUND

**Hooks API:** “Einhaken“ in den Lebenszyklus einer Komponente

- State, Context etc. auch in Funktionskomponenten

## HINTERGRUND

**Hooks API:** “Einhaken“ in den Lebenszyklus einer Komponente

- State, Context etc. auch in Funktionskomponenten

**Motivation:**

- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
- Bessere Wiederverwendbarkeit von Code

## HOOKS ALS ALTERNATIVE

### Hooks sind reguläre JavaScript-Funktionen

- Dürfen nur in Funktionskomponenten verwendet werden
- 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

# HOOKS ALS ALTERNATIVE

## Hooks sind reguläre JavaScript-Funktionen

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

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

# HOOKS ALS ALTERNATIVE

## Hooks sind reguläre JavaScript-Funktionen

```
// 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>
}
```

## HOOKS ALS ALTERNATIVE

### Beispiel DataLoader: Alternative zum Render Property

- Name, Signatur und Rückgabe kann frei gewählt werden

```
function useApi(url) {  
  const [loading, setLoading] = useState(true);  
  const [data, setData] = useState(null);  
  
  useEffect(() => {  
    const controller = new AbortController();  
    const signal = controller.signal;  
  
    fetch(url, { signal })  
      .then(response => response.json())  
      .then(data => setData(data))  
      .catch(error => console.error(error));  
  
    return () => controller.abort();  
  }, [url]);  
  
  return { loading, data };  
}
```

## HOOKS ALS ALTERNATIVE

### Beispiel DataLoader: Alternative zum Render Property

- Es kann eigener State definiert werden

```
function useApi(url) {  
  const [ apiState, setApiState ] =  
    React.useState({ loading: false, data: null });  
  
  // ...  
  
  return apiState;  
}
```

# HOOKS ALS ALTERNATIVE

## Beispiel DataLoader: Alternative zum Render Property

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

## HOOKS ALS ALTERNATIVE

### Beispiel DataLoader: Alternative zum Render Property

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

# HOOKS ALS ALTERNATIVE

## Beispiel DataLoader: Alternative zum Render Property

- Mehrere Hooks (vs. mehrere verschachtelte Komponenten)

```
function BlogListPage() {  
  const { config } = useConfiguration();  
  const { loading, data } = useApi(`${config.url}/posts`);  
  
  if (loading) {  
    return <LoadingIndicator />  
  }  
  
  return <BlogList posts={data} />  
}
```

## STATE IN HOOKS

### Exkurs: Einfacher State oder komplexer State?

```
// „Komplexer“ State
function useApi(url) {
  const [ apiState, setApiState ] =
    React.useState({ loading: false, data: null });
  ...
}
```

```
// „Einfacher“ State
function useApi(url) {
  const [ loading, setLoading ] = React.useState(false);
  const [ data, setData ] = React.useState(null);
  ...
}
```

### Exkurs: 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*)

### Exkurs: 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*)

Bei useState wird immer der gesamte State gesetzt!

- Im Gegensatz zu setState, dort wird er zusammengeführt

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 1: Reducer-Funktion (state, action) => newState

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 1: Reducer-Funktion (state, action) => newState

Actions sind einfache JavaScript-Objekte

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

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

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

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 1: Reducer-Funktion (state, action) => newState

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

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 1: Reducer-Funktion (state, action) => newState

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

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 1: Reducer-Funktion (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:** Redux für Komponenten?

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:** 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
- Im Gegensatz zu Redux
  - keine Developer Tools (Timetravelling, ...)
  - keine Middleware

# Globale Daten

TEIL 1

## GLOBALE DATEN

**Beispiel:** angemeldeter Benutzer

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

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

03.01.2020

**Routing Solutions for React**

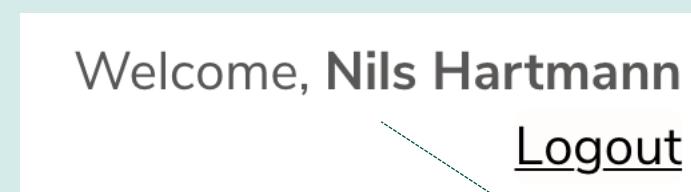
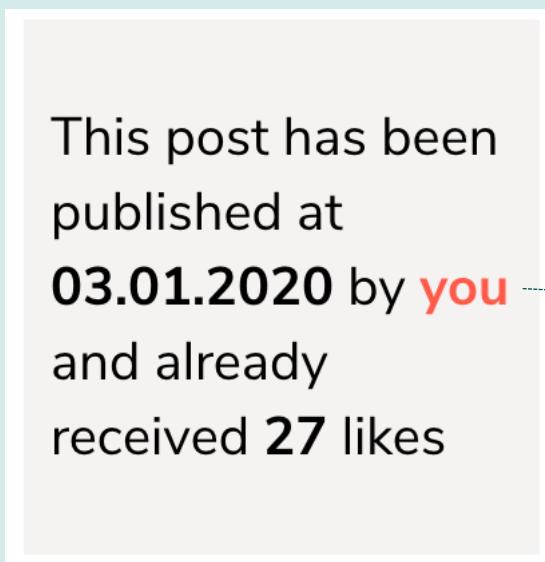
Your Post!

Read more

# GLOBALE DATEN

**Beispiel:** angemeldeter Benutzer

**Ansatz 1:** React Context



User-Objekt mit Daten und Funktionen liegt in einem Context



## REACT CONTEXT

**React Context:** Stellt Werte innerhalb einer Komponentenhierarchie zur Verfügung

- Erspart das Properties durchreichen
- Factory-Funktion erzeugt Provider und Consumer-Komponenten

```
const AuthContext = React.createContext(defaultValues);
```

## REACT CONTEXT

**React Context:** Stellt Werte innerhalb einer Komponentenhierarchie zur Verfügung

- Erspart das Properties durchreichen
- Factory-Funktion erzeugt Provider und Consumer-Komponenten

```
const AuthContext = React.createContext(defaultValues);
```

### Provider-Komponente

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

## REACT CONTEXT

**React Context:** Stellt Werte innerhalb einer Komponentenhierarchie zur Verfügung

- Erspart das Properties durchreichen
- Factory-Funktion erzeugt Provider und Consumer-Komponenten

```
const AuthContext = React.createContext(defaultValues);
```

### Provider-Komponente

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

### Provider-Komponente

- Kann die bereitgestellten Werte konsumieren

# REACT CONTEXT

## Provider-Komponente 1

- Stellt Werte für darunterliegende Komponenten zur Verfügung

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



Bereitstellte Werte

## Provider-Komponente 1

- Stellt Werte für darunterliegende Komponenten zur Verfügung

```
function AuthContextProvider(props) {  
  
  return (  
    <AuthContext.Provider value={ ... }>  
      { children }  
    </AuthContext.Provider>  
  );  
}  
  
function App(props) {  
  
  return <AuthContextProvider>  
    <BlogPost /> _____ Kann auf Kontext zugreifen  
  </AuthContextProvider>  
}
```

## Provider-Komponente 2

- 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

**Consumer:** Zugriff auf Daten aus dem Context

- Verwendet **Render Properties**

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

# REACT CONTEXT

**Consumer:** Modifzieren des Contexts

- Per Callback-Funktion (analog zu durchgereichten Properties)

```
function LoginForm(props) {  
  return (  
    <AuthContext.Consumer>  
    { ({ login }) => <form>  
      User: <input value="username" />  
      Password <input value="password" />  
      <button  
        onClick={() => login(username, password)}>  
        Login  
      </button>  
    </form>  
  )  
};
```

## 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>  
    User: <input value="username" />  
    Password <input value="password" />  
    <button onClick=  
      {() => login(username, password)}>Login  
    </button>  
  </form>  
}
```

# REACT CONTEXT

**useContext**: Verwendung des Contexts mit Hooks

Beispiel: Mehrere Contexte

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

## REACT CONTEXT

**Idee: Custom Hook** - „Fachlicher“ Zugriff auf Context

Versteckt, die Tatsache, dass es sich um einen Context handelt

```
function useAuth() {  
  // AuthContext ist Implementierungsdetail  
  
  const auth = useContext(AuthContext);  
  return auth;  
}  
  
function CurrentUser(props) {  
  const { user } = useAuth();  
}
```

## REACT CONTEXT

### useReducer & useContext – Redux Light?

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>
  );
}
```

# HIGHER ORDER COMPONENTS

**Pattern: Higher Order Components (HOC)**

## HIGHER ORDER COMPONENTS

**Higher Order Components (HOC)**: Vorgänger(?) von Render Props

- Erzeugen neue Komponente und wrappen bestehende
- Können Properties und Verhalten hinzufügen

## HIGHER ORDER COMPONENTS

**Higher Order Components (HOC)**: Vorgänger(?) von Render Props

- Erzeugen neue Komponente und wrappen bestehende
- Können Properties und Verhalten hinzufügen

```
// PostListPage weiß nichts davon, wie PostList Daten lädt
function PostListPage() {
  return <div className="PostList"><PostList /></div>
}
```

# HIGHER ORDER COMPONENTS

**Higher Order Components (HOC)**: Vorgänger(?) von Render Props

- Erzeugen neue Komponente und wrappen bestehende
- Können Properties und Verhalten hinzufügen

```
// PostListPage weiß nichts davon, wie PostList Daten lädt
function PostListPage() {
  return <div className="PostList"><PostList /></div>
}
```

```
// PostList weiß nichts, davon, dass/wie Daten geladen werden
// posts „normale“ Properties
function PostList( { posts } ) {
  return posts.map(post => <article>...</article>);
}
```

# HIGHER ORDER COMPONENTS

**Higher Order Components (HOC)**: Vorgänger(?) von Render Props

- Erzeugen neue Komponente und wrappen bestehende
- Können Properties und Verhalten hinzufügen

```
// PostListPage weiß nichts davon, wie PostList Daten lädt
function PostListPage() {
  return <div className="PostList"><PostList /></div>
}

// PostList weiß nichts, davon, dass/wie Daten geladen werden
// posts „normale“ Properties
function PostList( { posts } ) {
  return posts.map(post => <article>...</article>);
}

export default withDataLoader("http://api/posts")(PostList);
```



# HIGHER ORDER COMPONENTS

## HOC - Stark vereinfachtes Beispiel

- HOC-Funktion nimmt Komponente und ummantelt sie

```
function WithDataLoader(url)
  return RawComponent => {
    return class extends React.Component {
      state = { loading: false, data: null }
      componentDidMount() { // API Zugriff... }

      render() {
        if (this.state.loading) {
          return <LoadingIndicator />
        }
        return <RawComponent data={this.state.data} />
      }
    }
  }
}
```

# HIGHER ORDER COMPONENTS

## HOC – Probleme

- Schwer zu verstehen (Verwendung und Implementierung)
- Probleme mit kollidierenden Property-Namen („data“)
  - Ziel-Komponente muss wissen, unter welchen Namen die HOC Properties übergibt („data“ vs „posts“)

# HIGHER ORDER COMPONENTS

## HOC – Beispiele

- connect-Funktion von Redux (hat das Pattern populär gemacht)
- withRouter im React Router

## 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 globalen Redux Store

03.01.2020

**Routing Solutions for React**

[Read more](#)

Your Post!

## HIGHER ORDER COMPONENTS

### Beispiel: Redux mit connect-Funktion

```
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 (ab Version 7.1)

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

### Konsequenzen Redux Hooks API

- Code wird einfacher
- Komponente ist jetzt an Redux API gekoppelt
- useSelector funktioniert anders als zuvor
- useDispatch statt bindActionCreators

## 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
- Context nur in Fällen ohne viel Logik

### Szenarien für React Context

- Mehr oder minder statische, globale Daten
  - Aktueller Benutzer
- Für „teil-globale“ Daten um einen Zusammenschluss von Komponenten
  - Zum Beispiel Form-Komponente mit ihren Children
  - Context hält Validierungsinformationen etc

# REDUX MIT TYPESCRIPT

Redux mit TypeScript

## Getypter Zugriff auf globalen Zustand

- useSelector kann Typ-Parameter entgegen nehmen

```
function UserProfile(props) {  
  const username =  
    useSelector((state: AppState) => state.username);  
  
  // ...  
}
```

# REDUX MIT TYPESCRIPT

## Getypter Zugriff auf globalen Zustand

- Beispiel: Custom Hook für useSelector mit AppState (leicht vereinfacht)

```
// Kann in der kompletten Anwendung verwendet werden
function useAppSelector<R>(fn: (state: AppState) => R): R {
  return useSelector(fn)
}

function UserProfile(props) {
  // state ist hier AppState
  const username = useAppSelector(state => state.username);

  // ...
}
```

# REDUX MIT TYPESCRIPT

## Redux mit TypeScript

- Typ für den globalen Zustands kann abgeleitet werden
- Jede Reducer-Funktion liefert Teilzustand zurück
- combineReducers verbindet sie zu Gesamt-Zustand

```
const rootReducer = combineReducers({
  auth: authReducer,
  posts: postsReducer
});

export type AppState = ReturnType<typeof rootReducer>;
```

# REDUX MIT TYPESCRIPT

Action Creators noch zeitgemäß?

Beispiel: getypte Actions mit useDispatch

# REDUX MIT TYPESCRIPT

## Action Creators noch zeitgemäß?

Beispiel: getypte Actions mit useDispatch

```
export type LoginAction = {  
    type: "LOGIN",  
    username: string,  
    password: string  
}
```

```
export type LogoutAction = {  
    type: "LOGOUT"  
}
```

```
export type AppActions = LoginAction | LogoutAction;
```

# REDUX MIT TYPESCRIPT

## Action Creators noch zeitgemäß?

Beispiel: getypte Actions mit useDispatch

```
function LoginForm() {  
  const dispatch = useDispatch<Dispatch<AppActions>>();  
  
  // erlaubt:  
  dispatch({  
    type: "LOGIN",  
    username: "Klaus", password: "Secret"  
  });  
  
}
```

# REDUX MIT TYPESCRIPT

## Action Creators noch zeitgemäß?

Beispiel: getypte Actions mit useDispatch

```
function LoginForm() {  
  const dispatch = useDispatch<Dispatch<AppActions>>();  
  
  // erlaubt:  
  dispatch({  
    type: "LOGIN",  
    username: "Klaus", password: "Secret"  
  });  
  
  // compile Fehler:  
  dispatch({  
    type: "DO_LOGIN",  
    username: "Klaus", password: "Secret"  
  });  
}
```

# REDUX MIT TYPESCRIPT

## Action Creators noch zeitgemäß?

Custom Hook für useDispatch

```
function useAppDispatch() {  
  return useDispatch<Dispatch<AppActions>>();  
}
```

```
function LoginForm() {  
  const dispatch = useAppDispatch();  
  
  ...  
}
```

## Action Creators noch zeitgemäß?

- Durch Typisierung könnten Action Creator weggelassen werden
- Getype dispatch-Funktion stellt korrekte Verwendung sicher
- Action Creator können aber zum Beispiel
  - Parameter validieren (username !== "")
  - Default-Parameter anbieten

# REDUX MIT TYPESCRIPT

## Action-Typen können von Action Creator abgeleitet werden

```
function login(username: string, password: string) {  
    return {type: "LOGIN", username, password }  
}  
  
function logout() {  
    return { type: "LOGOUT" }  
}  
  
type LoginAction = ReturnType<typeof login>;  
type LogoutAction = ReturnType<typeof logout>;  
  
export type AppActions = LoginAction | LogoutAction;
```

# REDUX MIT TYPESCRIPT

Action-Typen können von Action Creator abgeleitet werden

getyptes Dispatch funktioniert trotzdem  
beide Ansätze können gemischt werden

```
function LoginForm() {  
  const dispatch = useDispatch<Dispatch<AppActions>>();  
  
  // erlaubt:  
  dispatch(login("Klaus", "Secret"));  
  
  // erlaubt:  
  dispatch({ type: "LOGOUT" });  
}
```

## 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
- Trotzdem einige neue Konzepte und Begriffe („Slices“)

# Suspense

RENDERN UNTERBRECHEN

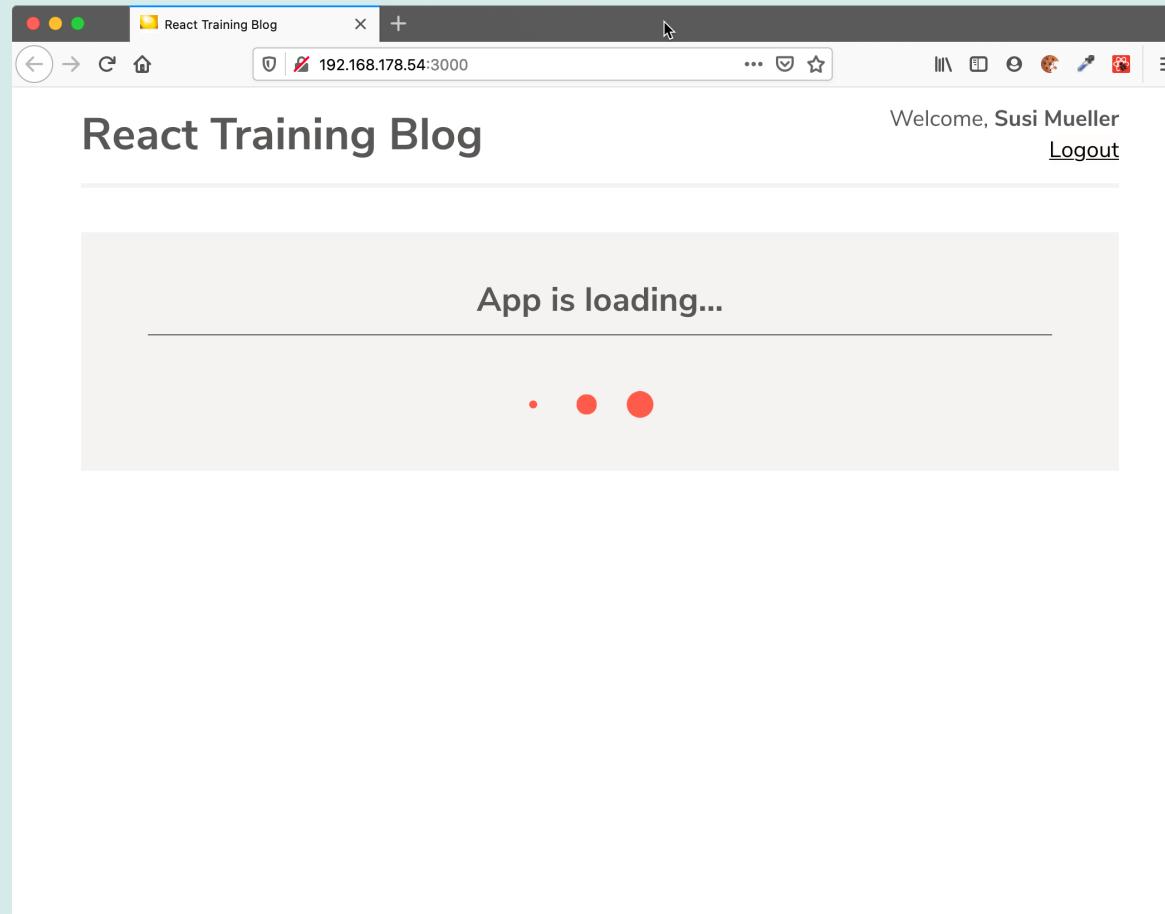
## SUSPENSE

**Suspense:** React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden

- Funktioniert aktuell für **Code Splitting (=> Code nachladen)**
- **Künftig** auch zum **Laden von beliebigen Daten** (z.Zt. experimentell)

## DEMO: LAZY UND SUSPENSE

- Mit dynamic Imports wird Code erst bei Bedarf geladen



# DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**

The screenshot shows a browser window for the "React Training Blog" at the URL `192.168.178.54:3000/add`. The page displays a "Create Post" form with fields for "Title" and "Body". Above the form, the user is logged in as "Susi Mueller" with a "Logout" link. The browser's developer tools Network tab is open, showing a list of network requests. A red box highlights the first request in the list:

Stat...	Met...	Domain	File	URL	Cause	Type	Transferred	Si...	0 ms
200	GET	192.168.17...	PostEditorPage.chunk.js	http://192.168.178.54:3000/static/j...	script	js	2.38 kB	1...	1 ms

At the bottom of the developer tools, it says "One request | 11.64 kB / 2.38 kB transferred | Finish: 1 ms".

# SUSPENSE

## React.lazy: Code splitting with Suspense

```
const LoginPage = React.lazy(() => import("./login/LoginPage"));  
Dynamic Import  
class App {  
  render() {  
    return <Switch>  
      <Route path="/login">  
        <LoginPage />  
      </Route>  
      // ...weitere Seiten...  
    </Switch>  
  }  
}
```

# SUSPENSE

**React.Suspense:** Zeigt „Fallback“-Komponente an

Bis Komponente geladen ist, muss Spinner o.ä. angezeigt werden

```
const LoginPage = React.lazy(() => import("./login/LoginPage"));

class App {
  render() {
    return <React.Suspense fallback={<LoadingIndicator />}>
      <Switch>
        <Route path=„/login“>
          <LoginPage />
        </Route>
        // ...weitere Seiten...
      </Switch>
    </React.Suspense>
  }
}
```

# **Concurrent Mode & Suspense for Data Fetching**

**AUSBLICK**

# Introducing Concurrent Mode (Experimental)

**Caution:**

This page describes **experimental features that are not yet available in a stable release**.

Don't rely on experimental builds of React in production apps. These features may change significantly and without a warning before they become a part of React.

This documentation is aimed at early adopters and people who are curious. If you're new to React, don't worry about these features — you don't need to learn them right now.

<https://reactjs.org/concurrent>

# Introducing Concurrent Mode (Experimental)

**Caution:**

This page describes **experimental features that are not yet available in a stable release**.

Don't rely on experimental builds of React in production apps. These features may change significantly and without a warning before they become a part of React.

This documentation is aimed at early adopters and people who are curious. **If you're new to React, don't worry about these features** — you don't need to learn them right now.

<https://reactjs.org/concurrent>

## Concurrent Mode 1

- Rendern ist eine "non-blocking" Operation
  - Es kann **immer** auf User-Interaktionen reagiert werden
- Updates können priorisiert werden

## Concurrent Mode 2

- Komponenten können u.a. vor-gerendert werden, ohne sofort sichtbar zu sein
  - Zum Beispiel beim **Laden von Code und Daten**
  - Verhindert überflüssige Warte- und Zwischen-Zustände
  - Komponenten müssen "etwas" haben, woher sie ihre Daten beziehen (gibt's aber noch nicht)
  - Erst wenn Komponente alle **gewünschten** Daten hat, wird sie angezeigt

# CONCURRENT REACT

- Focus im React-Team zzt. bei GraphQL/Relay mit Suspense

Nick Schrock  
@schrockn

Folgen ▾

From the talk about the rewrite of fb using Relay and GraphQL. This feature is so amazing and intuitive. Deliver js only if the graphql query returns data that requires that js.

Tweet übersetzen

AT&T LTE 8:52 AM 84%

Data-Driven Code-Splitting

Relay

```
... on Post {  
  ... on PhotoPost {  
    @module('PhotoComponent.js')  
    photo_data  
  }  
  ... on VideoPost {  
    @module('VideoComponent.js')  
    video_data  
  }  
  ... on SongPost {  
    @module('SongComponent.js')  
    song_data  
  }  
}
```

18:06 - 1. Mai 2019

<https://twitter.com/schrockn/status/1123619660732047360>

# CONCURRENT REACT

- Beispiel: Seite öffnen

<http://localhost:9081/?delay>

React Chat Example

React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

In the Office... Philosophy Coffee

involved and invested in our company and this is one way to do so. Curate.

 Harry  
Pushback.

 Peter  
Guerrilla marketing we don't want to boil the ocean we need to leverage our synergies touch base

 Maja  
The sprint is over please use "solutionise" instead of solution ideas! :).

 Sue  
Push back digitalize yet enough to wash your face, or low-hanging fruit horsehead offer, for Bob called an all-hands this afternoon that ipo will be a game-changer.

Anonymous-21 joined In the Office...

Please login to post messages [Login](#)

**Anonymous-21, you're not logged in.**  
To send messages, you need to login first

[Login](#)

Dashboard (Suspense)

Exit

<https://github.com/nilshartmann/react-chat-example>

# CONCURRENT REACT

**useTransition:** Übergänge, bei denen die Ziel-Komponente auf evtl. noch auf Daten wartet, müssen angegeben werden

- React verzögert den Übergang zur Ziel-Komponente
- Unnötige Warteindikatoren werden vermieden
- Ausgangskomponente kann anzeigen, dass gewartet wird

Code-Beispiel: ChatPage.js

```
export default function ChatPage() {
  const [setTransition, isPending] = useTransition(...);

  function openDashboard() {
    setTransition( () => setView("dashboard") );
  }

  return ...
  <Button onClick={openDashboard} pending={isPending}>
    Dashboard
  <Button>
}
```

## SUSPENSE FOR DATA FETCHING

- **Suspense for Data Fetching: Daten laden**

# SUSPENSE FOR DATA FETCHING

- **Beispiel: Admin Dashboard**

<http://localhost:9081/dashboard?delay>

/api/cpus

React Chat Example      React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard** Close

Server CPUs			
Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

# SUSPENSE FOR DATA FETCHING

- Beispiel: Admin Dashboard

<http://localhost:9081/dashboard?delay>

/api/cpus

/api/logs

React Chat Example  
React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard**

**Server CPUs**

Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

**Logs**

```
[Anonymous-14] Client registered
[Anonymous-14] join chatroom with id 'r1'
[Anonymous-14] client disconnected
[Anonymous-15] Assigned User id 'Anonymous-15'
[Anonymous-15] Client registered
[Anonymous-15] join chatroom with id 'r1'
[Anonymous-15] client disconnected
[Anonymous-16] Assigned User id 'Anonymous-16'
[Anonymous-16] Client registered
[Anonymous-16] join chatroom with id 'r1'
```

# SUSPENSE FOR DATA FETCHING

- Beispiel: Admin Dashboard

<http://localhost:9081/dashboard?delay>

/api/cpus

/api/logs

/api/users

React Chat Example      React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard** Close

Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

Logs	
[Anonymous-14]	Client registered
[Anonymous-14]	join chatroom with id 'r1'
[Anonymous-14]	client disconnected
[Anonymous-15]	Assigned User id 'Anonymous-15'
[Anonymous-15]	Client registered
[Anonymous-15]	join chatroom with id 'r1'
[Anonymous-15]	client disconnected
[Anonymous-16]	Assigned User id 'Anonymous-16'
[Anonymous-16]	Client registered
[Anonymous-16]	join chatroom with id 'r1'

User	
<b>Id</b>	<b>Name</b>
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

# SUSPENSE FOR DATA FETCHING

## • Beispiel: Admin Dashboard

<http://localhost:9081/dashboard?delay>

### Fachlich:

- Wo wollen wir warten?
- Welche Daten müssen da sein, damit Darstellung Sinn macht?

React Chat Example  
React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard**

**Server CPUs**

Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

**Logs**

```
[Anonymous-14] Client registered
[Anonymous-14] join chatroom with id 'r1'
[Anonymous-14] client disconnected
[Anonymous-15] Assigned User id 'Anonymous-15'
[Anonymous-15] Client registered
[Anonymous-15] join chatroom with id 'r1'
[Anonymous-15] client disconnected
[Anonymous-16] Assigned User id 'Anonymous-16'
[Anonymous-16] Client registered
[Anonymous-16] join chatroom with id 'r1'
```

**User**

Id	Name
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

**/api/cpus**

**/api/logs**

**/api/users**

# SUSPENSE FOR DATA FETCHING

## • Beispiel: Admin Dashboard

<http://localhost:9081/dashboard?delay>

### Fachlich:

- Wo wollen wir warten?
- Welche Daten müssen da sein, damit Darstellung Sinn macht?

### Technisch:

- Wie kommt die Seite an die Daten?
- Wie unterbrechen wir das Rendern?

React Chat Example  
React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard**

**Server CPUs**

Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

**Logs**

```
[Anonymous-14] Client registered
[Anonymous-14] join chatroom with id 'r1'
[Anonymous-14] client disconnected
[Anonymous-15] Assigned User id 'Anonymous-15'
[Anonymous-15] Client registered
[Anonymous-15] join chatroom with id 'r1'
[Anonymous-15] client disconnected
[Anonymous-16] Assigned User id 'Anonymous-16'
[Anonymous-16] Client registered
[Anonymous-16] join chatroom with id 'r1'
```

**User**

ID	Name
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

**/api/cpus**

**/api/logs**

**/api/users**

# SUSPENSE FOR DATA FETCHING

## • Beispiel: Admin Dashboard

<http://localhost:9081/dashboard?delay>

### Technisch:

- Wie kommt die Seite an die Daten?
  - Unklar! In Arbeit...
  - Nicht mehr mit useEffect, Komponenten-Lifecycle
- Wie unterbrechen wir das Rendern?
  - Suspense Komponente

React Chat Example  
React 16.11.0-experimental-f6b8d31a7 (with Hooks and more)

**Admin Dashboard**

**Server CPUs**

Model	Speed	User (ms)	Idle (ms)
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	20797620	133462890
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1460040	163234200
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16158880	142871250
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1510460	163107970
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16136390	142886280
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1512770	163103860
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	16121350	142914170
Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz	2500	1511200	163098080

**Logs**

```
[Anonymous-14] Client registered
[Anonymous-14] join chatroom with id 'r1'
[Anonymous-14] client disconnected
[Anonymous-15] Assigned User id 'Anonymous-15'
[Anonymous-15] Client registered
[Anonymous-15] join chatroom with id 'r1'
[Anonymous-15] client disconnected
[Anonymous-16] Assigned User id 'Anonymous-16'
[Anonymous-16] Client registered
[Anonymous-16] join chatroom with id 'r1'
```

**User**

Id	Name
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

**/api/cpus**

**/api/logs**

**/api/users**

# SUSPENSE FOR DATA FETCHING

**React.Suspense**: legt fest, wo in der Komponentenhierarchie gewartet werden soll

Code-Beispiel: DashboardWithSuspensePage.js

```
const dashboardData = loadDashboardData();

export default function DashboardPage({ onClose }) {
  return (
    <React.SuspenseList revealOrder="backwards">
      <React.Suspense fallback={<Spinner label="Loading Logs..." />}>
        <Logs logsResource={dashboardData.logs} />
      </React.Suspense>
      <React.Suspense fallback={<Spinner label="Loading User..." />}>
        <Users usersResource={dashboardData.users} />
      </React.Suspense>
    </React.SuspenseList>
  );
}
```

## SUSPENSE FOR DATA FETCHING

**React.Suspense**: legt fest, wo in der Komponentenhierarchie gewartet werden soll

Code-Beispiel: DashboardWithSuspensePage.js

```
function Logs({ logsResource }) {  
  const logs = logsResource.read(); ----- Hierauf wird gewartet  
  
  return (  
    <div>  
      <h2>Logs</h2>  
      <code>  
        {logs.map(l => (  
          <p key={l.eventId}>  
            [{l.user}] {l.msg}  
          </p>  
        ))}  
      </code>  
    </div>  
  );  
}
```

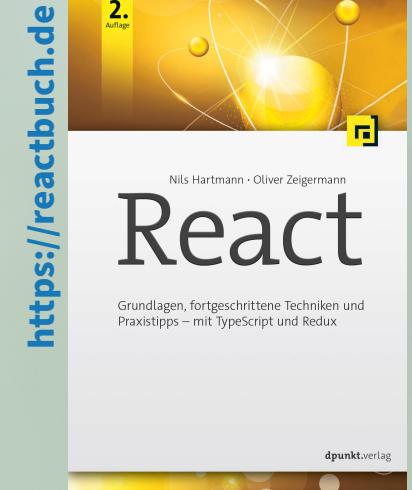
# CONCURRENT REACT

## Concurrent Mode: Aktueller Stand

- Experimentelle Version verfügbar, wird von FB produktiv eingesetzt
- Hat Veränderungen auf die Anwendungsarchitektur
  - Transitionen
  - Vorladen von Daten
- Ökosystem muss darauf vorbereitet sein
  - Router
  - Konzepte/Bibliotheken zum Vorladen von Daten
- Für wen ist der Suspense sinnvoll?

**NILS HARTMANN**

<https://nilshartmann.net>



# vielen Dank!

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

Source Code: <https://nils.buzz/react-blog-example>

Fragen & Kontakt: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

**NILS@NILSHARTMANN.NET**