NILS HARTMANN
https://nilshartmann.net

Hooks, Concurrent Mode, Suspense

React 2019

Alles neu?

Slides: https://nils.buzz/bedcon19-react

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings, Workshops und Beratung

HTTPS://NILSHARTMANN.NET

November 2018...

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

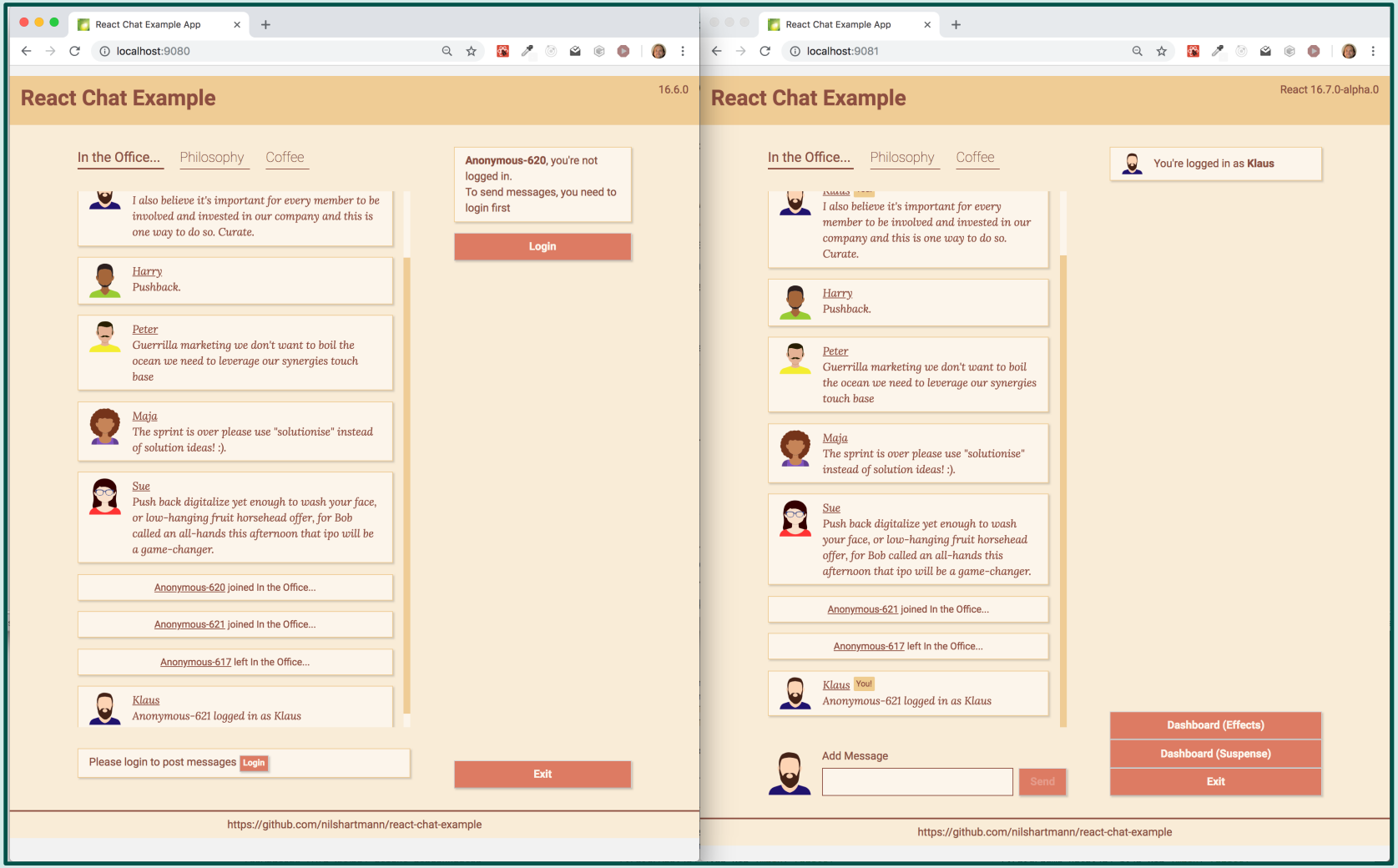
In November 2018, we have posted this roadmap for the 16.x releases:

- 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



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 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, Lifecycle etc auch in Funktionskomponenten

Motivation:

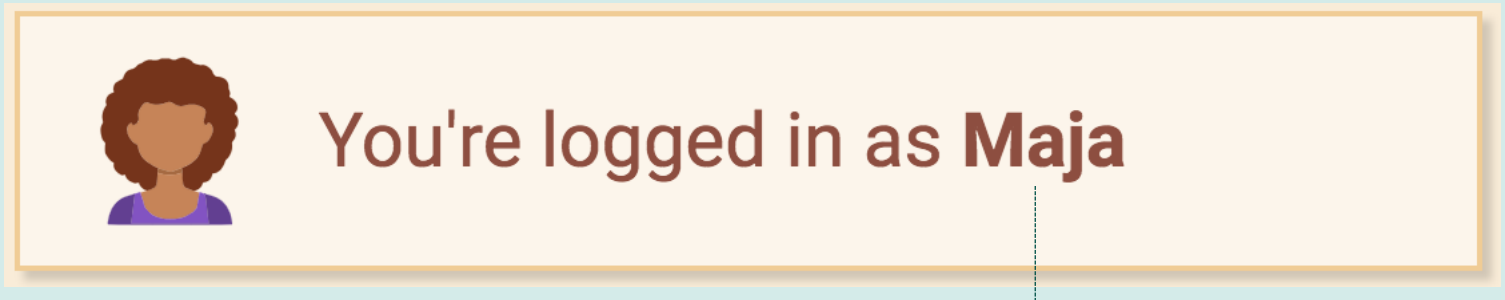
- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
  - Durch Concurrent Rendering noch problematischer

Hooks sind reguläre Funktionen, aber...

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

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

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

```
export default class Tabs extends React.Component {
  onTabChange(tabId) {
    this.setState({activeTabId: tabId})
  }
  render() {
    return <div>
        {props.tabs.map(tab => <Tab tabId={tab.id}
                    onTabClick={this.onTabChange} /> )}
    </div>;
  }
}
```

useState: State erzeugen

```
function Tabs(props) {
  const [activeTabId, setActiveTabId] = React.useState(0);

}
```

Aktueller State

Setter

Initialer Wert

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

USEREDUCER HOOK

useReducer: Redux für Komponenten?

- Für komplexen State mit viel Logik zur Veränderung

USEREDUCER HOOK

useReducer: Redux für Komponenten?

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

USEREDUCER HOOK

useReducer: Redux für Komponenten?

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

Actions sind einfache JavaScript-Objekte

```
const action = {
  type: "SET_USER",
  username: "..."
}
```

Type

Payload

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

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

USEREDUCER HOOK

useReducer: Redux für Komponenten?

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

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









  }
}
```

USEREDUCER 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 };









  }
}
```

USEREDUCER 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!");
  }
}
```

USEREDUCER HOOK

useReducer: Redux für Komponenten?

Schritt 2: Verwendung

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

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

  return (<>




  </>);
}
```

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

USEREDUCER 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"});
  </>);
}
```

ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

useContext & useReducer: Für globalen App State?

ARCHITEKTUR IDEE: USECONTEXT & USEREDUCER

useContext & useReducer: Für globalen App State?

Beispiel: Ein globaler Reducer für Session-Informationen

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

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) {




}
```

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 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?

- Unterkomponenten können den State verwenden

Beispiel: Verwendung innerhalb der Anwendung – State verwenden

```
function Avatar(props) {
  const {authState} = useContext(AuthDispatcher);

  return (
    <h1>Hello, {authState.user}</h1>
  );
}
```

ARCHITEKTUR IDEE: CUSTOM HOOKS

CustomHooks: Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen
- Damit wäre auch "fachliche" API möglich, um auf dispatch zu verzichten (ähnlich Action Creator in Redux)

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: "LOGOUT", ...});

  return {
   logout, login, authState
  }

}
```

ARCHITEKTUR IDEE: CUSTOM HOOKS

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 Logout(props) {
  const {logout} = useAuth();


  return <button onClick={logout}>Logout</button>
}
```

ARCHITEKTUR IDEE: CUSTOM HOOKS

CustomHooks: Für globalen App State?

- In Komponenten kann der State verändert werden

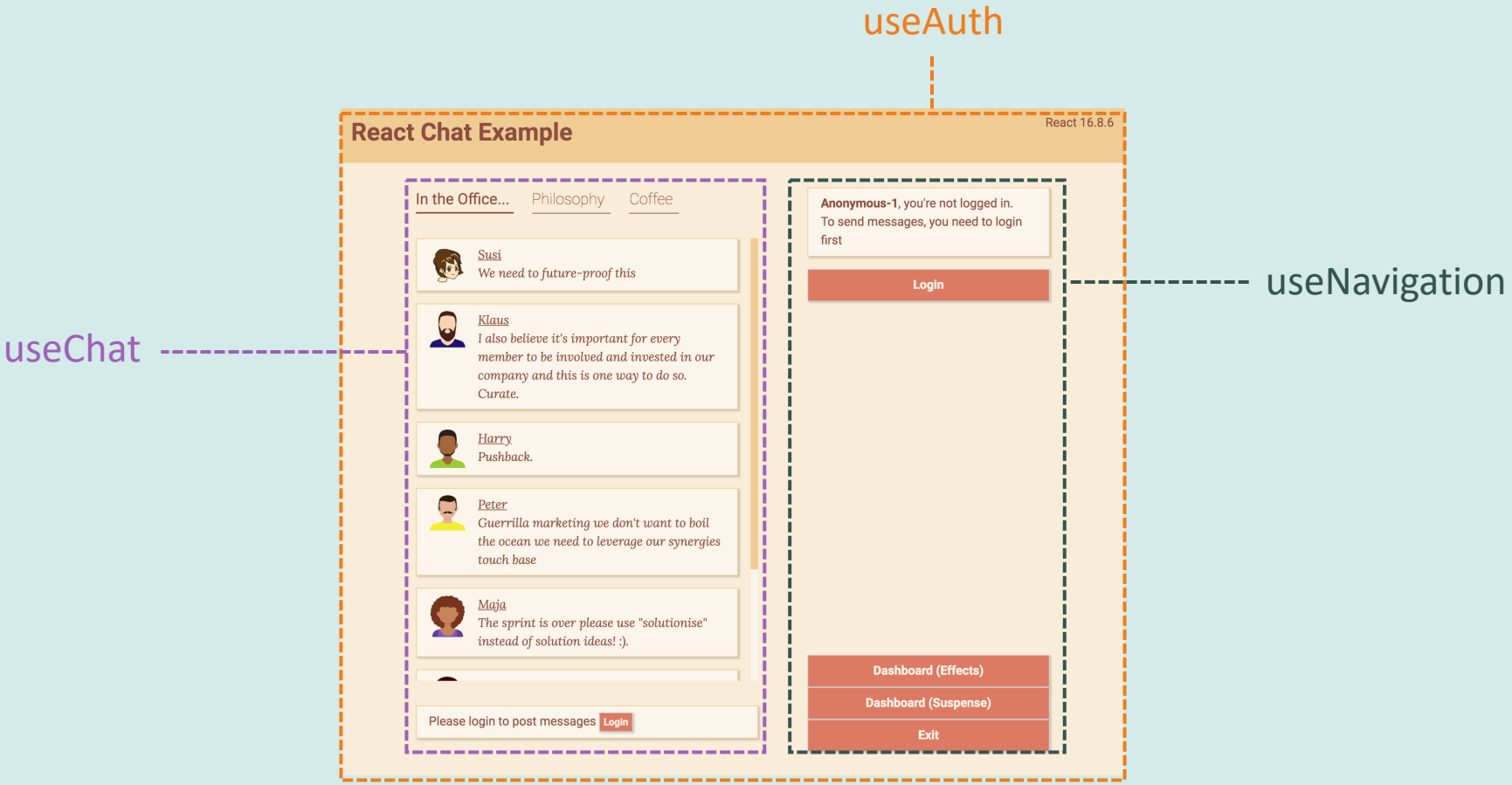
Beispiel: Verwendung Custom Hook (statt state)

```
function Avatar(props) {
  const {authState} = useAuth();

  return (
    <h1>Hello, {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

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> }
}
```

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> }
}
```

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

useEffect: Seiteneffekte in Funktionskomponenten

```
function ChatPage(props) {
  React.useEffect(
    () => {
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);


    },

  );

  return <div><h1>Chat</h1>...</div>
}
```

Ersetzt componentDidMount & componentDidUpdate

useEffect: Seiteneffekte in Funktionskomponenten

Aufräumen in Rückgabe-Funktion

```
function ChatPage(props) {
  React.useEffect(
    () => {
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);
      return () => disconnectFromApi();
    },

  );

  return <div><h1>Chat</h1>...</div>
}
```

Ersetzt componentWillUnmount

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]
  );

  return <div><h1>Chat</h1>...</div>
}
```

Ersetzt Property-Vergleich in componentDidUpdate

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: Generischer "fetch hook", zum Laden von Daten

```
function useApi(path, initialData) {

}
```

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, initialData) {
  const [data, setData] = React.useState(initialData);




}
```

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, initialData) {
  const [data, setData] = React.useState(initialData);

  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

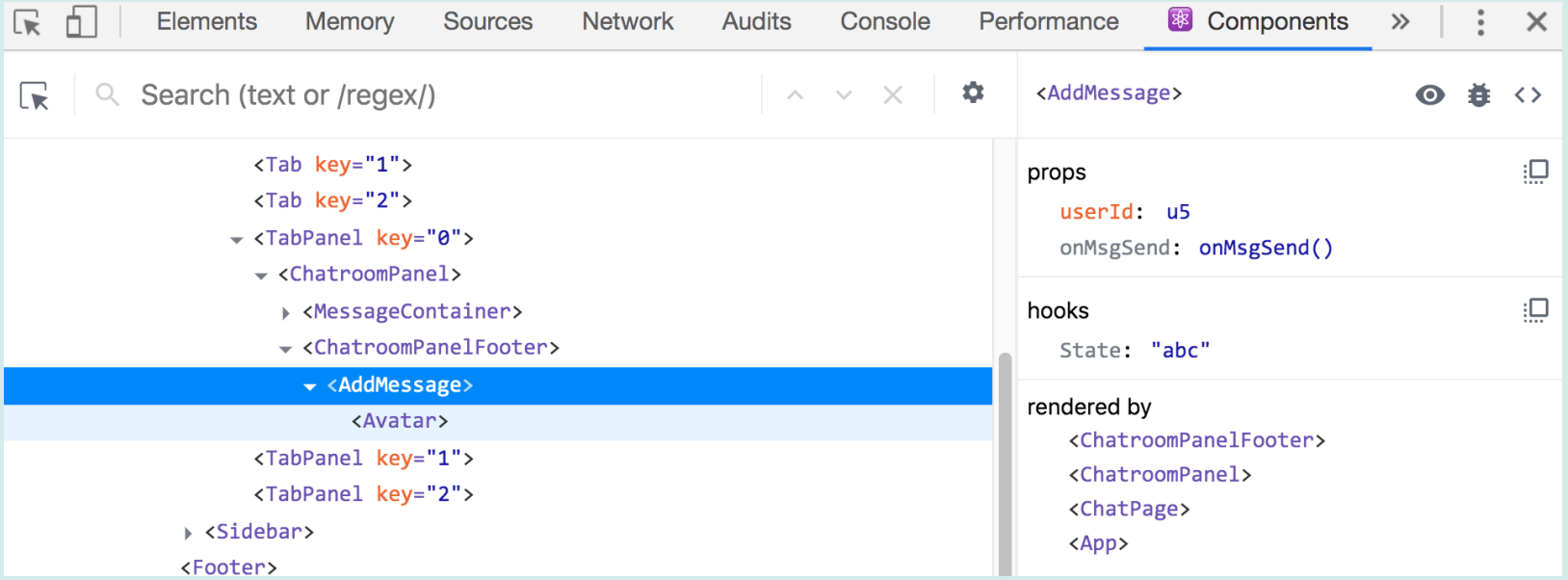
- Beispiel: Generischer "fetch hook"

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

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

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

NEUE DEVTOOLS

Neue Dev Tools unterstützen auch Hooks

- Beispiel: AddMessage



NEUE DEVTOOLS

useMemo: Kann verwendet werden, um teure Operationen zu "cachen"

- Komponentenfunktion wird bei jedem neu-rendern ausgeführt
- Funktion in useMemo aber nur, wenn sich Abhängigkeiten ändern

```
function AddMessage(props) {

  const [state, setState] = React.useState("");

  const avatar = React.useMemo(
     () => <Avatar userId={props.userId} />,
     [props.userId]
  )

  return <div>
    {avatar}
    <input value={state} onChange={...} />
  </div>;
}
```

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.

https://reactjs.org/docs/hooks-intro.html#gradual-adoption-strategy

16.6

Suspense

RENDERN UNTERBRECHEN

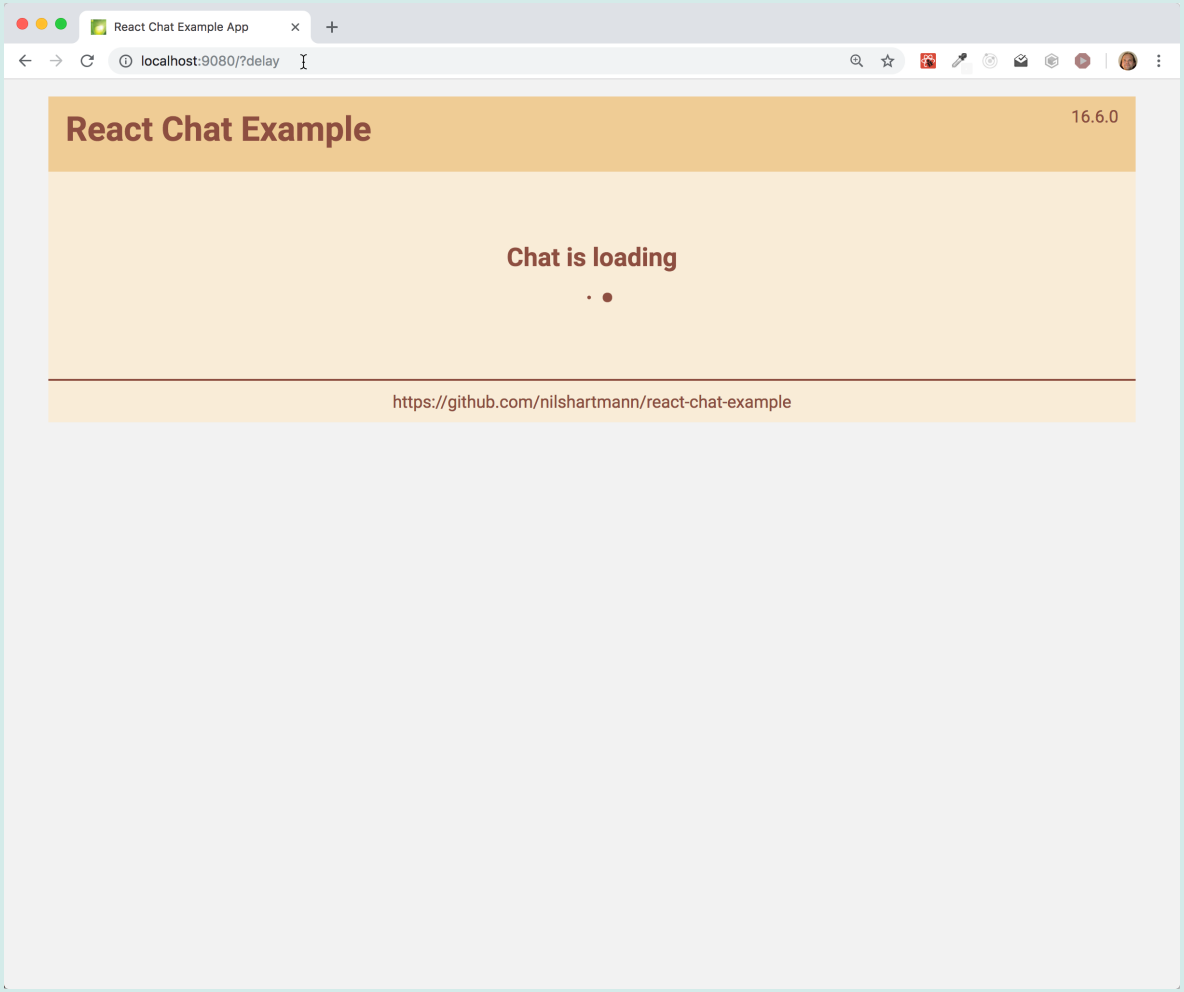
SUSPENSE

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

- Funktioniert aktuell (nur) für Code Splitting

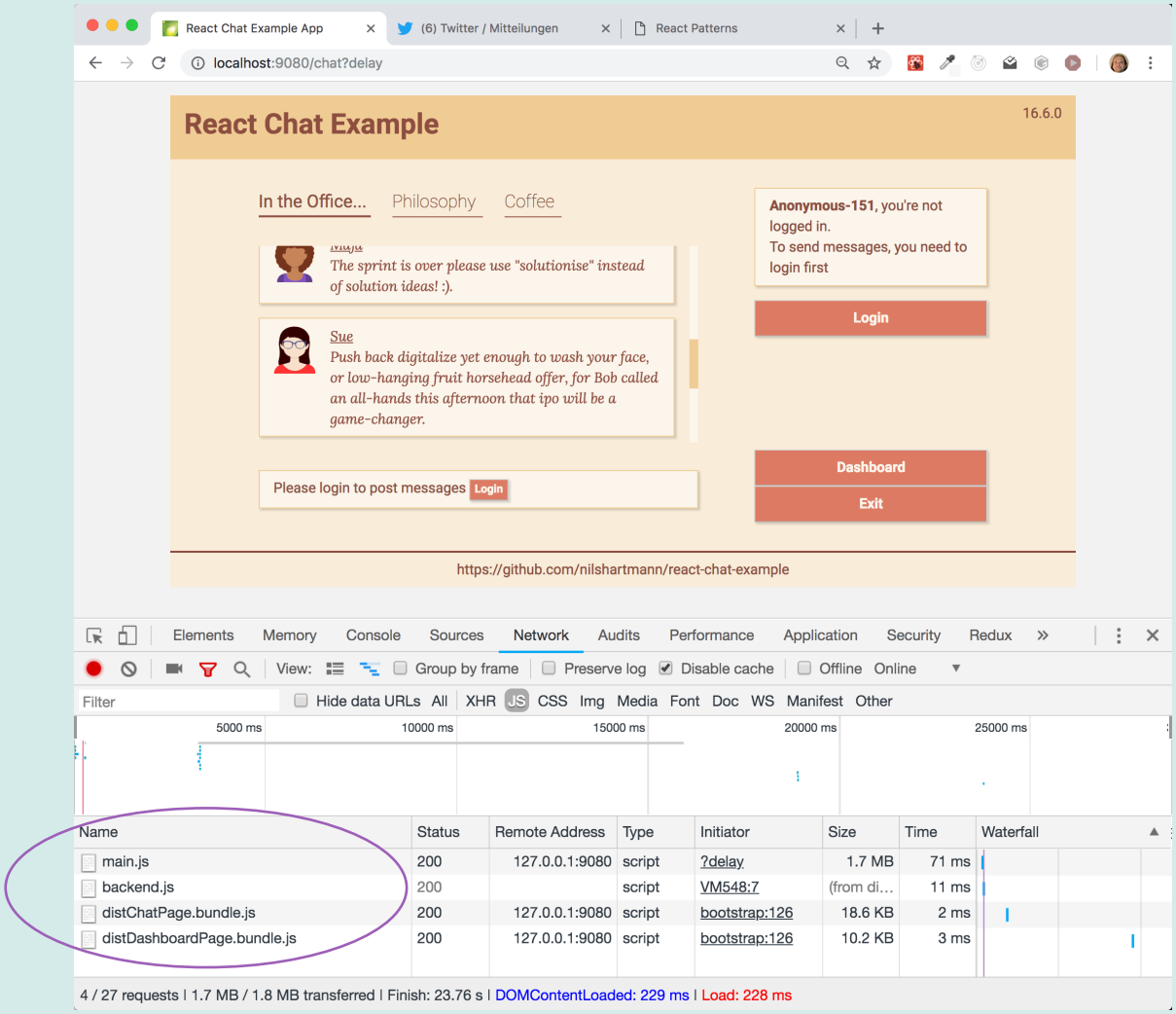
DEMO: LAZY UND SUSPENSE

- **Demo: Fallback Komponente**
  http://localhost:9080/?delay



DEMO: LAZY UND SUSPENSE

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



React.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>
    <>
  }
}
```

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

SUSPENSE FOR DATA FETCHING

unstable api!

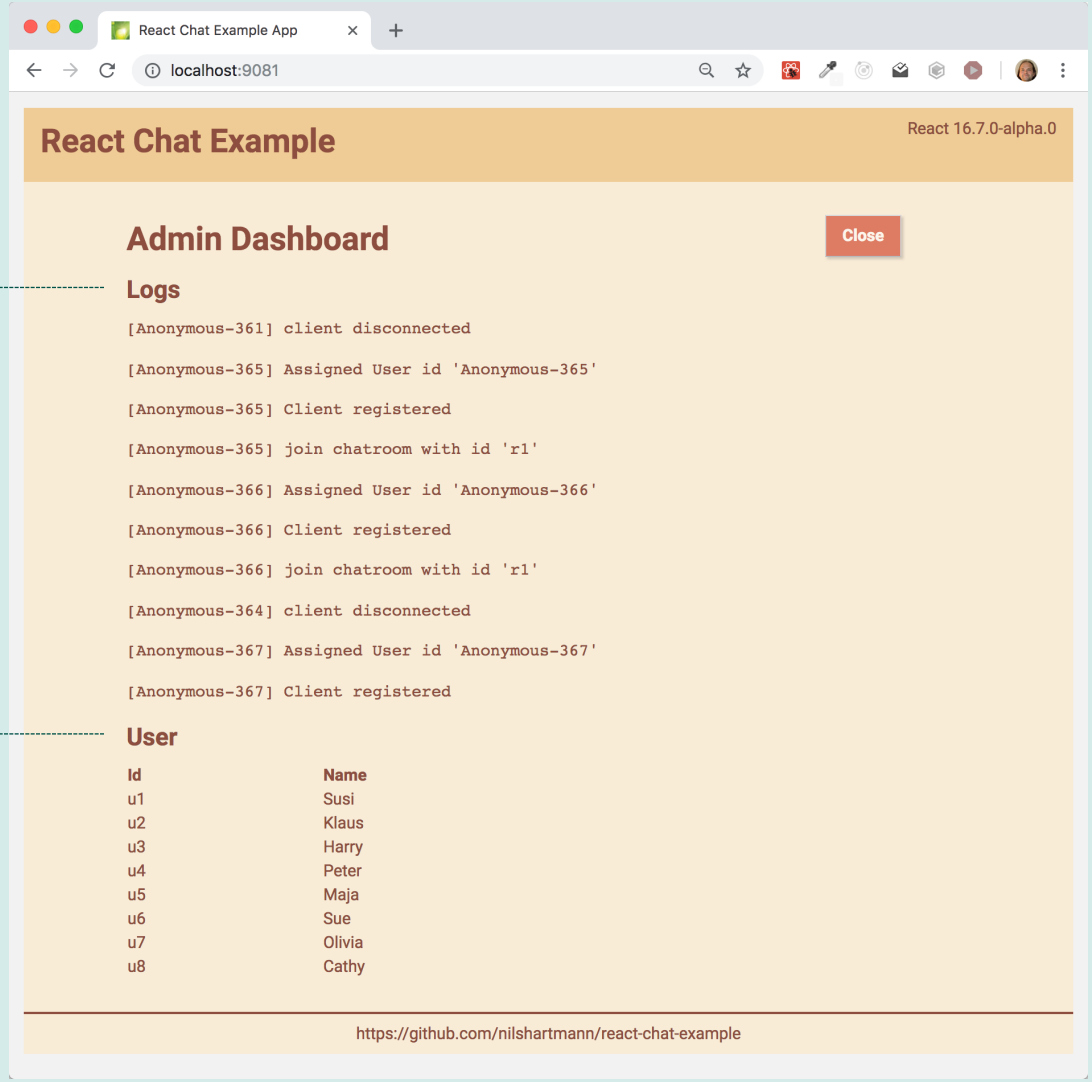
Ausblick [16.x]: Suspense for Data Fetching

- *Alle gezeigten Beispiele verwenden unstable API!!*

BEISPIEL: DATEN LADEN MIT SUSPENSE

unstable api!

- REST Aufrufe mit fetch http://localhost:9081/dashboard?delayfetch



ASYNCHRONES DATEN LADEN

unstable api!

- **"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: logs })
  }

  render() {
    if (!this.state.logs) { return <h1>Loading...</h1> }
    return <div> // render logs </div>;
  }
}
```

DATEN LADEN MIT SUSPENSE - 1

unstable api!

- **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
  - Daten werden aus react-cache kommen (unstabile API zurzeit)

```
function LogsView() {
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück

  return <div> ...geladene logs hier anzeigen... </div>;
}
```

DATEN LADEN MIT SUSPENSE - 2

unstable api!

- **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 fallback={...}>
    <LogsView />
  </Suspense>
}
```

AUSBLICK: SUSPENSE AUF DEM SERVER

unstable api!

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

Concurrent Mode

AUSBLICK

CONCURRENT REACT

Time Slicing: Bessere Nutzung von CPU Zeiten

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

CONCURRENT REACT

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)

CONCURRENT REACT

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 Komponeten

CONCURRENT MODE

unstable api!

Scheduler: Erlaubt es, Aktionen zu priorisieren

- "Unwichtige" Aktionen niedriger priorisieren (z.B. Grafik aktualisieren)
- Wichtige Aktionen (z.B. User-Interaktion) bleiben dadurch flüssig

```
import {
  unstable_LowPriority,
  unstable_runWithPriority,
  unstable_scheduleCallback
} from "scheduler";

function deferLoadUnimportantData(url) {
  unstable_runWithPriority(unstable_LowPriority, function() {
    unstable_scheduleCallback(function() {
      loadUnimportantData(url);
    });
  });
}
```

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~~

NILS HARTMANN
https://nilshartmann.net

Vielen Dank!

Slides: https://nils.buzz/bedcon19-react

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

Fragen & Kontakt: nils@nilshartmann.net