The background of the slide is a photograph of a winding river or stream bed, likely a dry riverbed, with light brown, textured sand dunes on either side. The water is a bright blue-green color where it flows. In the upper left corner, there is a semi-transparent white rectangular overlay containing the text.

NILS HARTMANN

Moderne

React

Pattern

Slides: <https://react.schule/jax-2020>

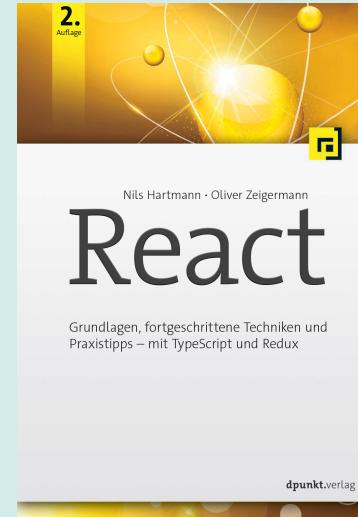
NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings & Workshops



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

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: 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() {  
  return <DataLoader url="...">  
    ??  
  </DataLoader>  
}
```

RENDER PROPERTIES

Beispiel: Generische DataLoader-Komponente

- "Infrastruktur"-Komponente, die Daten laden implementiert

```
function DataLoader(props) {  
  const state = React.useState({ loading: true, data: null });  
  
  React.useEffect( () => {  
    // vereinfacht  
    fetch(this.props.url)  
      .then(data => setState({data, loading: false}));  
  }  
}  
  
}
```

RENDER PROPERTIES

Beispiel: Generische DataLoader-Komponente

- "Infrastruktur"-Komponente, die Daten laden implementiert

```
function DataLoader(props) {  
  const state = React.useState({ loading: true, data: null });  
  
  React.useEffect( () => {  
    // vereinfacht  
    fetch(this.props.url)  
      .then(data => setState({data, loading: false}));  
  }  
  
  // Kind-Komponente rendern und Properties (loading, data)  
  // übergeben....  
  return .... ? // WIE? WAS?  
}
```

RENDER PROPERTIES

Render Property als Funktion

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

```
function DataLoader(props) {  
  const state = React.useState({ loading: true, data: null });  
  
  React.useEffect( () => {  
    // vereinfacht  
    fetch(this.props.url)  
      .then(data => setState({data, loading: false}));  
  }  
  
  // props.children ist eine Funktion!  
  return 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

RENDER PROPERTIES

Beispiel: Mehrere Funktionen als Kind-Elemente

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

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

HOOKS ALS ALTERNATIVE

Hooks API als Alternative

With Hooks, you can extract stateful logic from a component so it can be tested independently and reused. **Hooks allow you to reuse stateful logic without changing your component hierarchy.** This makes it easy to share Hooks among many components or with the community.

<https://reactjs.org/docs/hooks-intro.html>

- man kann eigene Hooks schreiben

HOOKS ALS ALTERNATIVE

Beispiel DataLoader: Alternative zum Render Property

- Name, Signatur und Rückgabe eines Hooks 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

Code ähnlich wie beim DataLoader...

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

HOOKS ALS ALTERNATIVE

Beispiel DataLoader: Alternative zum Render Property

...aber einfacher zu verwenden...

```
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

...auch bei mehreren Hooks

```
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 und bei denen keine inkonsistenten Zustände entstehen sollen

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

Beispiel: Lebenszyklus eines API Requests

```
const action = {  
  type: "LOAD_FINISHED", ----- Type  
  response: "... " ----- 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.response };  
  
    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(response => dispatch({type: "LOAD_FINISHED", response }))
  }, []);

  return state;
}
```

USERREDUCER HOOK

useReducer: Konsequenzen

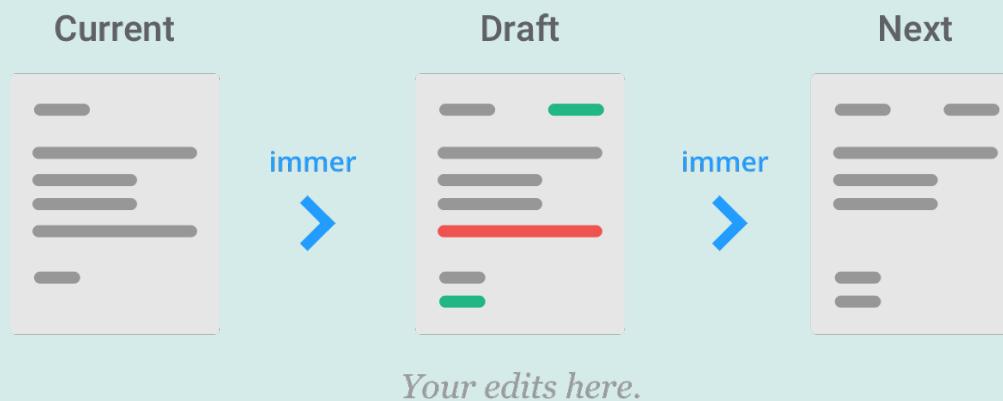
- Reducer Standard-JavaScript-Funktion, d.h. gut test- und wiederverwendbar
- Logik zur Behandlung des Zustandes an einer zentralen Stelle
- dispatch von Actions Code-intensiv
- Arbeiten mit immutable State anstrengend

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 apiReducer(oldState, action) {
  return produce(oldState, state => {
    switch (action.type) {
      case "FETCH_START": {
        state.loading = true;
        return;
      }
      // ...
    }
  });
}
```

state ist ein Proxy,
oldState bleibt
unverändert!

Sieht aus wie
"mutable Code",
keine zusätzliche
API notwendig

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

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>  
  );  
}
```

Bereitstellte Werte

```
function App(props) {  
  
  return <AuthContextProvider>  
    <BlogPost />  
  </AuthContextProvider>  
}
```

Kann auf Kontext zugreifen
(überall in der Hierarchie)

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

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();  
  const { color } = useTheme();  
  
  return ...;  
}
```

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

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()) })  
);
```

HIGHER ORDER COMPONENTS

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>  
}
```

HIGHER ORDER COMPONENTS

Konsequenzen (Redux) Hooks API statt HOC

- Code wird einfacher
- Komponente ist jetzt an Redux API gekoppelt

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
- Redux performanter
 - Einzel-Auswahl aus dem Store möglich
- 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

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

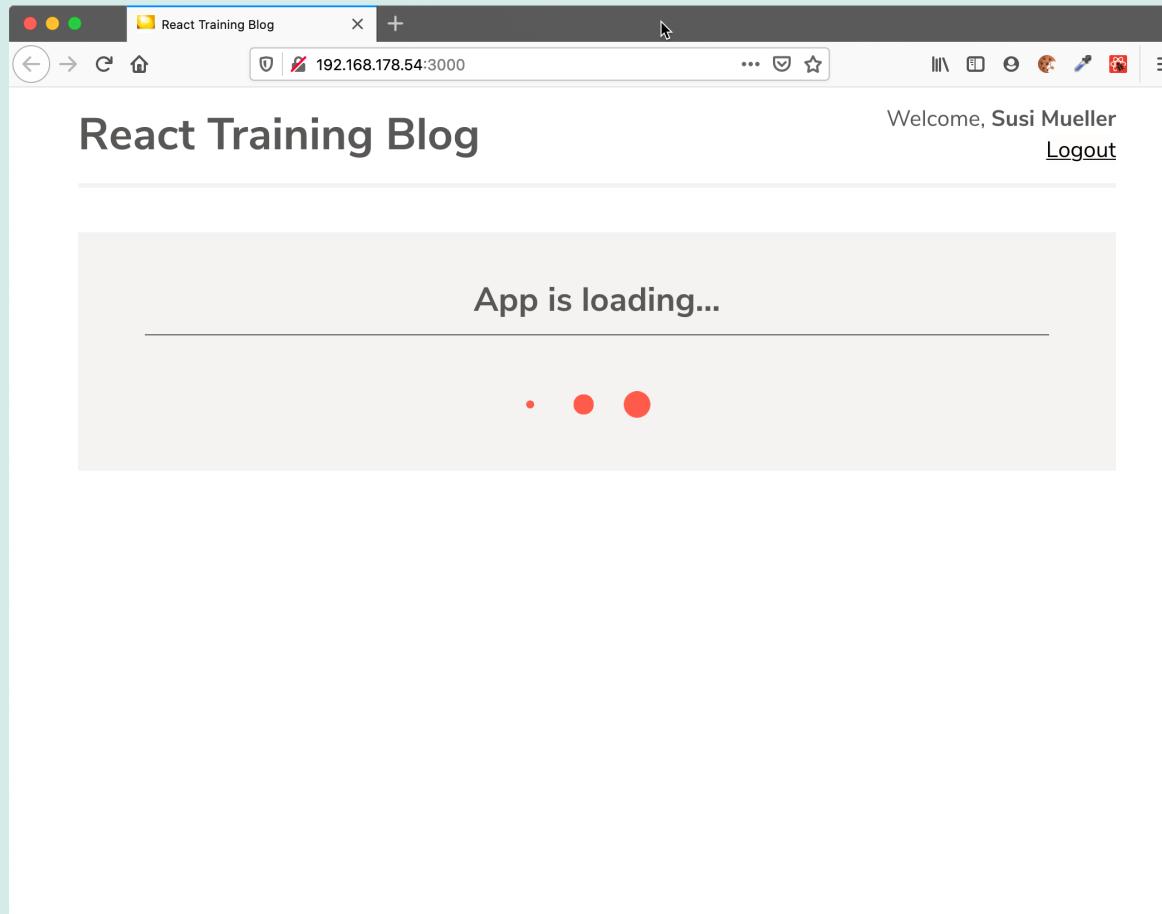
SUSPENSE

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 web browser window for a "React Training Blog" application. The URL in the address bar is 192.168.178.54:3000/add. The page displays a "Create Post" form with fields for "Title" and "Body". Above the form, the text "Welcome, Susi Mueller" and a "Logout" link are visible. The browser's developer tools are open, specifically the Network tab, which lists 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>

SEPTEMBER 2020 😮

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

- 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](#)

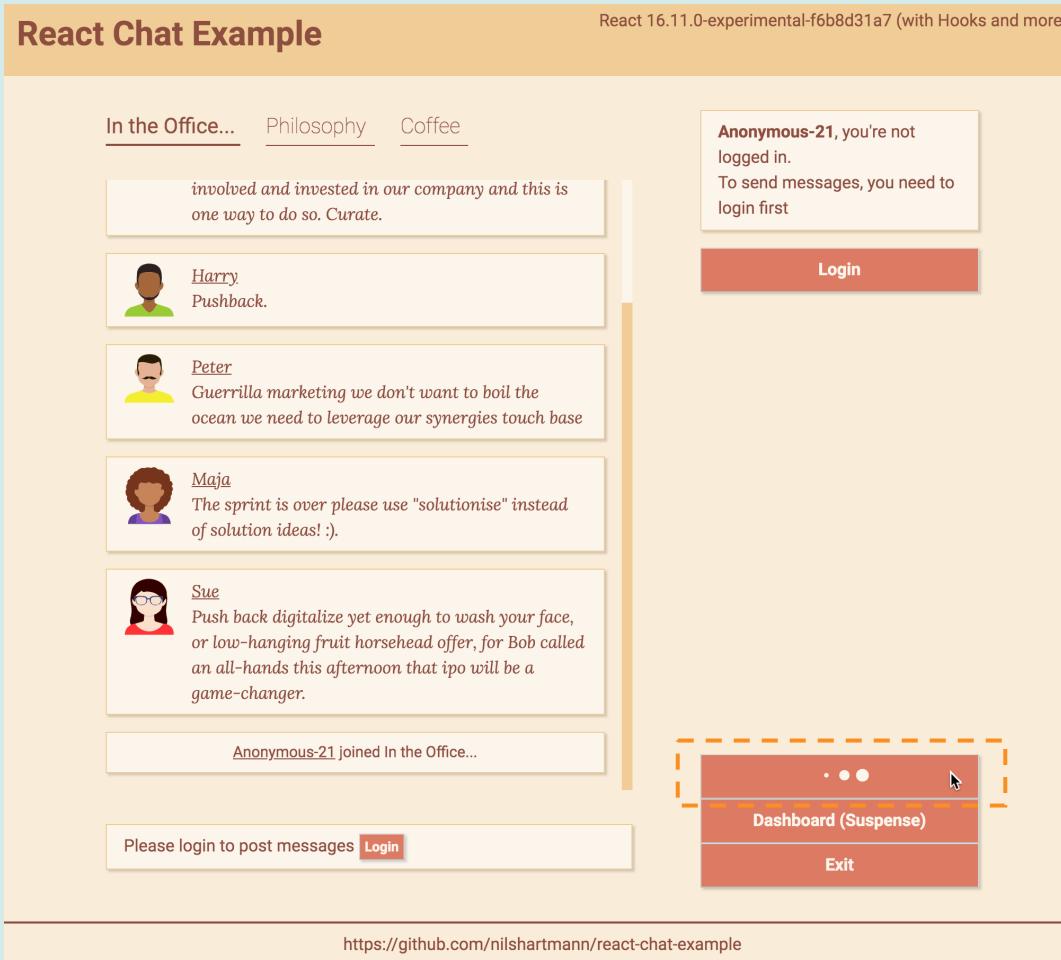
Anonymous-21, you're not logged in.
To send messages, you need to login first

[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)

Close

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

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	
Id	Name
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>  
  );  
}
```

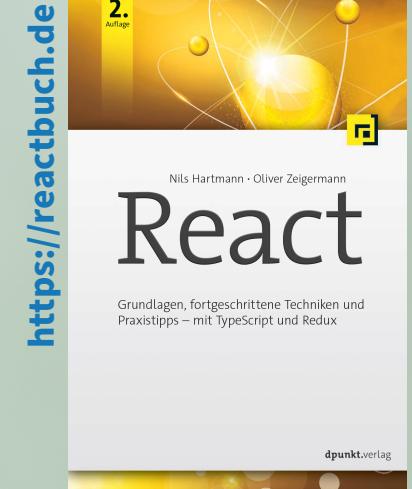
CONCURRENT REACT

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://react.schule/jax-2020>

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

Fragen & Kontakt: nils@nilshartmann.net

NILS@NILSHARTMANN.NET