

**Neues Jahr, alles neu?**

# React 2019

Slides: <https://bit.ly/oose-react-2019>

# **NILS HARTMANN**

## **Freiberuflicher Programmierer und Trainer aus Hamburg**

**JavaScript, TypeScript, React  
Java  
Trainings, Workshops**



**nils@nilshartmann.net**

**[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)**

# NILS HARTMANN



gaearon commented 5 days ago

Member

+ 😊 ...

[REDACTED]

[REDACTED]

@nilshartmann

[REDACTED]

I do think you're a bit confused.

[REDACTED]

[REDACTED]



2

**DISCLAIMER: "A BIT CONFUSED"**

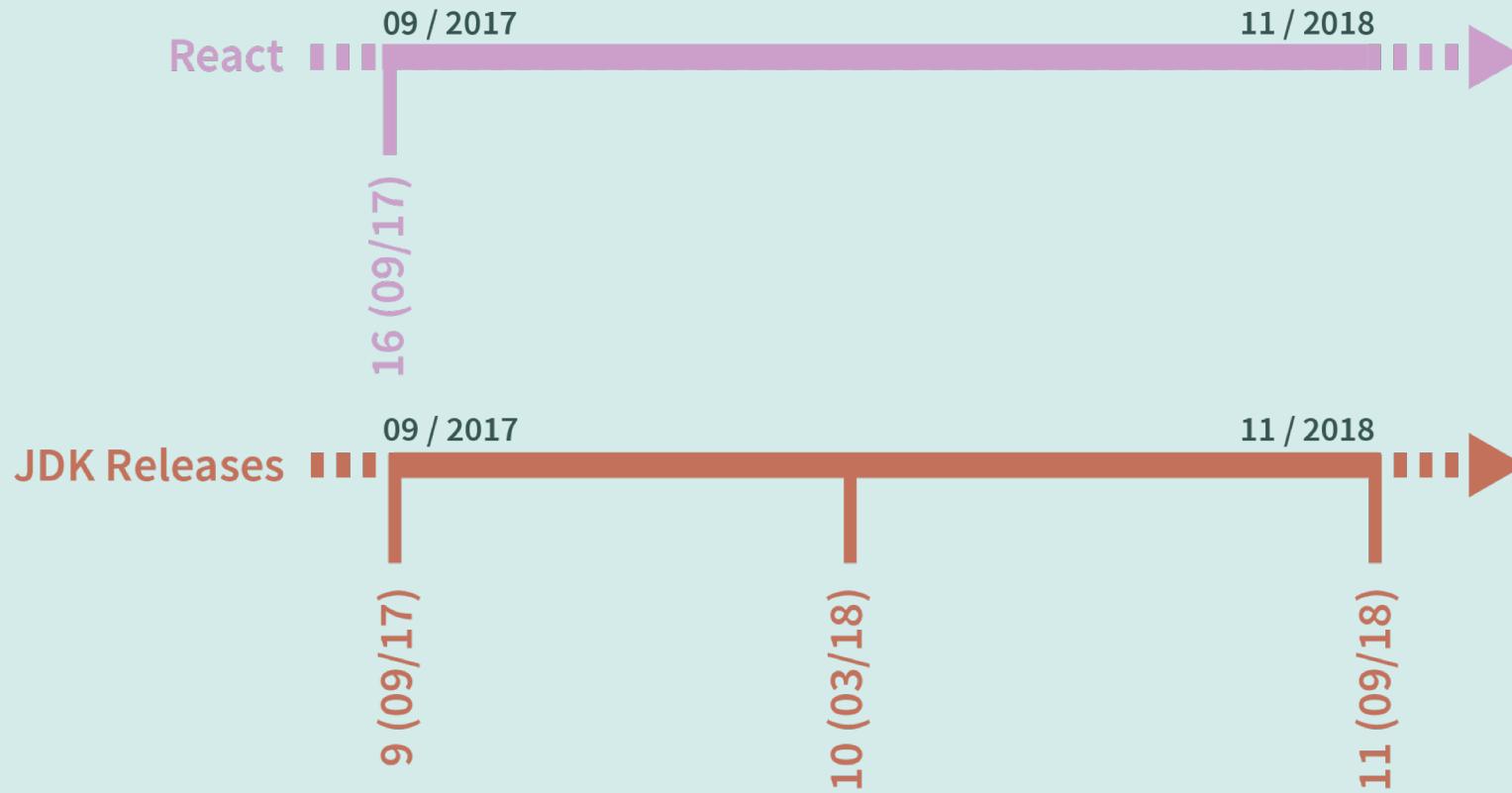
**React 16**  
**Erschienen September 2017**

# Blick zurück

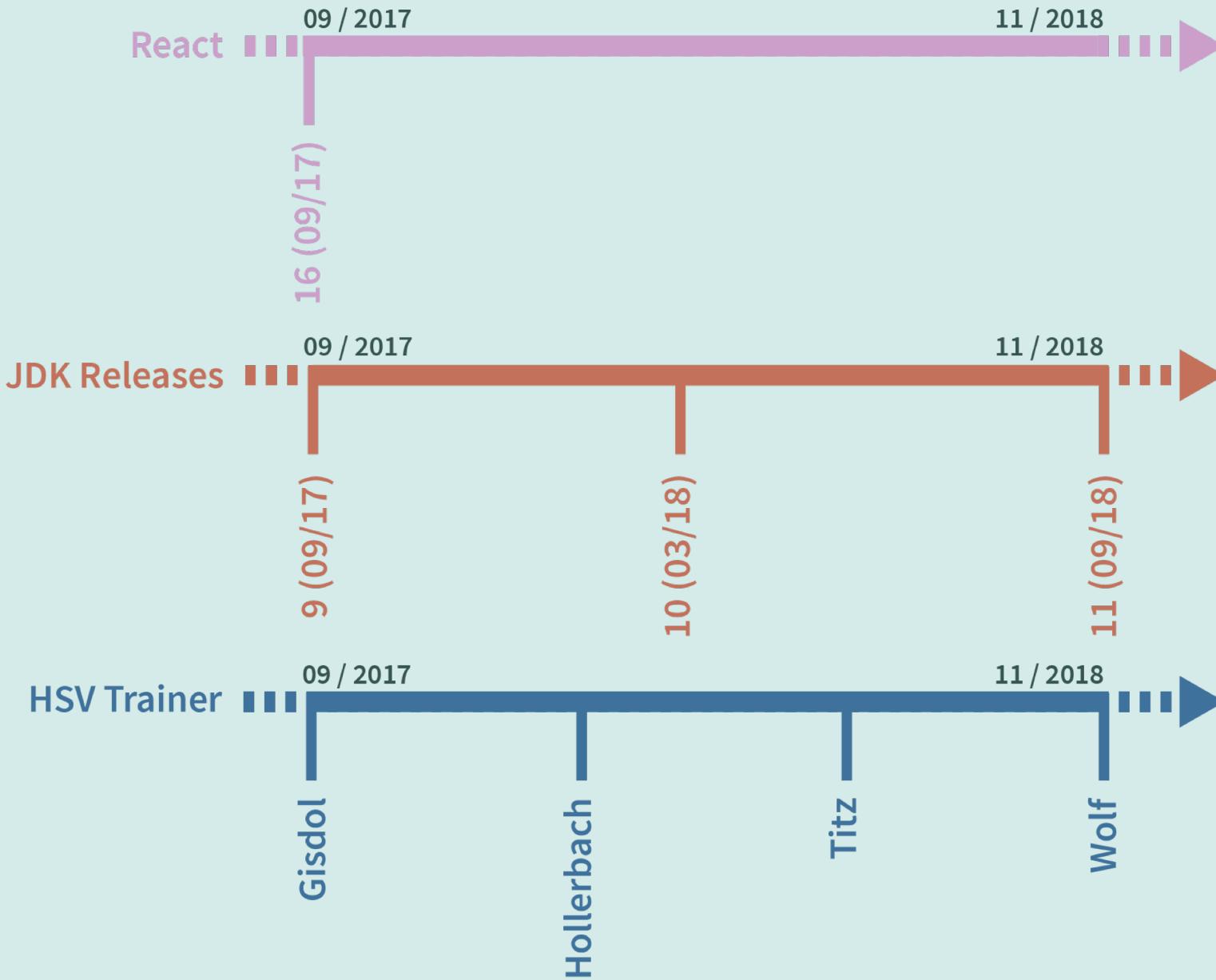
# REACT 2018... ZUM VERGLEICH...



## REACT 2018... ZUM VERGLEICH...



# REACT 2018... ZUM VERGLEICH...



React 16.x

**Blick voraus**

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 (!)

The image displays two side-by-side screenshots of a web application titled "React Chat Example".

**Left Screenshot (React 16.6.0):**

- Header:** "React Chat Example" and "16.6.0".
- Top Bar:** Buttons for "In the Office...", "Philosophy", and "Coffee".
- Message List:**
  - Anonymous-620:** "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."
  - 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-620:** "joined in the Office..."
  - Anonymous-621:** "joined in the Office..."
  - Anonymous-617:** "left in the Office..."
  - Klaus:** "Anonymous-621 logged in as Klaus"
- Login Area:** A box containing the message "Anonymous-620, you're not logged in. To send messages, you need to login first" with a "Login" button.
- Footer:** "Please login to post messages" and a "Login" button.

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

- Header:** "React Chat Example" and "React 16.7.0-alpha.0".
- Top Bar:** Buttons for "In the Office...", "Philosophy", and "Coffee".
- Message List:** The same messages as the left screenshot, including the "Klaus" message.
- User Profile:** A box on the right showing "You're logged in as Klaus" with a profile picture.
- Input Field:** "Add Message" with a text input field and a "Send" button.
- Footer:** Three red buttons labeled "Dashboard (Effects)", "Dashboard (Suspense)", and "Exit".

**URLs:**

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

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

EIN BEISPIEL...

**16.8**

# Hooks

\* <https://github.com/facebook/react/pull/14692/files>

**FUNCTIONS EVERYWHERE**

## HINTERGRUND

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

### Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
  - Durch Concurrent Rendering noch problematischer

## HINTERGRUND

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

### Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
  - Durch Concurrent Rendering noch problematischer

**Hooks sind reguläre Funktionen**

## HINTERGRUND

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

**Motivation:**

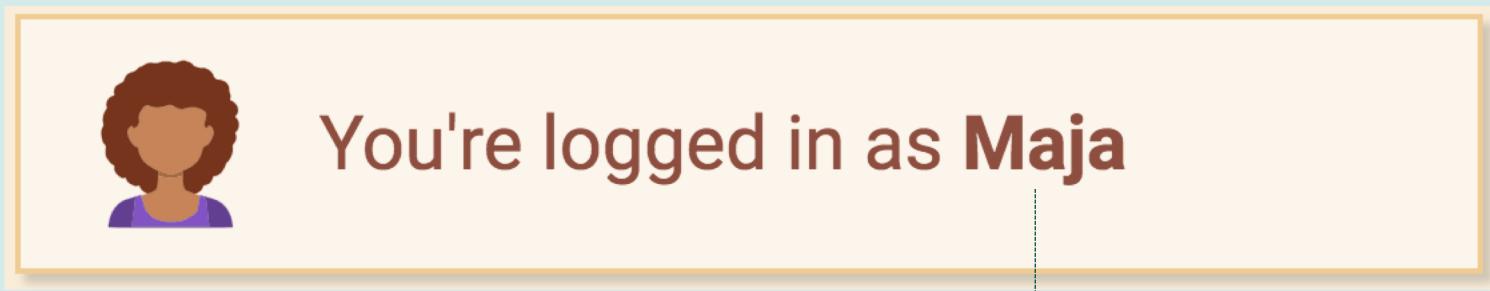
- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
  - Durch Concurrent Rendering noch problematischer

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

- **müssen** mit "use" beginnen
- **müssen** am Anfang einer Komponente stehen
- nur in Funktionskomponenten

## USECONTEXT HOOK

**useContext**: Vereinfachter Zugriff auf den Context



User-Objekt liegt in einem  
Context

## 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";

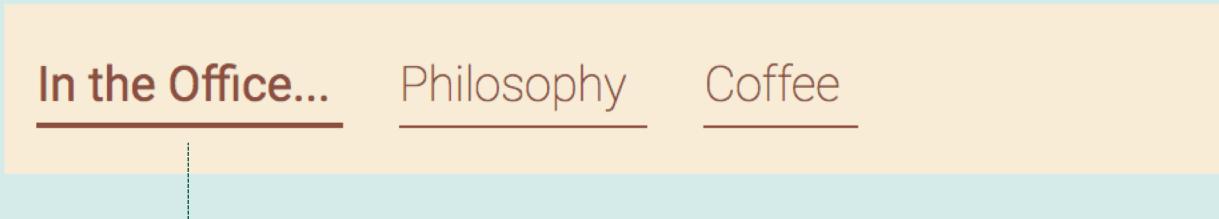
function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

  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

### Vorher: Setzen von State in Funktionen nicht möglich

- Setzen und Lesen nicht einheitlich (this.setState vs this.state.x)
- setState "seltsame" Semantik
- this-Problematik

```
class Tabs extends React.Component {  
  state = { activeTabId: 0 }  
  render() {  
    return (  
      <div>  
        {props.tabs.map(tab => {  
          return <Tab  
            classname={tab.id === activeTabId ? "active" : ""}  
            onClick={() => this.setState({activeTabId: tab.id})}  
          />  
        })}  
      </div>);  
  }  
}
```

# USESTATE HOOK

## useState: State erzeugen

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

## 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 oldState;  
  }  
}
```

# 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

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

- Reducer für globalen App State

# USERREDUCER HOOK

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

- Reducer für globalen App State

**Beispiel: globaler Reducer für Session-Informationen**

```
function authenticationReducer(state, action) {  
  switch (action.type) {  
    case "LOGIN_SUCCESS":  
      return { sessionToken: action.sessionToken }  
    case "LOGIN_FAILED":  
      return { sessionToken: null, error: action.error }  
    case "LOGOUT":  
      return { sessionToken: null, error: null }  
    default:  
      return null;  
  }  
}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

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

```
function ChatApp(props) {  
  const [state, dispatch] = useReducer(authenticationReducer);
```

```
}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

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

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);

}

}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

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

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcher.Provider value={dispatch}>
      </AuthDispatcher.Provider>
    );
}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

**Beispiel: Provider-Komponente für dispatcher**

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

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcher.Provider value={dispatch}>
      {state.sessionToken ? props.children : "please login"}
    </AuthDispatcher.Provider>
  );
}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht
- Unterkomponenten können dispatch-Funktion verwenden

```
function LogoutButton(props) {  
  const dispatch = useContext(AuthDispatcher);  
  
}  
}
```

## USERREDUCER HOOK

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

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht
- Unterkomponenten können dispatch-Funktion verwenden

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

## 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

```
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: Handler für Input-Felder

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) {  
  const [value, setValue] = React.useState(initialValue);  
  
  function onEnterHandler(e) {  
    const keyCode = e.which || e.keyCode;  
    if (keyCode === 13) {  
      onEnter(value);  
    }  
  }  
  
  return {  
    value,  
    onChange: e => setValue(e.target.value),  
    onKeyPress: onEnterHandler  
  };  
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) { ... }
```

```
// Verwendung:
```

```
function LoginDialog(props) {
  const usernameInput = useFormInput("", ChatApi.login);

  return <form>
    <input {...usernameInput} />
  </form>
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) { ... }
```

// Verwendung:

```
function LoginDialog(props) {
  const usernameInput = useFormInput("", ChatApi.login);
  const passwordInput = useFormInput("", ChatApi.login);

  return <form>
    <input {...usernameInput} />
    <input {...passwordInput} />
  </form>
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook"
- Alle Hooks können verwendet werden

```
function useApi(path, initData) {  
  const [data, setData] = React.useState(initialData);  
  
  React.useEffect(async () => {  
    const response = await fetch(`http://localhost:9000/${path}`);  
    const data = await response.json();  
  
    setData(data);  
  }, [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} />
  </>;
}
```

# 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

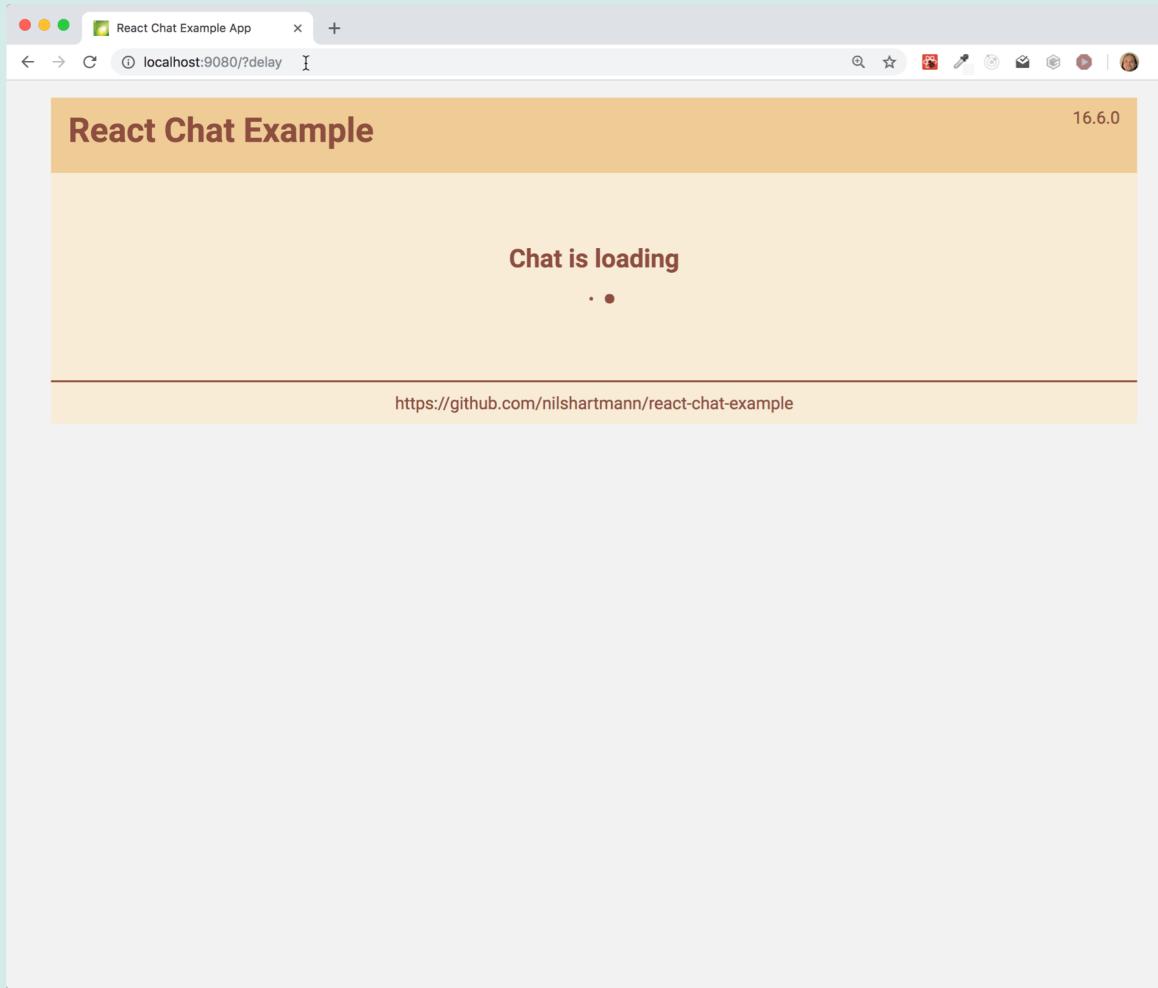
**Suspense:** React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden [16.6]

- Funktioniert aktuell (nur) für Code Splitting

# DEMO: LAZY UND SUSPENSE

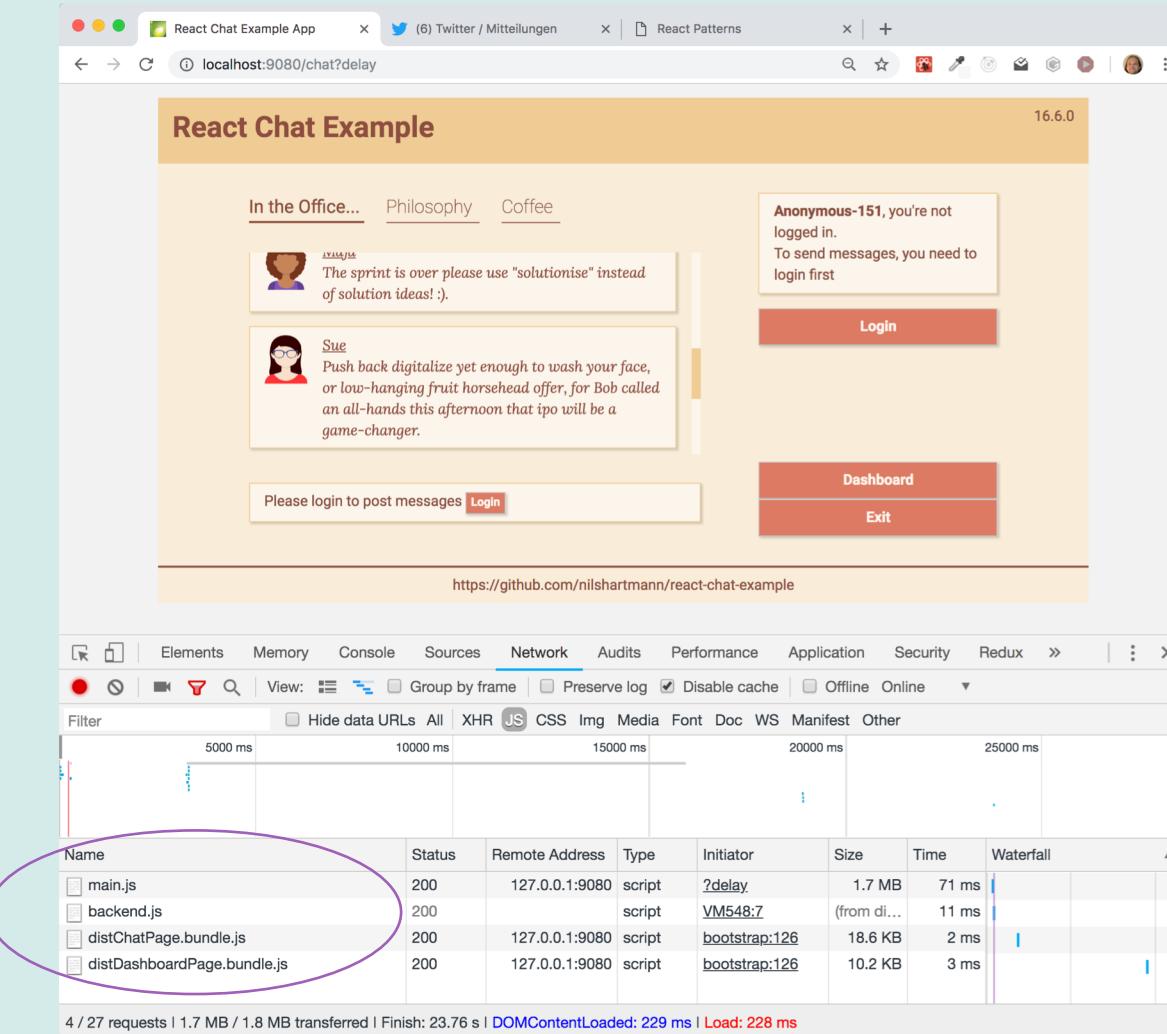
- **Demo: Fallback Komponente**

<http://localhost:9080/?delay>



# DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**  
<http://localhost:9080/?delay>



React Chat Example 16.6.0

In the Office... Philosophy Coffee

AYAMAJM 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.

Please login to post messages [Login](#)

Anonymous-151, you're not logged in. To send messages, you need to login first

[Login](#)

[Dashboard](#)

[Exit](#)

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

Network

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	

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"));  
class App {  
  render() {  
    return <>  
      <ChatPage />  
      // more pages...  
    </>  
  }  
}
```

**Dynamic Import**

# 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>
    <>
  }
}
```

## DEMO: LAZY UND SUSPENSE

### Problem: "Flickern"

- Entsteht, wenn Ladezeiten sehr schnell sind
- Loading indicator wird für wenige Millisekunden angezeigt und verwirrt eher als es nützt

**16.x-alpha** unstable!

# Concurrent React

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)

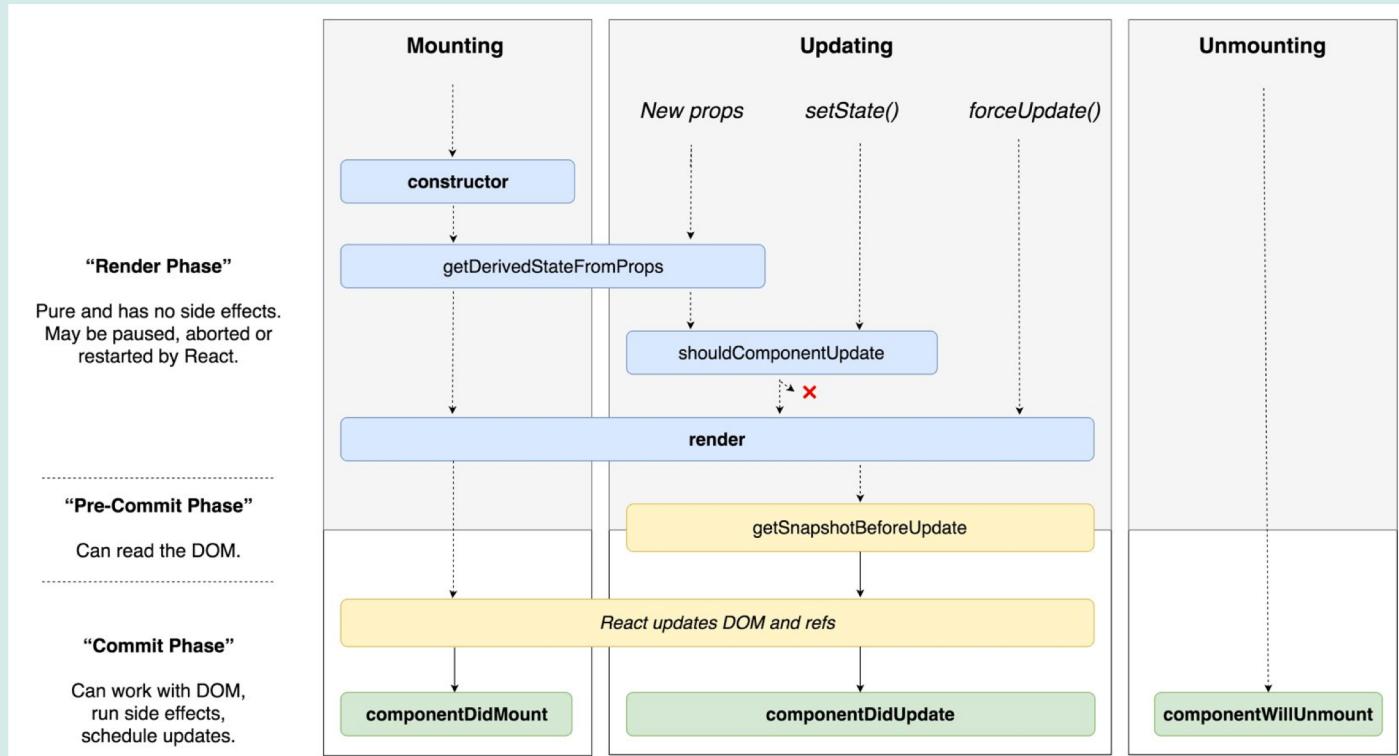
### Suspense: Besseres Umgehen mit IO

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

# ASYNCHRONES RENDERN

## Unterscheidung in Render- und Commit-Phase

- Render Phase ist "pure", darf keine Nebeneffekte haben
- Deswegen neue Lifecycle-Methoden



[https://twitter.com/dan\\_abramov/status/981712092611989509](https://twitter.com/dan_abramov/status/981712092611989509)

# CONCURRENT MODE

## Concurrent Mode [16.7]

- Concurrent Mode muss explizit eingeschaltet werden
- Geht auf jeder Ebene in der Anwendung
  - Sehr gut für Migration, falls es Probleme gibt

```
ReactDOM.createRoot(getDocumentById("..."))
  .render(
    <React.StrictMode>
      <React.ConcurrentMode>
        <ErrorHandler>
          <App />
        </ErrorHandler>
      </React.ConcurrentMode>
    </React.StrictMode>
  );
}
```

## SUSPENSE MIT CONCURRENT MODE

**Suspense:** Flickern verhindern mit Concurrent Mode

- `maxDuration` legt eine Zeit fest, bis `fallback` gerendert wird
- Bis dahin wird bestehende Komponente angezeigt
- Vor React 16.7 nicht / nur schwer möglich

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

class App {
  render() {
    return <>
      <React.Suspense maxDuration={100} fallback={<h1>...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

## LAZY UND SUSPENSE

- **Demo: Fallback Komponente mit maxDuration**

<http://localhost:9081>

## SUSPENSE

**Ausblick [16.x ~mid 2019]:** Suspense for Data Fetching

- *Alle gezeigten Beispiele verwenden unstable API!!*

# BEISPIEL: DATEN LADEN MIT SUSPENSE

- REST Aufrufe mit fetch

/api/logs

## Admin Dashboard

Close

### Logs

```
[Anonymous-361] client disconnected  
[Anonymous-365] Assigned User id 'Anonymous-365'  
[Anonymous-365] Client registered  
[Anonymous-365] join chatroom with id 'r1'  
[Anonymous-366] Assigned User id 'Anonymous-366'  
[Anonymous-366] Client registered  
[Anonymous-366] join chatroom with id 'r1'  
[Anonymous-364] client disconnected  
[Anonymous-367] Assigned User id 'Anonymous-367'  
[Anonymous-367] Client registered
```

/api/users

### User

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

# ASYNCHRONES DATEN LADEN

- **"Klassisches" Daten laden**

- In componentDidMount Daten das Laden anstoßen
- In der Zwischenzeit Loading Indicator anzeigen
- (Mit Hooks andere API, aber gleiches Konzept)

```
class LogsView extends React.Component {  
  state = {};  
  
  async componentDidMount() {  
    const response = await fetch("/api/logs");  
    const logs = await response.json();  
    this.setState({ logs })  
  }  
  
  render() {  
    if (!this.state.logs) { return <h1>Loading...</h1> }  
    return <div> // render logs </div>;  
  }  
}
```

## DATEN LADEN MIT SUSPENSE - 1

- **Daten laden mit Suspense**

- Beim Rendern wird eine Funktion aufgerufen die Daten liefert – oder auch nicht, dann wird Rendern **pausiert**
- Sobald die Funktion (später) Daten liefert, wird die Komponente gerendert

```
function LogsView() {  
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück  
  
  return <div> ...geladene logs hier anzeigen... </div>;  
}
```

## DATEN LADEN MIT SUSPENSE - 2

- **Daten laden mit Suspense**

- Beim Rendern wird eine Funktion aufgerufen die Daten liefert – oder auch nicht, dann wird Rendern **pausiert**
- Sobald die Funktion (später) Daten liefert, wird die Komponente gerendert
- Komponente wird irgendwo im Tree mit **Suspense** umschlossen

```
function LogsView() {  
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück  
  
  return <div> ...geladene Logs hier anzeigen... </div>;  
}  
  
function DashboardPage() {  
  return <Suspense maxDuration={...} fallback={...}>  
    <LogsView />  
  </Suspense>  
}
```

## DATEN LADEN MIT SUSPENSE - 3

- **react-cache** (zzt 2.0.0-alpha): Funktioniert mit React 16.x-alpha NICHT
  - Geladene Daten (**Resourcen**) können gecached werden
  - Wenn Daten noch nicht vorhanden, werden sie vom Server gelesen

```
import { unstable_createResource } from "react-cache";

// Liefert Promise zurück
async function loadLogsFromApi() {

  const response = await fetch("http://localhost:9000/api/logs");
  return await response.json();

}

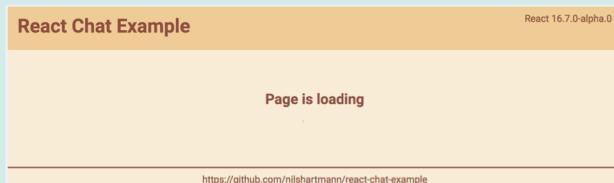
const LogsResource = unstable_createResource(loadLogsFromApi);
```

# DATEN LADEN MIT SUSPENSE

- Demo: Suspense an diversen Stellen

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

(anpassen in DashboardPageWithSuspense.js)



## HINTERGRUND: SUSPENSE

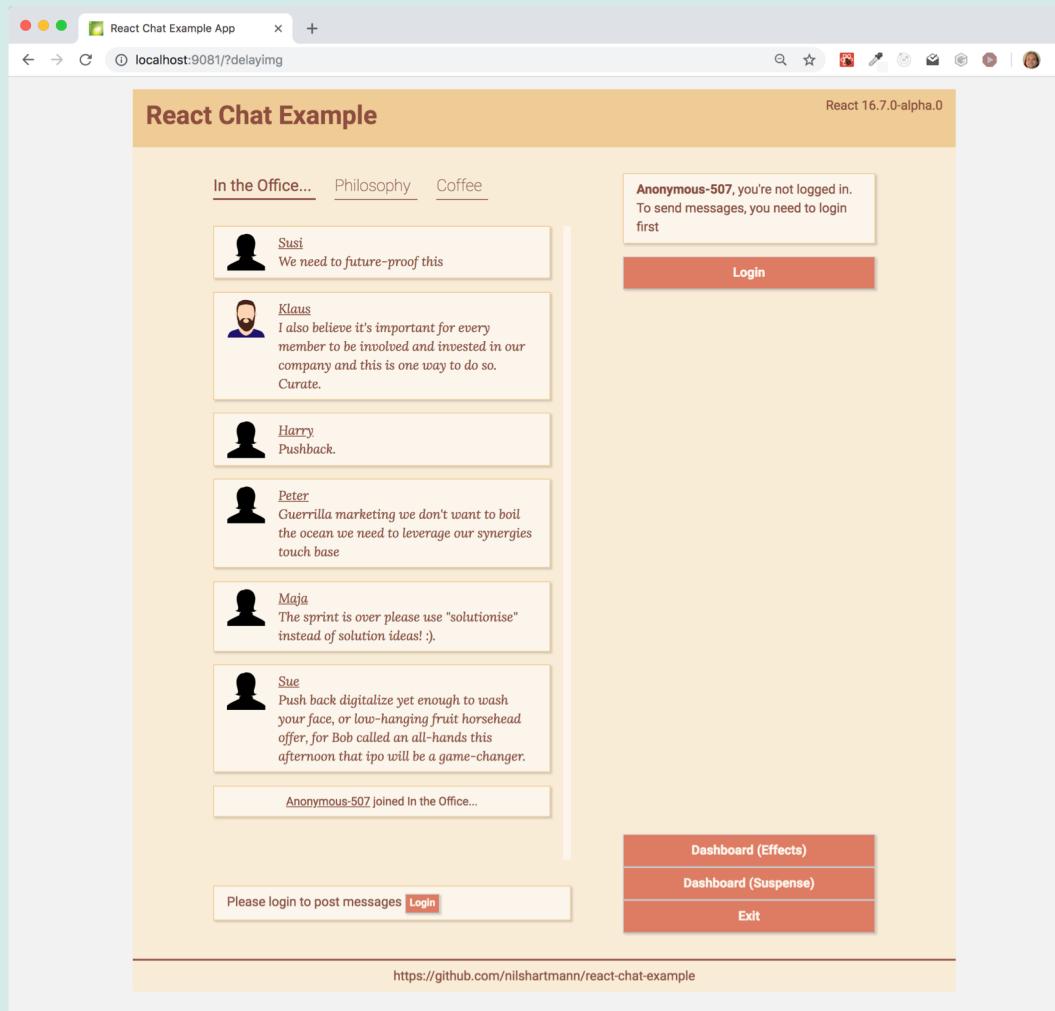
- Wie funktioniert das eigentlich?

```
function LogsView(props) {  
  const logs = LogsResource.read();  
  
  // ⏵ wird nur ausgeführt, wenn logs zurückgeliefert wird: 🤔🤔  
  return <> ... Logs hier anzeigen ... </>;  
}
```

# BEISPIEL: VORSCHAUEN MIT RESPONSE

- **Suspense nutzen, um Vorschauen zu laden**

Demo: <http://localhost:9081/?delayimg>



## BEISPIEL: VORSCHAUEN MIT SUSPENSE

**Vorher – ohne Vorschau, so wie gewohnt**

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  return <img className="Avatar" src={src} />;  
}
```

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Vorher – ohne Vorschau, so wie gewohnt

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  return <img className="Avatar" src={src} />;  
}  
  
function ChatMessage(props) {  
  
  return (  
    <div className="Message">  
      <Avatar userId={message.user.id} />  
      { props.message.text}  
      ...  
    </div>  
  );  
}
```

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Avatar Komponente mit Suspense

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  ImageResource.read(src); // <-- "Wartet" auf Image  
  return <img className="Avatar" src={src} />;  
}
```

credits: @jaredpalmer

<https://github.com/jaredpalmer/react-conf-2018/blob/master/full-suspense/src/components/ArtistDetails.js>

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Image Resource

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  ImageResource.read(src); // <-- "Wartet" auf Image  
  return <img className="Avatar" src={src} />;  
}  
  
const ImageResource = unstable_createResource(  
  source =>  
    new Promise(resolve => {  
      const img = new Image();  
      img.src = source;  
      img.onload = resolve;  
    })  
)
```

"Trick", um zu warten, bis der Browser ein Image geladen hat

credits: @jaredpalmer

<https://github.com/jaredpalmer/react-conf-2018/blob/master/full-suspense/src/components/ArtistDetails.js>

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Einbinden

```
function ChatMessage(...) {  
  
  return (  
    <div className="Message">  
      <React.Suspense fallback={}>  
        <Avatar userId={message.user.id} />  
      </React.Suspense>  
  
      { props.message.text }  
      ...  
    </div>  
  );  
}
```

## AUSBLICK: SUSPENSE AUF DEM SERVER

### Suspense for Server Rendering

We started designing a new server renderer that supports Suspense (including waiting for asynchronous data on the server without double rendering) and progressively loading and hydrating page content in chunks for best user experience. You can watch an overview of its early prototype in [this talk](#). The new server renderer is going to be our major focus in 2019, but it's too early to say anything about its release schedule. Its development, as always, will happen on [GitHub](#).

<https://reactjs.org/blog/2018/11/27/react-16-roadmap.html#suspense-for-server-rendering>

# ZUSAMMENFASSUNG – SUSPENSE & CONCURRENT RENDERING

- **Ab React 16.x**

- Suspense
  - Kann das Rendern eines Teils der Hierarchie unterbrechen und später fortsetzen
  - Funktioniert heute für Lazy Loading von Komponenten
- Concurrent Mode
  - Erlaubt es React, verschiedene Render Vorgänge unterschiedlich zu priorisieren
  - ~~Kann ab React 16.7 testweise aktiviert werden~~
- Cache API
  - Neue Möglichkeit, Daten für React zu laden
  - Sieht synchron aus, blockiert aber (trotzdem) nicht

@NILSHARTMANN

# vielen Dank!

Slides: <https://bit.ly/oose-react-2019>

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

NILS@NILSHARTMANN.NET