

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

Fiber

RÜCKBLICK...

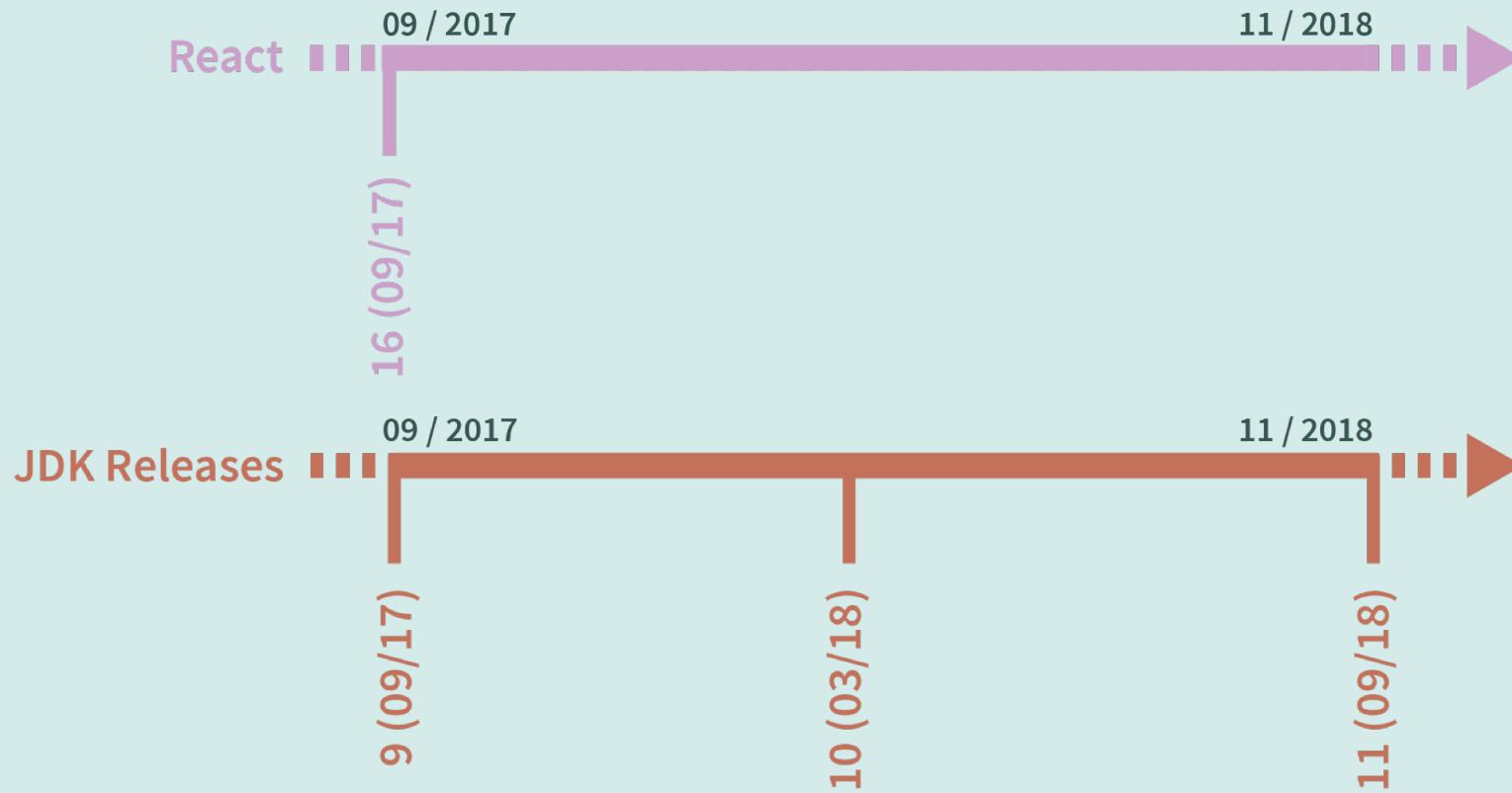
Rückblick...

- Erste 16.x Major-Version im September 2017
 - Seitdem nur Minor-Versionen
- Trotzdem sehr viele neue Features
 - neuer Rendering Modus
 - Hooks
 - Suspense
- Gut geeignet für langlaufende (Enterprise-)Anwendungen

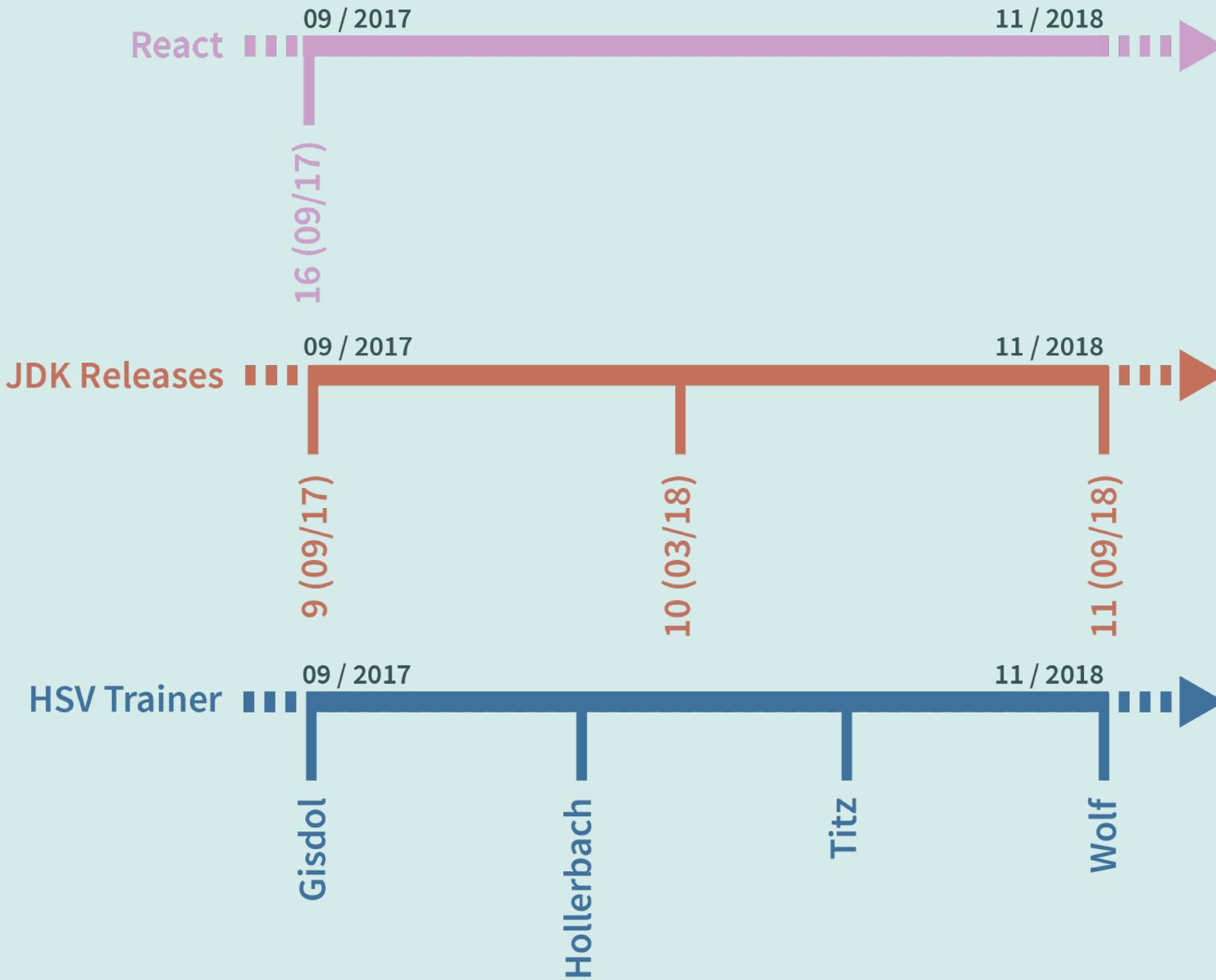
REACT 2018... ZUM VERGLEICH...



REACT 2018... ZUM VERGLEICH...



REACT 2018... ZUM VERGLEICH...



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>

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:** A series of messages from users like Harry, Peter, Maja, and Sue.
- Anonymous User Alert:** A message box for "Anonymous-620" stating they are not logged in and need to log in to send messages.
- Login Button:** A red "Login" button.
- Footer:** "Please login to post messages" and a "Login" button.
- Bottom Bar:** "Exit" button.
- Page URL:** <https://github.com/nilshartmann/react-chat-example>

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:** A series of messages from users like Harry, Peter, Maja, and Sue.
- Anonymous User Alert:** A message box for "Anonymous-620" stating they are not logged in and need to log in to send messages.
- User Profile:** A message box for "Klaus" stating "You're logged in as Klaus".
- Input Field:** "Add Message" input field and a "Send" button.
- Side Panel:** Buttons for "Dashboard (Effects)", "Dashboard (Suspense)", and "Exit".
- Bottom Bar:** "Exit" button.
- Page URL:** <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

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
- es gibt noch mehr Regeln
- eigenes es-lint Plug-in

USEDISPATCH HOOK

useContext: Vielleicht als ersten zeigen?

- Kein "mergen" von State mehr!

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

USESTATE HOOK

useState: State in Funktionskomponenten

Beispiel: Tab Bar



In the Office... Philosophy Coffee

```
function Tabs(props) {  
  return <div>...</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)} />  
  </>);  
}
```

USEDISPATCH HOOK

useDispatch: Für komplexe State-Objekte

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

USEDISPATCH HOOK

useContext & useDispatch: Für globalen App State

- Statt Properties rumreichen
- Empfänger kann Actions dispatchen

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

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

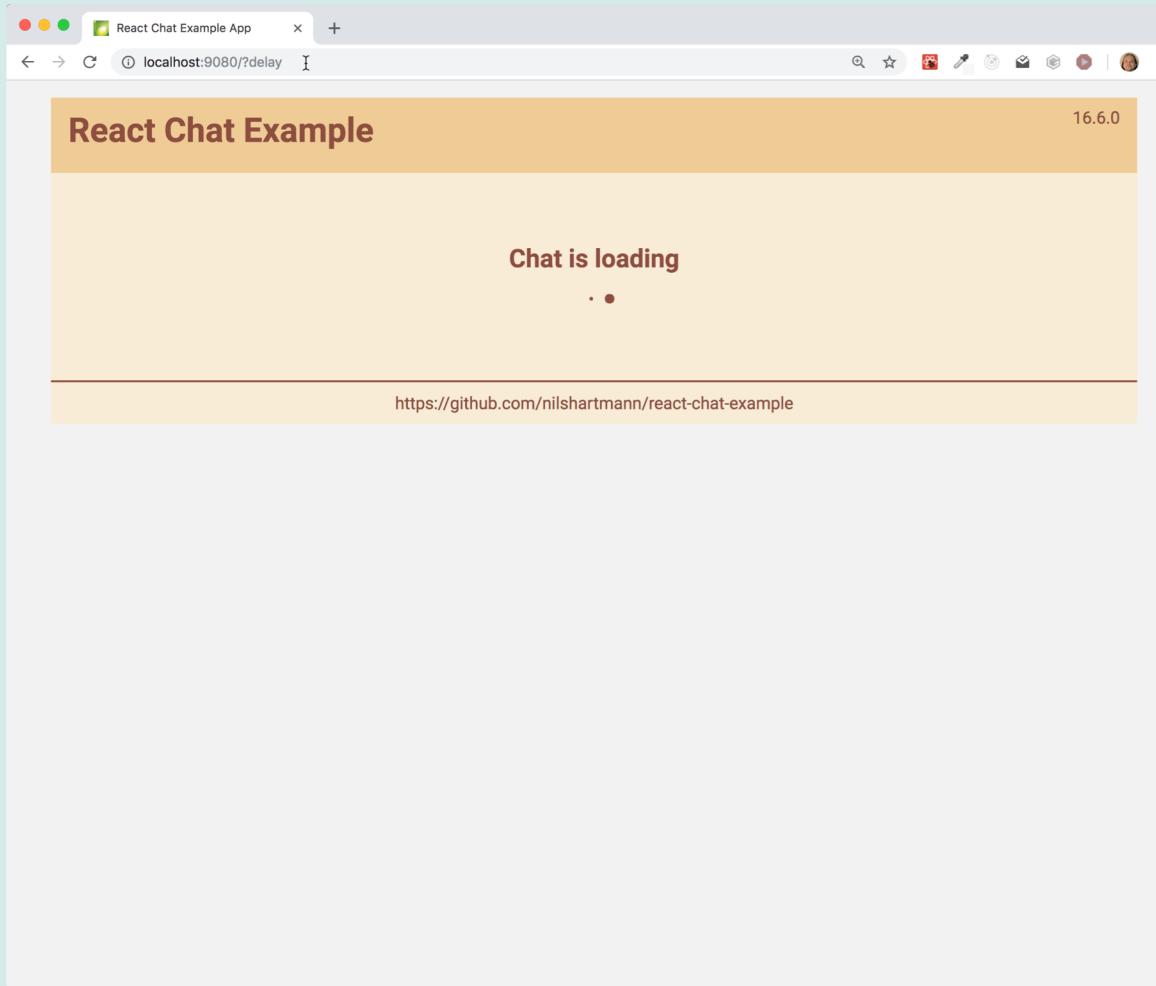
Suspense: React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden [16.6]

- Funktioniert aktuell (nur) für Code Splitting

DEMO: LAZY UND SUSPENSE

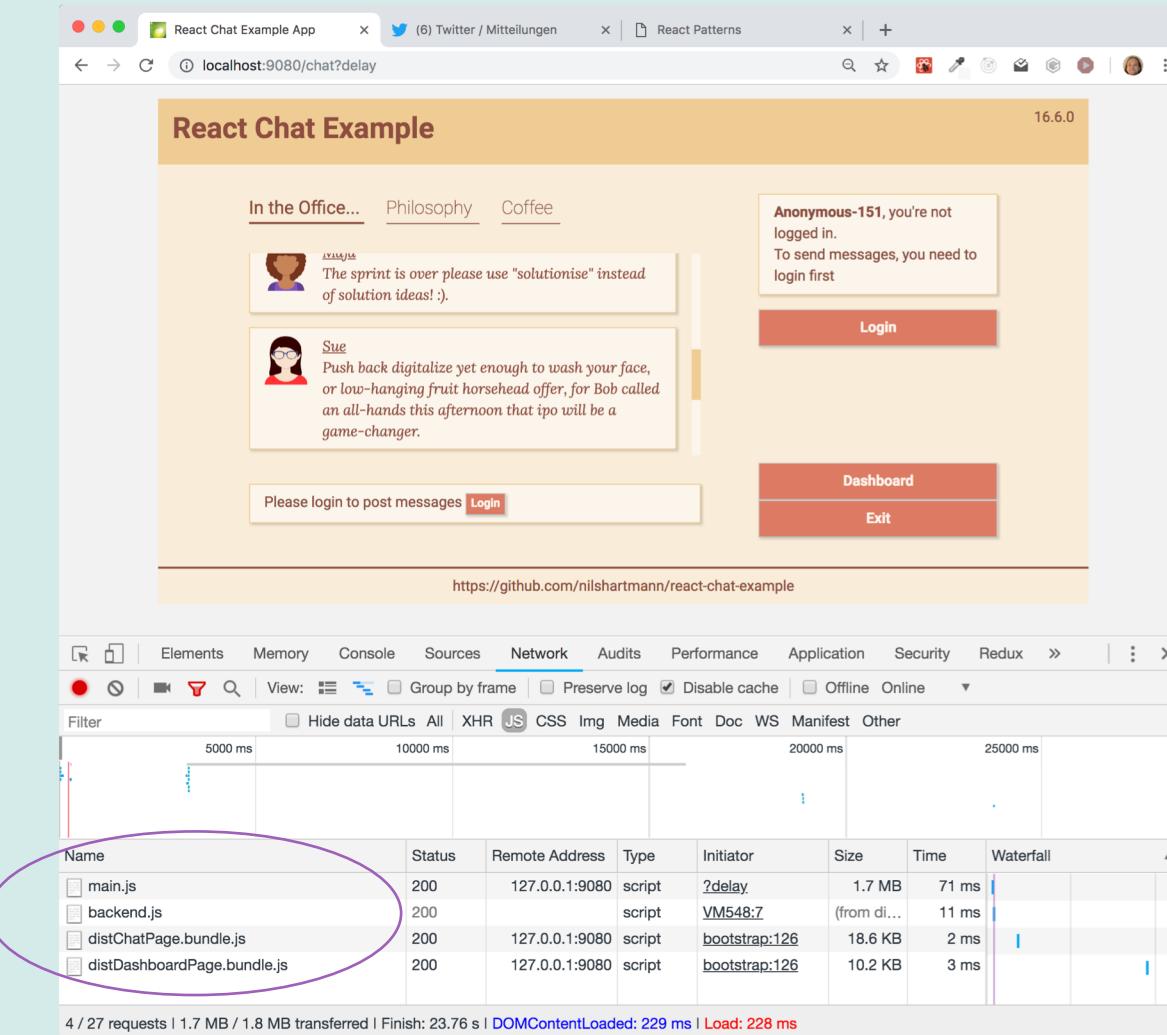
- **Demo: Fallback Komponente**

<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)

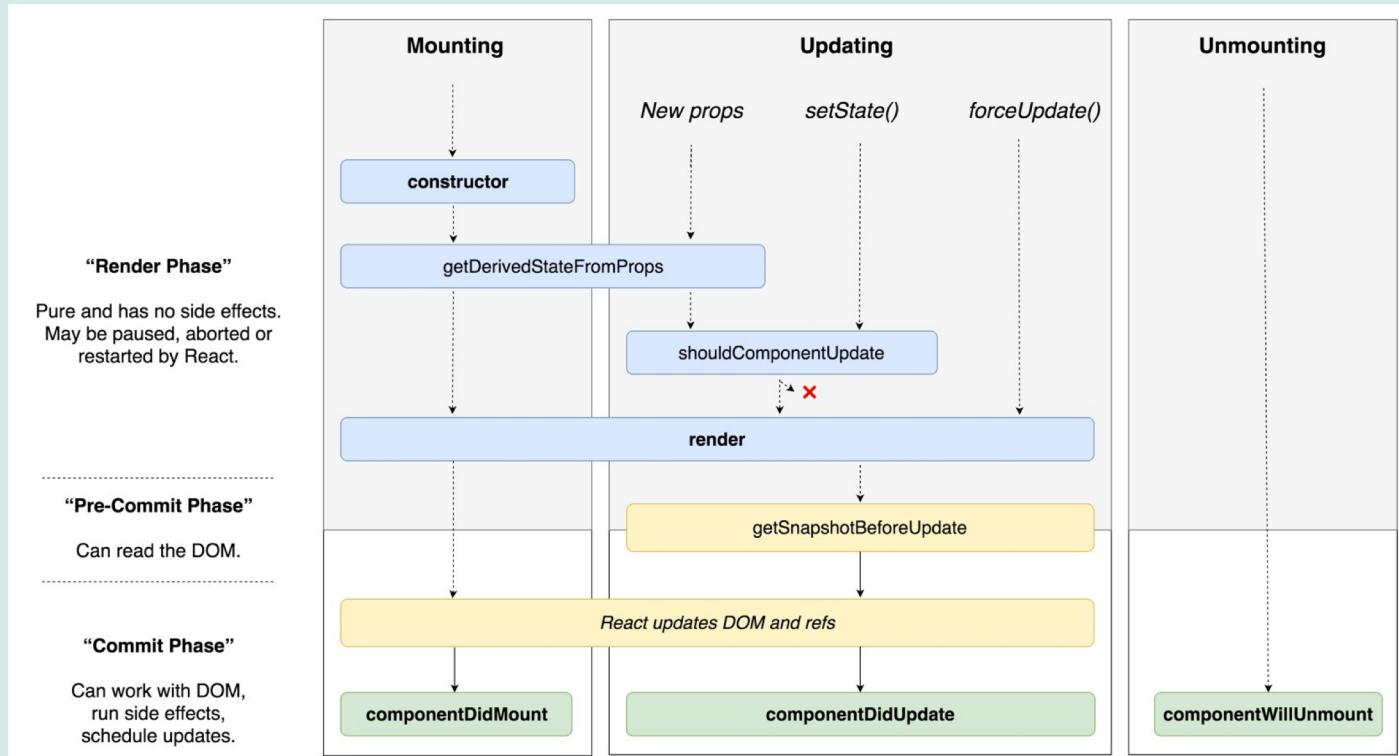
Suspense: Besseres Umgehen mit IO

- 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

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();

}

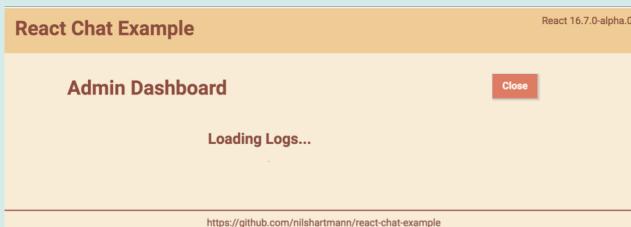
const LogsResource = unstable_createResource(loadLogsFromApi);
```

DATEN LADEN MIT SUSPENSE

- Demo: Suspense an diversen Stellen

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

(anpassen in DashboardPageWithSuspense.js)



HINTERGRUND: SUSPENSE

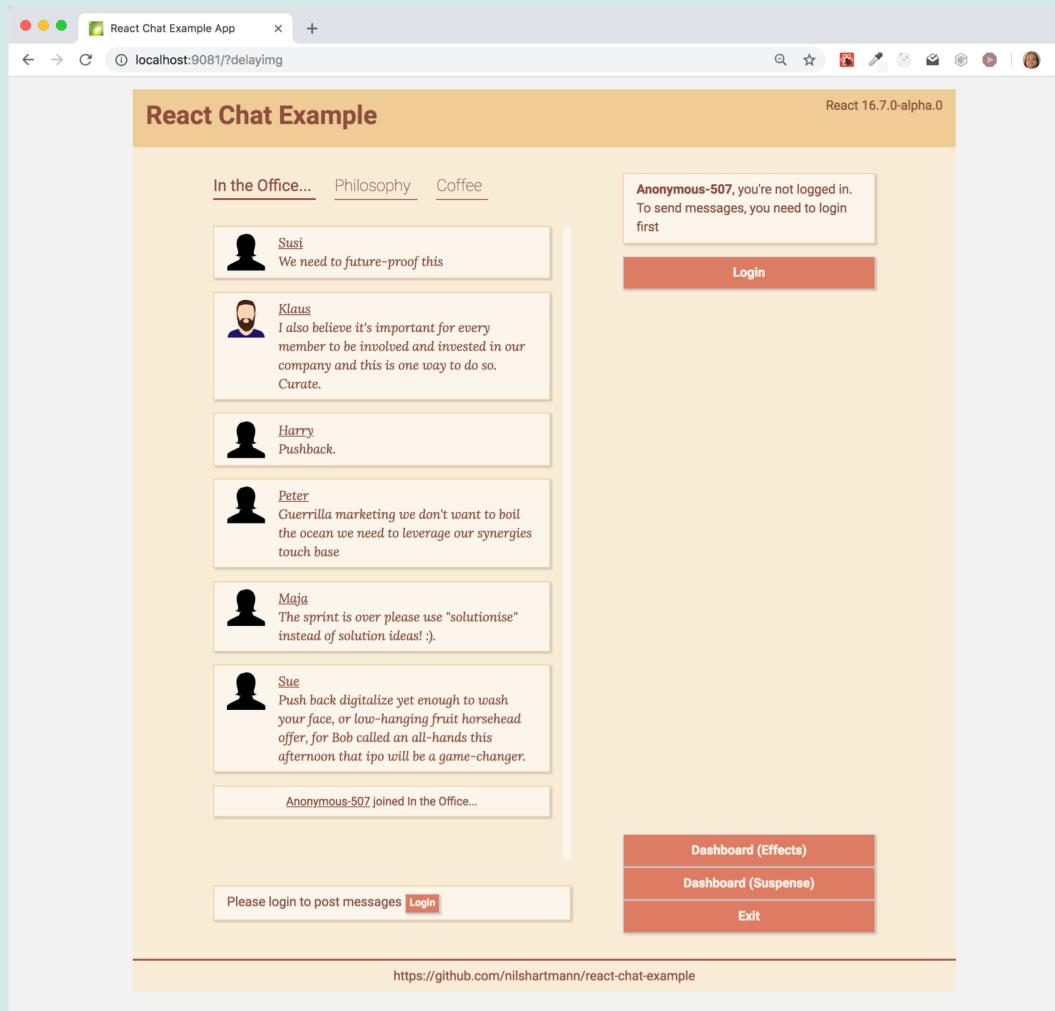
- Wie funktioniert das eigentlich?

```
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/wjax2018-react>

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

NILS@NILSHARTMANN.NET