

**NILS HARTMANN**  
<https://nilshartmann.net>

**State of the Art**

# React 2019

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

# NILS HARTMANN

nils@nilshartmann.net

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

Java  
JavaScript, TypeScript  
React  
GraphQL

Trainings, Workshops und  
Beratung



**2. Auflage, Dez. 2019**

**HTTPS://NILSHARTMANN.NET**

# **November 2018...**

**EIN BLICK ZURÜCK**

We plan to split the rollout of new React features into the following milestones:

- React 16.6 with Suspense for Code Splitting (*already shipped*)
- A minor 16.x release with React Hooks (~Q1 2019)
- A minor 16.x release with Concurrent Mode (~Q2 2019)
- A minor 16.x release with Suspense for Data Fetching (~mid 2019)

<https://reactjs.org/blog/2018/11/27/react-16-roadmap.html>

Weiterhin nur Minor-Versionen (!)

We plan to split the rollout of new React features into the following milestones:

- React 16.6 with Suspense for Code Splitting (*already shipped*)
- A minor 16.x release with React Hooks (~Q1 2019) ----- **16.8 (Februar 2019)**
- A minor 16.x release with Concurrent Mode (~Q2 2019)
- A minor 16.x release with Suspense for Data Fetching (~mid 2019)

<https://reactjs.org/blog/2018/11/27/react-16-roadmap.html>

Weiterhin nur Minor-Versionen (!)

# An Update to the Roadmap

In November 2018, we have posted this roadmap for the 16.x releases:

- A minor 16.x release with React Hooks (past estimate: Q1 2019)
- A minor 16.x release with Concurrent Mode (past estimate: Q2 2019)
- A minor 16.x release with Suspense for Data Fetching (past estimate: mid 2019)

These estimates were too optimistic, and we've needed to adjust them.

**tldr:** We shipped Hooks on time, but we're regrouping Concurrent Mode and Suspense for Data Fetching into a single release that we intend to release later this year.

<https://reactjs.org/blog/2019/08/08/react-v16.9.0.html>

# An Update to the Roadmap

In November 2018, we have posted this roadmap for the 16.x releases:

- A minor 16.x release with React Hooks (past estimate: Q1 2019)
- ~~A minor 16.x release with Concurrent Mode (past estimate: Q2 2019)~~
- ~~A minor 16.x release with Suspense for Data Fetching (past estimate: mid 2019)~~

These estimates were too optimistic, and we've needed to adjust them.

**tldr:** We shipped Hooks on time, but we're regrouping Concurrent Mode and Suspense for Data Fetching into a single release that we intend to release later this year.

<https://reactjs.org/blog/2019/08/08/react-v16.9.0.html>

React Conf 2019 - Fall 2019 in Lake Las Vegas, Nevada

React Conf 2019 - Fall 2019 in L X +

← → ⌂ ⌄ https://conf.reactjs.org/ir Search

**REACT**  
CONF 2019

SPEAKERS   SCHEDULE   VENUE   TICKETS

OCTOBER 24 & 25  
HENDERSON, NEVADA

The screenshot shows a web browser window for the React Conf 2019 website. The title bar reads "React Conf 2019 - Fall 2019 in Lake Las Vegas, Nevada". The address bar shows the URL "https://conf.reactjs.org/ir". The page features a dark blue header with the "REACT CONF 2019" logo on the left and navigation links for "SPEAKERS", "SCHEDULE", "VENUE", and "TICKETS" on the right. Below the header, large white text displays the event dates "OCTOBER 24 & 25" and location "HENDERSON, NEVADA" over a background of abstract purple and red wavy lines. The overall design is modern and professional.

HEUTE & MORGEN 😲

The image displays two side-by-side screenshots of a web application titled "React Chat Example" running on localhost:9081. The left screenshot is for React 16.6.0 and the right is for React 16.7.0-alpha.0.

**React 16.6.0 (Left):**

- Header:** "React Chat Example" and "16.6.0".
- Navigation:** "In the Office..." (highlighted), "Philosophy", "Coffee".
- Content:**
  - A message from "Anonymous-620" stating: "I also believe it's important for every member to be involved and invested in our company and this is one way to do so. Curate." with a "Login" button below it.
  - Messages from "Harry", "Peter", "Maja", and "Sue" with their respective messages.
  - Logins and logouts: "Anonymous-620 joined in the Office...", "Anonymous-621 joined in the Office...", "Anonymous-617 left in the Office...".
  - A message from "Klaus" stating: "Anonymous-621 logged in as Klaus".
  - A footer message: "Please login to post messages" with a "Login" button.
- Footer:** "https://github.com/nilshartmann/react-chat-example"

**React 16.7.0-alpha.0 (Right):**

- Header:** "React Chat Example" and "React 16.7.0-alpha.0".
- Navigation:** "In the Office..." (highlighted), "Philosophy", "Coffee".
- Content:**
  - A message from "Anonymous-620" stating: "I also believe it's important for every member to be involved and invested in our company and this is one way to do so. Curate." with a "Login" button below it.
  - Messages from "Harry", "Peter", "Maja", and "Sue" with their respective messages.
  - Logins and logouts: "Anonymous-620 joined in the Office...", "Anonymous-621 joined in the Office...", "Anonymous-617 left in the Office...".
  - A message from "Klaus" stating: "You're logged in as Klaus".
  - An "Add Message" input field and a "Send" button.
  - A sidebar with three buttons: "Dashboard (Effects)", "Dashboard (Suspense)", and "Exit".
- Footer:** "https://github.com/nilshartmann/react-chat-example"

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

EIN BEISPIEL...

**16.8**

# Hooks

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

**FUNCTIONS EVERYWHERE**

## HINTERGRUND

**Hooks:** State, Context, Lifecycle etc auch in Funktionskomponenten

## HINTERGRUND

**Hooks:** State, Context, Lifecycle etc auch in Funktionskomponenten

**Hooks sind reguläre Funktionen, aber...**

- **nur in** Funktionskomponenten (oder anderen Hooks) erlaubt
- **müssen** auf Top-Level-Ebene stehen (nicht in Schleife, if, ...)
- **müssen** mit "use" benannt werden

## USECONTEXT HOOK

**useContext:** Vereinfachter Zugriff auf den Context



You're logged in as **Maja**

User-Objekt liegt in einem Context



Maja You!

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

## USECONTEXT HOOK

**useContext**: Vereinfachter Zugriff auf den Context

- **Bisher**: Zugriff über Consumer-Komponente und Render Properties

```
function CurrentUserProfile(props) {  
  
  return (  
    <ChatContext.Consumer>  
      {chatValue => {  
        return <div>  
          <Avatar userId={chatValue.user.id} />  
          You're logged in as {chatValue.user.name} />  
        </div>;  
      }}  
    </ChatContext.Consumer>  
  );  
}
```

## USECONTEXT HOOK

**useContext**: Vereinfachter Zugriff auf den Context

- **Bisher**: Zugriff über Consumer-Komponente und Render Properties
- Unübersichtlich bei mehreren Kontexten

```
function CurrentUserProfile(props) {  
  
  return (  
    <ChatContext.Consumer>  
      {chatValue => {  
        return <ThemeContext.Consumer>  
          { themeValue => {  
            return <div className={themeValue.name}>  
              <Avatar userId={chatValue.user.id} />  
              You're logged in as {chatValue.user.name} />  
            </div>;  
          }  
        }  
      }  
    </ThemeContext.Consumer>  
  }  
}</ChatContext.Consumer>  
);
```

## USECONTEXT HOOK

**useContext**: Vereinfachter Zugriff auf den Context

- Hook: "normale" Funktion, Komponente wird gerendert, wenn Context sich ändert

```
import { useContext } from "react";

function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

}
```

## USECONTEXT HOOK

**useContext**: Vereinfachter Zugriff auf den Context

- Hook: "normale" Funktion, Komponente wird gerendert, wenn Context sich ändert

```
import { useContext } from "react";

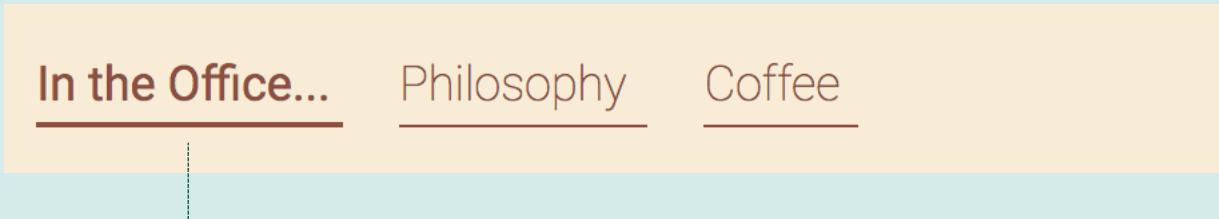
function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

  return (
    <div className={themeValue.name}>
      <Avatar userId={chatValue.user.id} />
      You're logged in as {chatValue.user.name} />
    </div>
  );
}
```

# USESTATE HOOK

**useState:** State in Funktionskomponenten

**Beispiel: Tab Bar**



Zustand: welche Tab ist geöffnet?

# USESTATE HOOK

## useState: State erzeugen

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
}  
  
|  
Aktueller State  
|  
Setter  
|  
Initialer Wert  
|  
}  
}
```

# USESTATE HOOK

## useState: Aktuellen State verwenden

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}  
        />  
      })}  
    </div>  
  );  
}
```

Zugreifen auf State

# USESTATE HOOK

## useState: State verändern

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}  
          onClick={() => setActiveTabId(tab.id)}  
        />  
      })}  
    </div>  
  );  
}
```

**Setzen von State  
(kein Objekt mehr!)**

## USESTATE HOOK

**useState**: Mehrere States in einer Komponente möglich

- Kein "mergen" von State mehr!

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

# USERREDUCER HOOK

**useReducer:** Redux für Komponenten?

- Für komplexen State mit viel Logik zur Veränderung

Login

Username

Sue

Password

.....

Cancel

Reset

Login

## 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: "SET_USER", ----- Type
  username: "..." ----- Payload
}
```

```
const action = {
  type: "SET_PASSWORD",
  password: "..."
}
```

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

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

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

```
function loginReducer(oldState, action) {  
  switch (action.type) {  
    case "SET_USER":  
      return action.user; // action.user ist ein Objekt mit dem Benutzernamen  
  }  
}  
}
```

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

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

```
function loginReducer(oldState, action) {  
  switch (action.type) {  
    case "SET_USER":  
      return {...oldState, username: action.username};  
  }  
}  
}
```

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

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

```
function loginReducer(oldState, action) {  
  switch (action.type) {  
    case "SET_USER":  
      return {...oldState, username: action.username};  
  
    case "SET_PASSWORD":  
      return {...oldState, password: action.password};  
  
    case "RESET":  
      return { user: "", password: "" };  
  
    default:  
      return throw new Error("Invalid action!");  
  }  
}
```

# USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 2: Verwendung

```
function loginReducer() { ... }

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

  return (>
    </>);
}
```

## USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 2a: Zugriff auf den State

```
function loginReducer() { ... }

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

  return (<>
    <input value={state.username}>

    />

    <input value={state.password}>

    />

    <button />;
  </>);
}
```

# USERREDUCER HOOK

**useReducer**: Redux für Komponenten?

Schritt 2b: Verändern des States über Actions

```
function loginReducer() { ... }

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

  return (<>
    <input value={state.username}
      onChange={e =>
        dispatch({type: "SET_USER", username: e.target.value})} />

    <input value={state.password}
      onChange={e =>
        dispatch({type: "SET_PASSWORD", password: e.target.value})}/>

    <button onClick={() => dispatch({type: "CLEAR"})};
  </>);
}
```

## 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

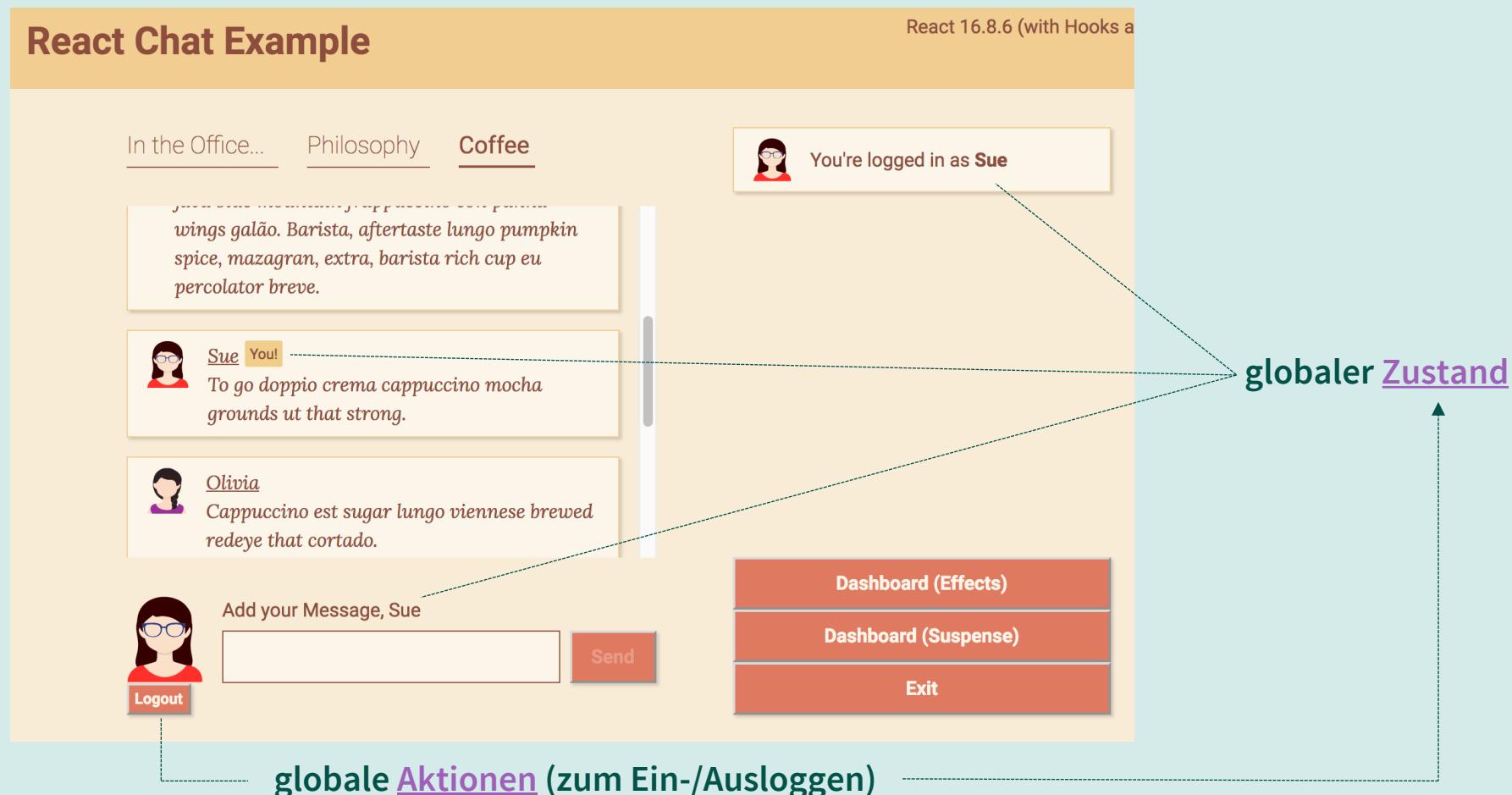
# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

Beispiel: globale Logik für Login/Logout bzw. angemeldeten User



# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

**Beispiel: Ein Reducer für Aktionen zum Ein-/Ausloggen**

```
function authenticationReducer(state, action) {  
  switch (action.type) {  
    case "LOGIN":  
      return { user: action.user };  
  
    case "LOGOUT":  
      return { user: null };  
  
    default:  
      throw new Error("...");  
  }  
}
```

# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Über Context werden Zustand und Aktionen nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

```
const AuthDispatcherCtx = React.createContext(null);  
  
function AuthProvider(props) {
```

```
}
```

# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Über Context werden State und Dispatch nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

```
const AuthDispatcherCtx = React.createContext(null);

function AuthProvider(props) {
  const [authState, dispatch] = useReducer(authenticationReducer);

}

}
```

# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Über Context werden State und Dispatch nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

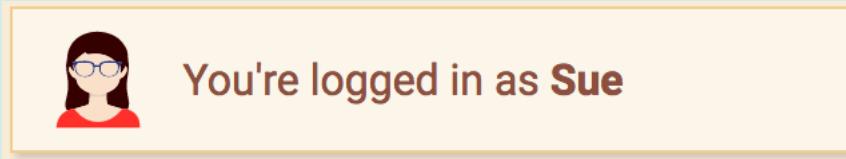
```
const AuthDispatcherCtx = React.createContext(null);

function AuthProvider(props) {
  const [authState, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcherCtx.Provider
      value={{authState, dispatch}}>
      {props.children}
    </AuthDispatcherCtx.Provider>
  );
}
```

# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Unterkomponenten können den State verwenden



**Beispiel: Verwendung innerhalb der Anwendung – State verwenden**

```
function Avatar(props) {  
  const {authState} = useContext(AuthDispatcher);  
  
  return (  
    <h1>You're logged in as {authState.user}</h1>  
  );  
}
```

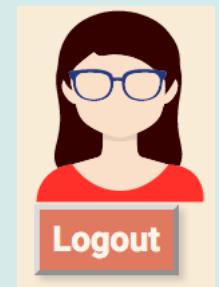
# ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Unterkomponenten können dispatch-Funktion dann verwenden
- (ähnlich wie in Redux)

**Beispiel: Verwendung innerhalb der Anwendung – Actions auslösen**

```
function LogoutButton(props) {  
  const {dispatch} = useContext(AuthDispatcher);  
  
  return (  
    <button onClick={() => dispatch({type: "LOGOUT"})}>  
      Logout  
    </button>  
  );  
}
```



## ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

**useContext & useReducer:** Für globalen App State?

- Schwäche 1: Verwender muss Context API verwenden (eigentlich Implementierungsdetail)
- Schwäche 2: Verwender muss dispatch verwenden und Action-Objekte selber erzeugen

## ARCHITEKTUR IDEE: CUSTOM HOOKS

**CustomHooks:** Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen

## ARCHITEKTUR IDEE: CUSTOM HOOKS

**CustomHooks:** Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen

👉 Damit wäre "fachliche" API möglich, um auf dispatch zu verzichten (ähnlich Action Creator in Redux)

👉 Context API und useReducer wird zum Implementierungsdetail und könnte z.B. durch Redux ersetzt werden

# ARCHITEKTUR IDEE: CUSTOM HOOKS

## CustomHooks: Für globalen App State?

- Custom Hook kapselt Zugriff auf Context, dispatch und state
- Stellt fachliche Methoden zur Verfügung

### Beispiel: Custom Hook

```
function useAuth(props) {  
  const {dispatch, authState} = useContext(AuthDispatcher);  
  
  const logout = () => dispatch({type: "LOGOUT"});  
  const login = () => dispatch({type: "LOGIN", ...});  
  
  return {  
    logout, login, authState  
  }  
}
```

# ARCHITEKTUR IDEE: CUSTOM HOOKS

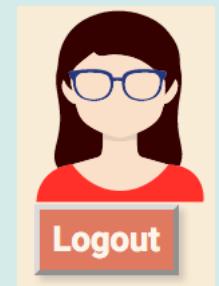
3

## CustomHooks: Für globalen App State?

- In Komponenten wird der Custom Hook verwendet, um Zugriff auf "action creator" zu erhalten

### Beispiel: Verwendung Custom Hook (statt dispatch)

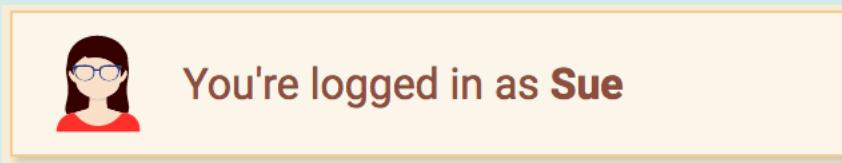
```
function LogoutButton(props) {  
  const { logout } = useAuth();  
  
  return (  
    <button onClick={logout}>  
      Logout  
    </button>  
  );  
}
```



# ARCHITEKTUR IDEE: CUSTOM HOOKS

## CustomHooks: Für globalen App State?

- In Komponenten wird der Custom Hook verwendet, um Zugriff auf den State zu bekommen



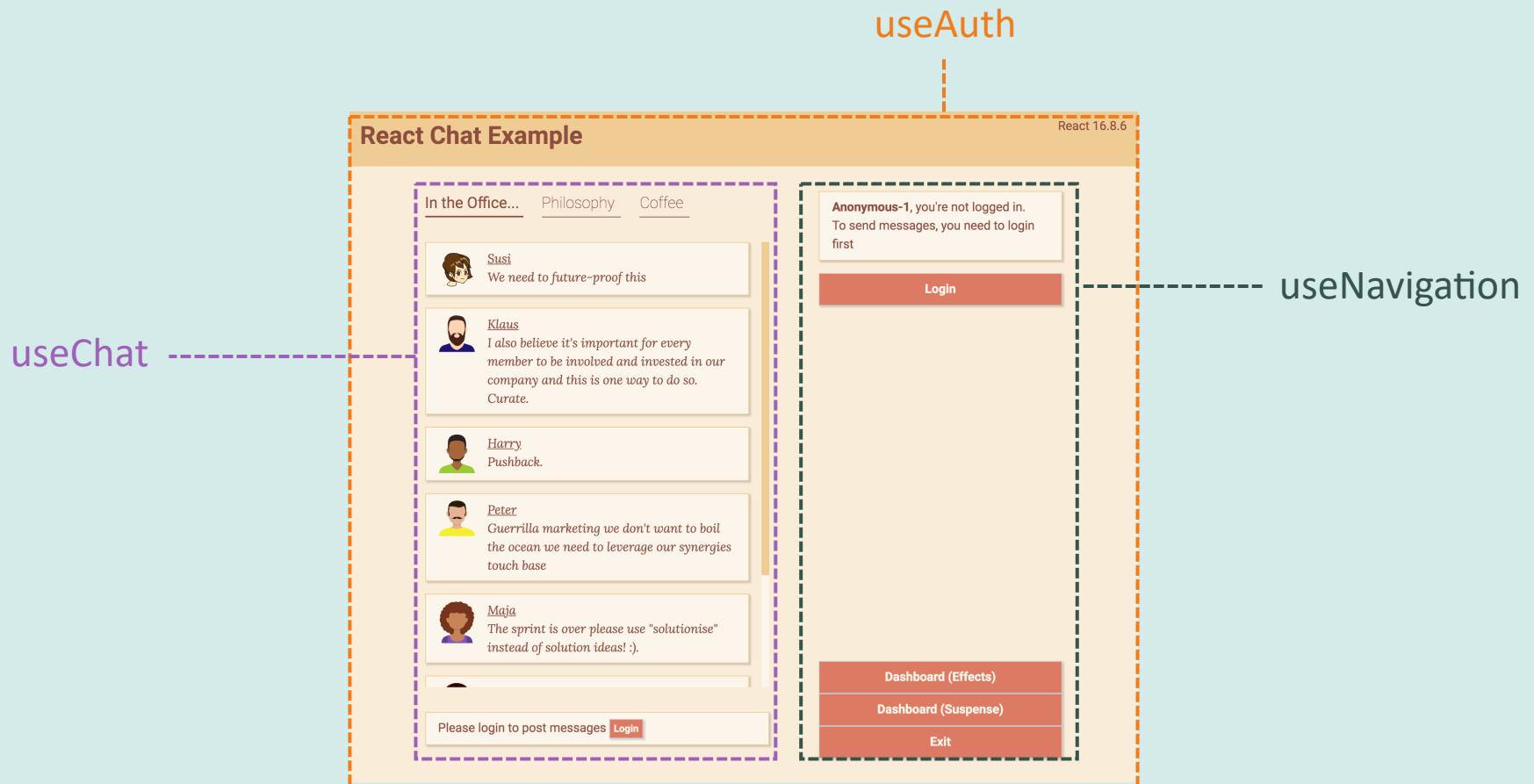
### Beispiel: Verwendung Custom Hook (statt state)

```
function Avatar(props) {  
  const { authState } = useAuth();  
  
  return (  
    <h1>You're logged in as {authState.user}</h1>  
  );  
}
```

# ARCHITEKTUR IDEE: CUSTOM HOOKS

## App-State per Context: Flexibilität

- Hooks könnten auch für "Teil-State" der Anwendung verwendet werden (im Gegensatz zu Redux):



## ARBEITEN MIT SEITENEFFEKTEN

**Server-Zugriffe, Subscriptions etc sind Seiteneffekte**

## ARBEITEN MIT SEITENEFFEKTEN

### Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  componentDidUpdate(prevProps) {  
    if (prevProps.apiKey !== this.props.apiKey) {  
      ChatApi.subscribe(this.props.apiKey);  
    }  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

Nur ausführen, wenn Properties sich geändert haben

# ARBEITEN MIT SEITENEFFEKTEN

## useEffect: Seiteneffekte in Funktionskomponenten

```
function ChatPage(props) {  
  React.useEffect(  
    () => { ----- Ersetzt componentDidMount & componentDidUpdate  
            const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
  
      },  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## useEffect: Seiteneffekte in Funktionskomponenten

Aufräumen in Rückgabe-Funktion

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    |  
    Ersetzt componentWillMount  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

## ARBEITEN MIT SEITENEFFEKTEN

### useEffect: Seiteneffekte in Funktionskomponenten

Bedingte Ausführung (Mit welchem State soll der Effekt synchronisiert werden?)

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    [props.apiKey] ----- Ersetzt Property-Vergleich in componentDidUpdate  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook", zum Laden von Daten

```
function useApi(path, initData) {
```

Beispiel eventuell komplett raus, weil sehr speziell,  
und Custom Hook haben wir schon gezeigt

```
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook", zum Laden von Daten
- Alle Hooks können in Custom Hooks verwendet werden

```
function useApi(path, initialValue) {  
  const [data, setData] = React.useState(initialValue);
```

```
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook", zum Laden von Daten
- Alle Hooks können in Custom Hooks verwendet werden

```
function useApi(path, initialValue) {
  const [data, setData] = React.useState(initialValue);

  React.useEffect(() => {
    async function readData() {
      const response = await fetch(`http://localhost:9000/${path}`);
      const data = await response.json();
      setData(data);
    }

    readData();
  }, [path]);

  return data;
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook"

```
function useApi(path, initialValue) { ... }

// Verwendung
function Dashboard(props) {
  const logs = useApi("/logs", []);
  const users = useApi("/users", []);

  return <>
    <LogViewer logs={logs} />
    <UsersViewer users={users} />
  </>;
}
```

# REACT HOOKS

## Bibliotheken mit Hooks API

- **Redux** (useSelector, useDispatch, useStore)
- **Router** (useHistory, useParams, useLocation)
- **Apollo Client** (useQuery, useMutation)
- **React Intl** (useIntl)
- **React i18n** (useTranslation)

# HOOKS

- Müssen wir jetzt alle Hooks verwenden? 😱
- Was ist mit unseren Klassen? 😱

# HOOKS

- **Müssen wir jetzt alle Hooks verwenden?** 😱
- **Was ist mit unseren Klassen?** 😱
- **Zunächst:**
  - Hooks sind "opt-in"
  - Hooks sind abwärtskompatibel
  - Eingeführt in Minor-Version (!)

# HOOKS

- Müssen wir jetzt alle Hooks verwenden? 😱
- Was ist mit unseren Klassen? 😱
- ...also: keine Panik! React bleibt stabil! 😊

Finally, there is no rush to migrate to Hooks. We recommend avoiding any “big rewrites”, especially for existing, complex class components. It takes a bit of a mindshift to start “thinking in Hooks”. In our experience, it’s best to practice using Hooks in new and non-critical components first, and ensure that everybody on your team feels comfortable with them. After you give Hooks a try, please feel free to send us feedback, positive or negative.

We intend for Hooks to cover all existing use cases for classes, but **we will keep supporting class components for the foreseeable future**. At Facebook, we have tens of thousands of components written as classes, and we have absolutely no plans to rewrite them. Instead, we are starting to use Hooks in the new code side by side with classes.

16.6

# Suspense

RENDERN UNTERBRECHEN

## SUSPENSE

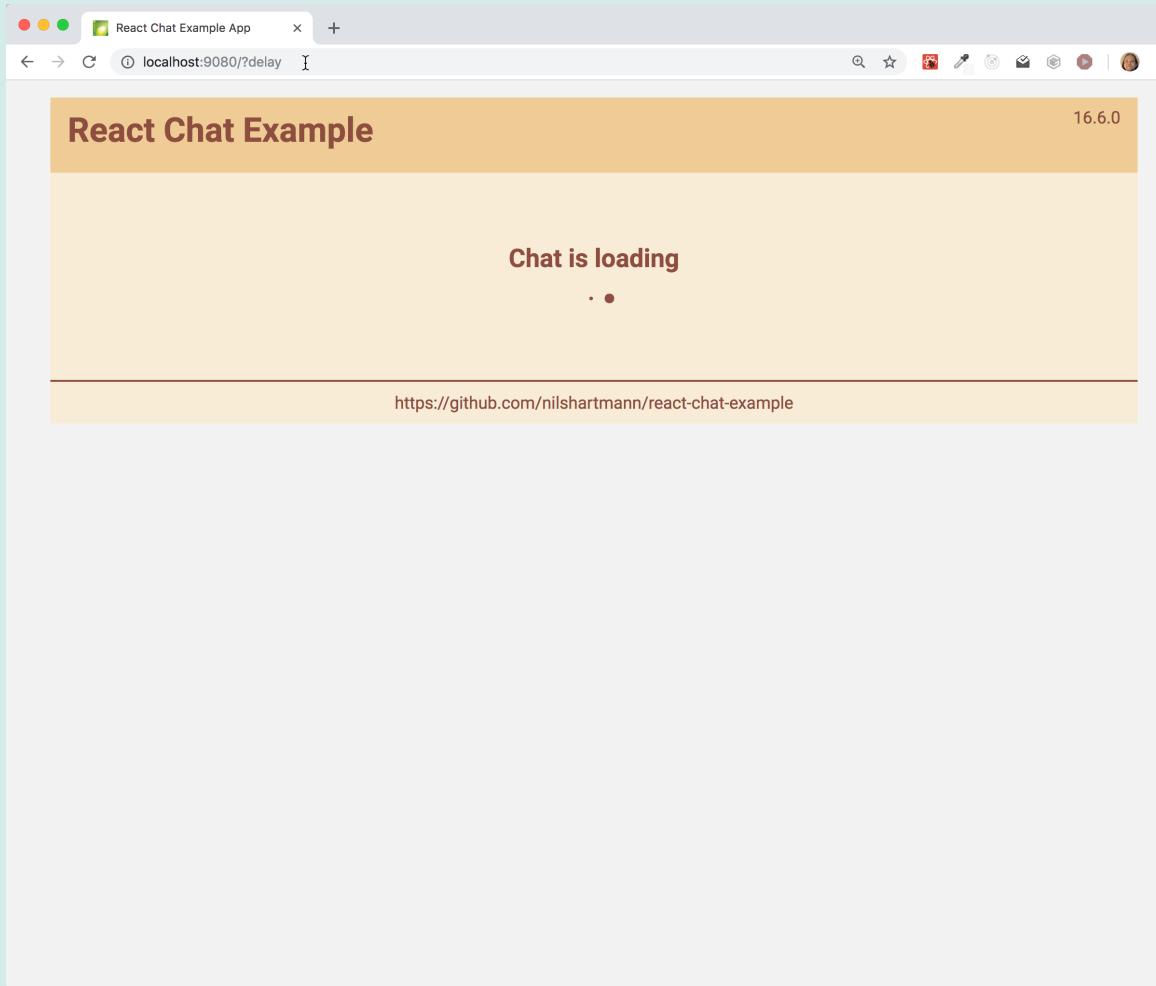
**Suspense:** React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden

- Funktioniert aktuell für **Code Splitting**
- In Planung auch zum **Laden von beliebigen Daten**

# DEMO: LAZY UND SUSPENSE

- **Demo: Fallback Komponente**

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



# DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**  
<http://localhost:9081/?delay>

The screenshot shows a web browser window with the URL [localhost:9081/chat?delay](http://localhost:9081/chat?delay). The main content is a "React Chat Example" application. It displays two messages:

- A message from user "LYAM/JM": "The sprint is over please use "solutionise" instead of solution ideas! )."
- A message from user "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."

To the right of the messages is a login form:

- Input fields for "Email" and "Password".
- A "Login" button.
- A "Dashboard" button.
- An "Exit" button.

Below the login form, a link <https://github.com/nilshartmann/react-chat-example> is visible.

At the bottom of the browser window, the developer tools Network tab is open, showing a list of network requests. A purple oval highlights the first four requests in the list:

Name	Status	Remote Address	Type	Initiator	Size	Time	Waterfall
main.js	200	127.0.0.1:9080	script	?delay	1.7 MB	71 ms	
backend.js	200	127.0.0.1:9080	script	VM548:7	(from di...	11 ms	
distChatPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	18.6 KB	2 ms	
distDashboardPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	10.2 KB	3 ms	

At the bottom of the developer tools, the status bar shows: 4 / 27 requests | 1.7 MB / 1.8 MB transferred | Finish: 23.76 s | DOMContentLoaded: 229 ms | Load: 228 ms

# SUSPENSE

## React.lazy: Code splitting with Suspense [16.6]

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));  
                                         Dynamic Import  
  
class App {  
  render() {  
    return <>  
  
      <ChatPage />  
      // more pages...  
  
    </>  
  }  
}
```

# SUSPENSE

**React.Suspense:** Zeigt Fallback Komponente an [16.6]

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

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));

class App {
  render() {
    return <>
      <React.Suspense fallback={<h1>Loading...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

## An Update to the Roadmap

**tldr:** We shipped Hooks on time, but we're regrouping Concurrent Mode and Suspense for Data Fetching into a single release that we intend to release later this year.

# Concurrent Mode & Suspense for Data Fetching

AUSBLICK

### Time Slicing: Bessere Nutzung von CPU Zeiten

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

### Time Slicing: Bessere Nutzung von CPU Zeiten

- Rendern ist eine "non-blocking" Operation
  - Es kann **immer** auf User-Interaktionen reagiert werden
- Komponenten können vor-gerendert werden, ohne sofort sichtbar zu sein
  - Ohne Nachteile für sichtbare Komponenten (Performance)

### Time Slicing: Bessere Nutzung von CPU Zeiten

- Rendern ist eine "non-blocking" Operation
  - Es kann **immer** auf User-Interaktionen reagiert werden
- Komponenten können vor-gerendert werden, ohne sofort sichtbar zu sein
  - Ohne Nachteile für sichtbare Komponenten (Performance)

### Suspense: Besseres Umgehen mit IO

- Einheitliche API für das Arbeiten mit asynchronen Daten
- Pausieren des Renders von **einem Teil** der Komponenten



# vielen Dank!

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

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

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