

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



React

IN ENTERPRISE-ANWENDUNGEN

Slides: <http://bit.ly/jax2018-react-enterprise>

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

NILS HARTMANN

Programmierer aus Hamburg

Java
JavaScript, TypeScript, React

Trainings, Workshops

nils@nilshartmann.net

@NILSHARTMANN

Enterprise

Anwendungen

Enterprise Anwendungen

- Große Code-Basis
- Viele Entwickler, mehrere Teams
- Langlebig

Herausforderungen: beim Entwickeln

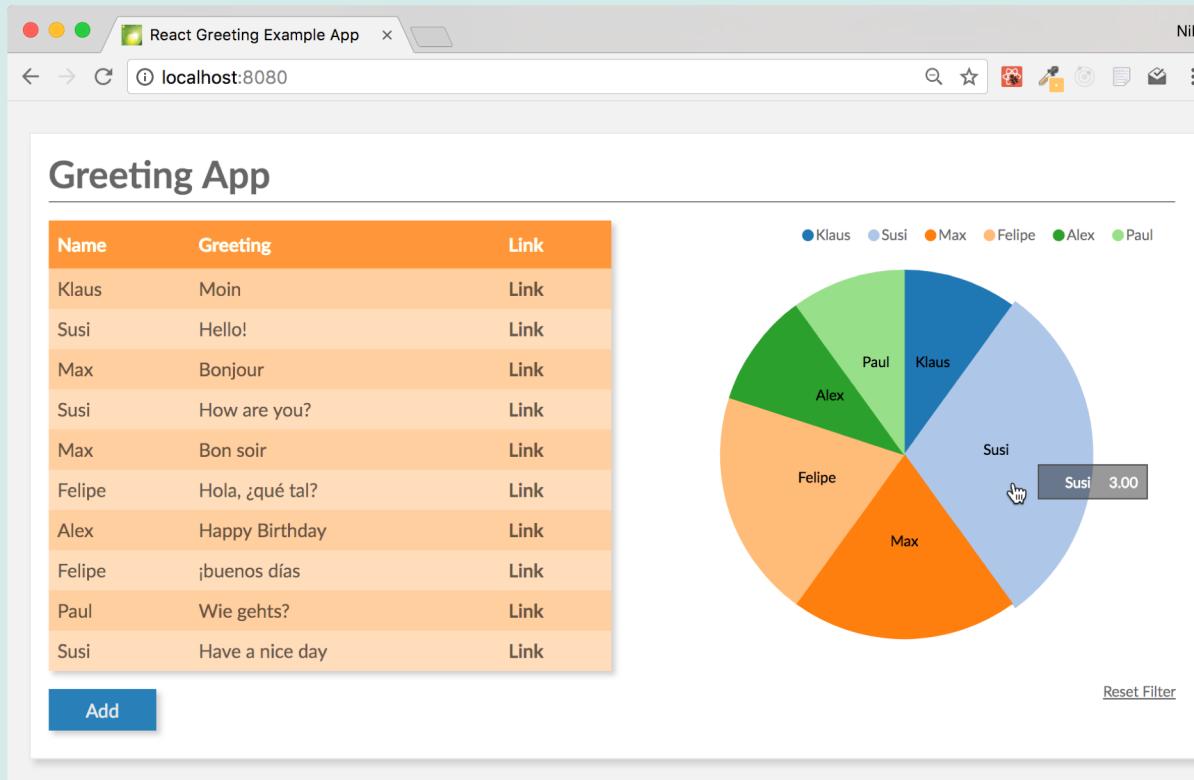
- Wartbarkeit
- Mehrere Entwickler und Teams müssen Code bearbeiten und verstehen
 - Am besten auch noch nach einem Jahr
- Neue Team-Mitglieder müssen Code verstehen

Herausforderungen: beim Entwickeln

- Wartbarkeit
- Mehrere Entwickler und Teams müssen Code bearbeiten und verstehen
 - Am besten auch noch nach einem Jahr
- Neue Team-Mitglieder müssen Code verstehen

Herausforderungen: zur Laufzeit

- Performance
- Bundle Größen
 - Auswirkung auf Start der Anwendung



Beispiel Anwendung

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

Code Struktur

für React-Anwendungen

CODE STRUKTUR

Es gibt kein "richtig" oder "falsch"

Es gibt kein "richtig" oder "falsch"

"Facebook has 30,000 react components.

How do you manage large project directories with many components?"

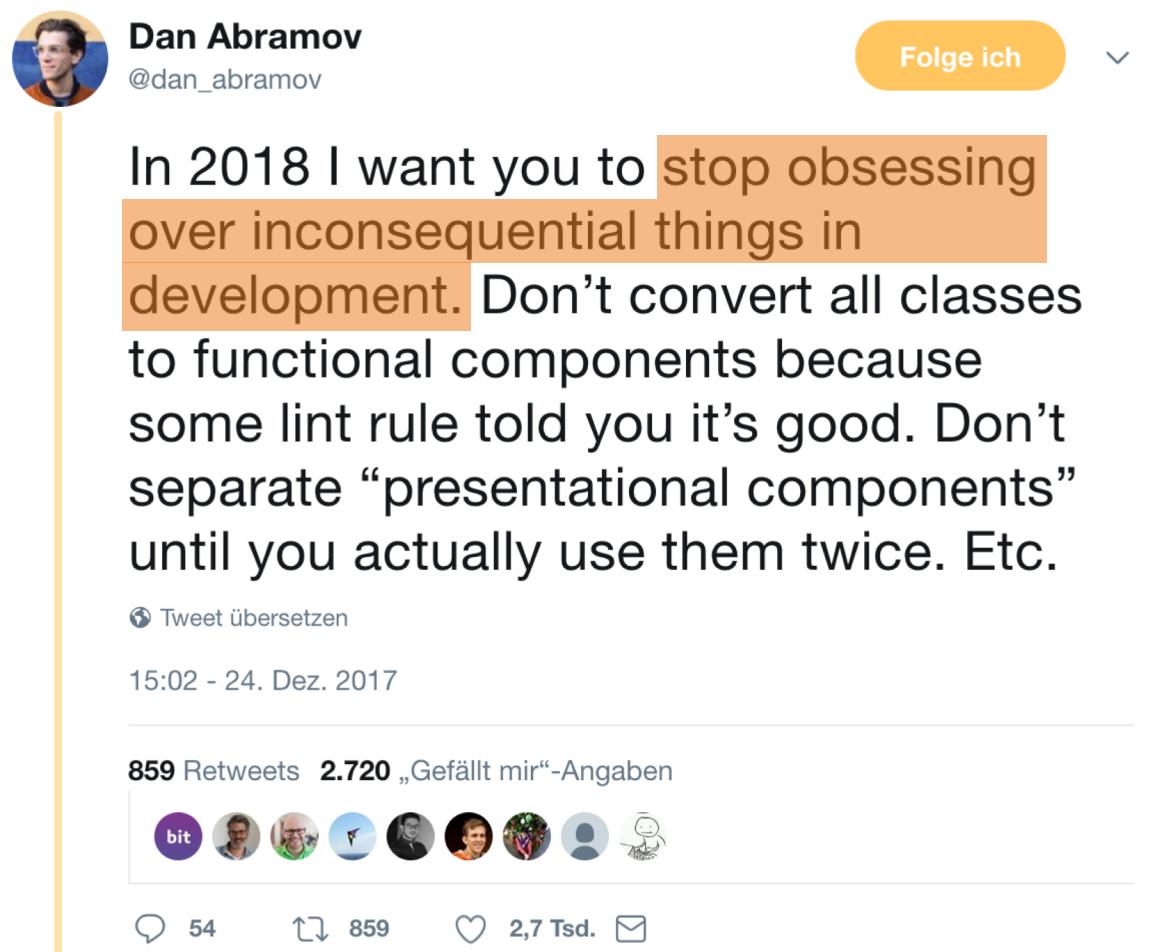
"...each JS filename is unique and can be imported absolutely from any other file in the source.

This means filenames are relatively verbose, e.g. require('AdsManagerPrivacyDialogButton'), ...

You might think this is a terrible idea, but it actually works great for us. ..."

(https://www.reddit.com/r/reactjs/comments/6al7h2/facebook_has_30000_react_components_how_do_you/)

Es gibt kein "richtig" oder "falsch"



A screenshot of a Twitter post from Dan Abramov (@dan_abramov). The post features a profile picture of Dan Abramov and the text: "In 2018 I want you to stop obsessing over inconsequential things in development. Don't convert all classes to functional components because some lint rule told you it's good. Don't separate “presentational components” until you actually use them twice. Etc." Below the text are engagement metrics: 859 Retweets, 2.720 „Gefällt mir“-Angaben, 54 comments, 859 likes, and 2,7 Tsd. shares.

In 2018 I want you to stop obsessing over inconsequential things in development. Don't convert all classes to functional components because some lint rule told you it's good. Don't separate “presentational components” until you actually use them twice. Etc.

859 Retweets 2.720 „Gefällt mir“-Angaben

54 859 2,7 Tsd.

https://twitter.com/dan_abramov/status/944931372820549632

Es gibt kein "richtig" oder "falsch"

"Facebook has 30,000 react components.

Was passt für Euer Team am besten?

"...each JS filename is unique and can be imported absolutely from any other file in the source.

This means filenames are relatively verbose, e.g. require('AdsManagerPrivacyDialogButton'), ...

You might think this is a terrible idea, but it actually works great for us. ..."

(https://www.reddit.com/r/reactjs/comments/6al7h2/facebook_has_30000_react_components_how_do_you/)

Es gibt kein "richtig" oder "falsch"

"Facebook has 30,000 react components.

Was passt für Euer Team am besten?

"...each JS filename is unique and can be imported absolutely from any other file in the source.

This means filenames are relatively verbose e.g. `require('AdsManagerPrivacyDialogButton')`, ...

If you're just starting a project, don't spend more than five minutes on
You might choose a file structure.

(<https://reactjs.org/docs/faq-structure.html>) If you feel completely stuck, start by keeping all files in a single folder.

In general, it is a good idea to keep files that often change together close to each other. This principle is called “colocation”.

(<https://reactjs.org/docs/faq-structure.html>)

CODE STRUKTUR

"Colocation": Komponenten in React

Klassische Aufteilung

Logik, Model
(JS)



View
(HTML, Template)



Gestaltung
(CSS)



Aufteilung in Komponenten

Button



Eingabefeld



Label



Evolution einer Komponente - 1

Starten: Alles in der render-Funktion

```
export default class Greeting extends React.Component {  
  render() {  
    const { greeting, name } = this.props;  
  
    return <>  
      <h1>Greeting for {name}</h1>  
      <div className="...">{greeting}</div>  
    </>  
  };  
}
```

Evolution einer Komponente - 2

Ggf. auslagern in einzelne Render Funktionen

- "Umstrittener" Ansatz
- Vorteil: State und Props vorhanden

```
export default class Greeting extends React.Component {  
    renderName() { . . . }  
    renderPhrase() { . . . }  
  
    render() {  
        return <>  
            { this.renderName() }  
            { this.renderPhrase() }  
        </>  
    }  
}
```

Evolution einer Komponente - 3

"Private Komponenten" extrahieren

- In einer Datei, aber nicht exportiert

```
class GreetingName extends React.Component {...}
class GreetingPhrase extends React.Component {...}

export default class Greeting extends React.Component {

  render() {
    return <>
      <GreetingName name="..." />
      <GreetingTitle phrase="..." />
    </>
  }
}
```

Evolution einer Komponente - 4

Komponenten extrahieren

- Wenn Datei zu groß wird oder Komponenten wiederverwendet werden

```
// GreetingName.js
export default class GreetingName extends React.Component {...}

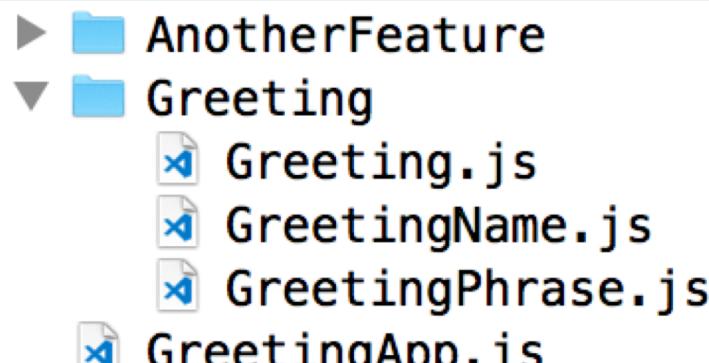
// GreetingPhrase.js
export default class GreetingPhrase extends React.Component {...}

// Greeting.js
import GreetingName from "./GreetingName";
import GreetingPhrase from "./GreetingPhrase";
export default class Greeting extends React.Component {
  ...
}
```

Evolution einer Komponente / Anwendung - 1

Aufteilen in Verzeichnisse

- Gruppieren nach Fachlichkeit/Feature (nicht technisch!)
- Nicht zu stark schachteln



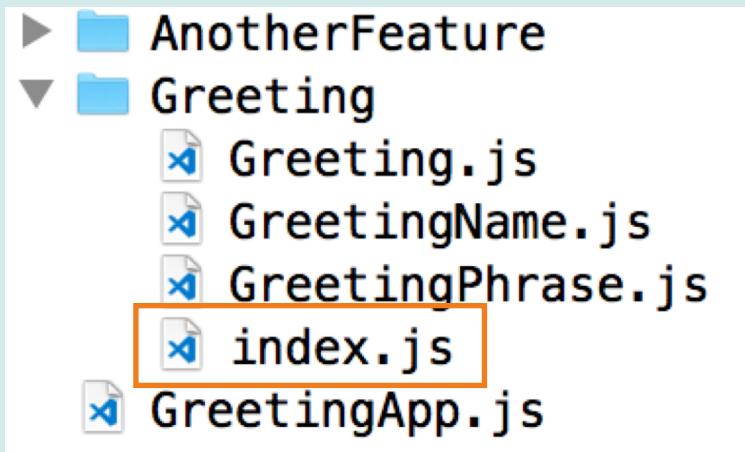
```
// GreetingApp.js
import Greeting from "./Greeting/Greeting";

class GreetingApp extends React.Component { . . . }
```

Evolution einer Komponente / Anwendung - 2

Aufteilen in Verzeichnisse

- index.js exportiert "sichtbare" Komponente



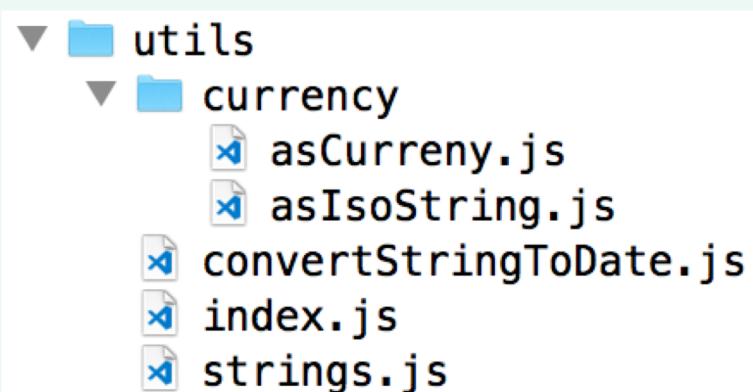
```
// index.js
export { default } from "./Greeting";
```

```
// GreetingApp.js
import Greeting from "./Greeting";

class GreetingApp extends React.Component { . . . }
```

index.js (re-)exportiert sichtbare Teile eines Ordners

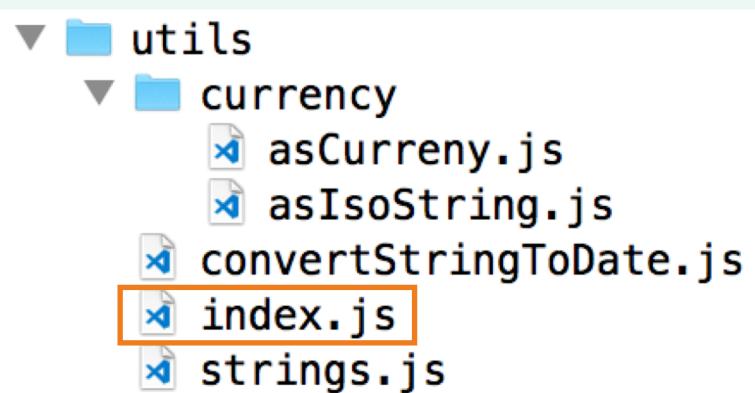
- Problem: interne Struktur
- Problem: was soll sichtbar sein?



EXKURS: ÖFFENTLICHE SCHNITTSTELLE VS INTERNE STRUKTUR

index.js (re-)exportiert sichtbare Teile eines Ordners

- Problem: interne Struktur
- Problem: was soll sichtbar sein?

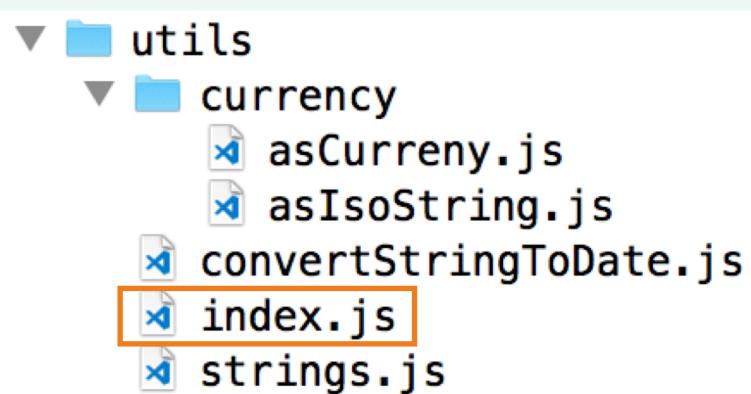


```
// index.js
export * from "./strings";
export * from "./convertStringToDate";
export * from "./currency/asCurrency";
export * from "./currency/asIsoString";
```

EXKURS: ÖFFENTLICHE SCHNITTSTELLE VS INTERNE STRUKTUR

index.js (re-)exportiert sichtbare Teile eines Ordners

- Problem: interne Struktur
- Problem: was soll sichtbar sein?



```
// index.js
export * from "./strings";
export * from "./convertStringToDate";
export * from "./currency/asCurrency";
export * from "./currency/asIsoString";
```

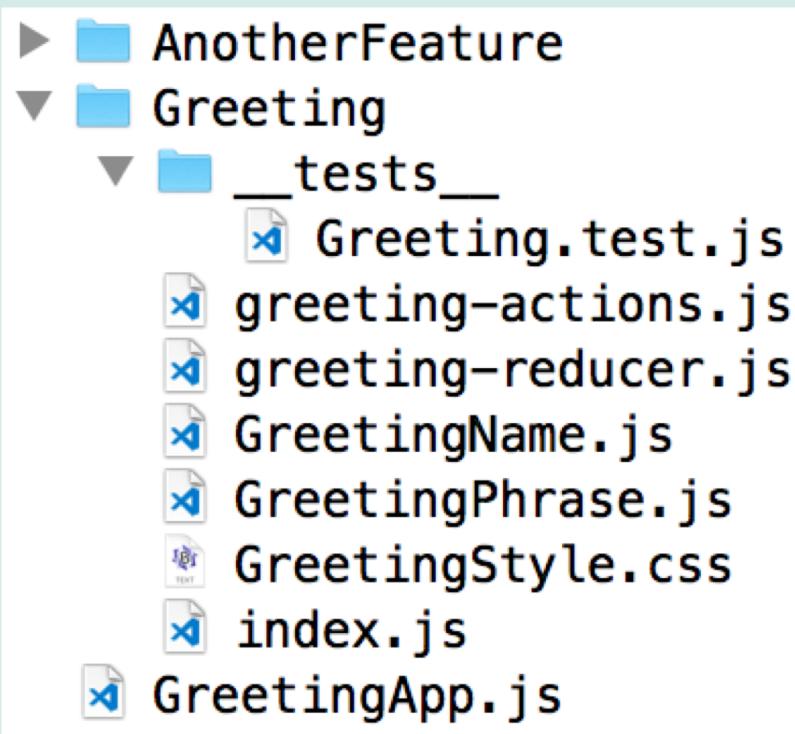
// Verwender

```
import { } from "./utils";
    asCurrency
    asIsoString
    convertStringToDate
    shorten          function shorten(): void ⓘ
    translate
```

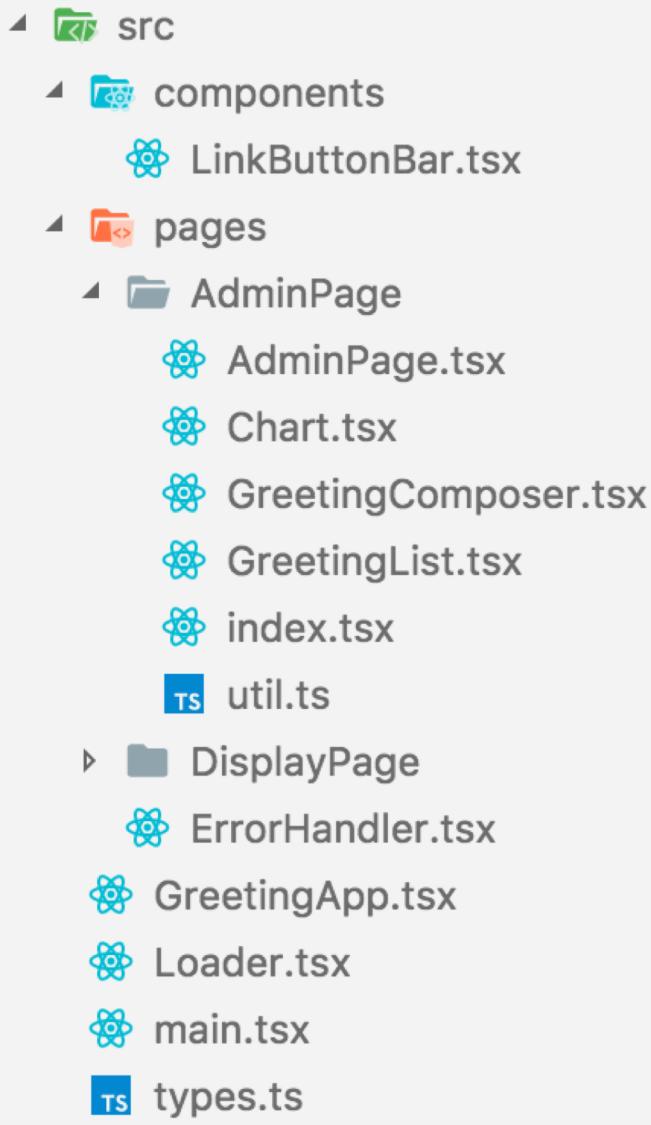
Evolution einer Komponente / Anwendung - 3

Aufteilen in Verzeichnisse

- Alles "relevante" kommt in das Verzeichnis
- Auch Redux-Dateien (umstritten!), CSS, Tests...



BEISPIEL: GREETING APP



Gemeinsam genutzte Komponenten (Kandidat für eigenes Modul)

Fachliche Gliederung der Anwendung, angelehnt an Routen

Applicationmodule (Root Routen, Redux, ...)

Einstiegspunkt (ReactDOM.render)

Große Anwendungen in mehrere (npm) Module zerlegen

- "Fachliche" Verzeichnisse können Kandidaten sein
- Lohnt sich nur bei mehrfach verwendeten Modulen
- Module können in (private) npm Registry veröffentlicht werden
 - z.B. Nexus
 - `npm publish --registry https://our-nexus.com`
- Guter Kandidat: Komponentenbibliothek mit wirklich wiederverwendbaren Komponenten

npm link: Module lokal (zum Testen) verwenden

- Problem: langer turn-around beim Arbeiten mit Registry
- npm link / yarn link
- Verwendet Modul aus lokalem Verzeichnis/Workspace statt Registry

2. greeting-components (bash)

```
$ yarn link
yarn link v1.6.0
success Registered "greeting-components".
info You can now run `yarn link "greeting-components"`
  in the projects where you want to use this package
  and it will be used instead.
✨ Done in 0.03s.

nils at MacBook-Pro-2 in ~/develop/javascript/greeting-components
$
```

2. react-greeting-example-16.3 (bash)

```
$ yarn link greeting-components
yarn link v1.6.0
warning package.json: No license field
success Using linked package for "greeting-components"
.
✨ Done in 0.04s.

nils at MacBook-Pro-2 in ~/develop/javascript/react-greeting-example-16.3 on master [?]
$
```

Schritt 1: Bereitstellen: yarn link

Schritt 2: Verwenden: yarn link modul-name

Fehler-Handling

Fehler vermeiden & behandeln

Neue Komponente: React.StrictMode (16.3)

- Nur im Development Modus aktiv
- Warnt vor typischen React-Fehlern
 - Zum Beispiel Lifecycle-Hooks, die deprecated sind
- Wird künftig wohl noch weitere Prüfungen geben

```
class GreetingApp extends React.Component {  
  render() {  
    return <React.StrictMode>  
      // hier kommt die Anwendung  
    </React.StrictMode>;  
  }  
}
```

Neue Komponente: React.StrictMode (16.3)

- Nur im Development Modus aktiv
- Warnt vor typischen React-Fehlern
 - Zum Beispiel Lifecycle-Hooks, die deprecated sind
- Wird künftig wohl noch weitere Prüfungen geben

```
class GreetingApp extends React.Component {  
  render() {  
    return <React.StrictMode>  
      // hier kommt die Anwendung  
    </React.StrictMode>;  
  }  
}
```

The screenshot shows a browser's developer tools console output. At the top, it says "warning.js:33". Below that, there is a red warning message: "Warning: Unsafe lifecycle methods were found within a strict-mode tree: componentWillMount: Please update the following components to use componentDidMount instead: Route componentWillReceiveProps: Please update the following components to use static getDerivedStateFromProps instead: Chart, Route". At the bottom, it says "Learn more about this warning here: https://fb.me/react-strict-mode-warnings".

```
✖ ▶ Warning: Unsafe lifecycle methods were found within a strict-mode tree:  
  in Switch (created by GreetingApp)  
  in div (created by GreetingApp)  
  in Router (created by BrowserRouter)  
  in BrowserRouter (created by GreetingApp)  
  in GreetingApp  
componentWillMount: Please update the following components to use componentDidMount instead: Route  
componentWillReceiveProps: Please update the following components to use static  
getDerivedStateFromProps instead: Chart, Route  
Learn more about this warning here:  
https://fb.me/react-strict-mode-warnings
```

Globale Fehlerbehandlung: componentDidCatch

- Lifecycle Hook seit React 16
- Fängt Fehler auf, die beim rendern in einer **unterliegenden** Komponente aufgetreten sind

```
class ErrorHandler extends React.Component {  
  
  componentDidCatch(error, errorInfo) {  
    this.setState({ hasError: true })  
  }  
  
}
```

Lifecycle Hook
(könnte Fehler z.B. auch
an einen Server
schicken/alarmieren)

Globale Fehlerbehandlung: componentDidCatch

- Lifecycle Hook seit React 16
- Fängt Fehler auf, die beim rendern in einer **unterliegenden Komponente** aufgetreten sind

```
class ErrorHandler extends React.Component {  
  
  componentDidCatch(error, errorInfo) {  
    this.setState({ hasError: true })  
  }  
  
  render() {  
    return this.state.hasError ?  
      <h1>Ein Fehler aufgetreten!</h1>  
      :  
      this.props.children;  
  }  
}
```

Lifecycle Hook
(könnte Fehler z.B. auch
an einen Server
schicken/alarmieren)

Fehlermeldung
anzeigen

ERROR BOUNDARIES

Globale Fehlerbehandlung: componentDidCatch

- Verwendung

```
class GreetingApp extends React.Component {  
  
  render() {  
    return <ErrorHandler>  
      <FachlicheKomponenten .... />  
    </ErrorHandler>  
  }  
}
```

TypeScript

für React-Anwendungen

Komponenten Props als Typen in TypeScript

```
function GreetingList(props: GreetingListProps) {  
  return <table>{props.greetings.map(...)}</table>;  
}  
  
interface Greeting { . . . };  
interface GreetingListProps {  
  greetings: Greeting[]  
};
```

Typ definieren

Überprüfung zur
Compile-Zeit
(auch direkt in der IDE)

```
[ts]  
Type '{ greetings: string; }' is not assignable to type 'I  
ntrinsicAttributes & GreetingListProps'.  
  Type '{ greetings: string; }' is not assignable to type  
'GreetingListProps'.  
    Types of property 'greetings' are incompatible.  
      Type 'string' is not assignable to type 'Greeting  
[]'.  
(JSX attribute) greetings: string  
<GreetingList greetings="hello" />
```

TYPESCRIPT UND REACT: PROPERTIES & STATE

Komponenten-Klassen - 1

- Types für Properties und State definieren

Typ für Props

```
interface GreetingComposerProps {  
  initialName: string;  
  onSave(newGreeting: NewGreeting) => void;  
};
```

Typ für State

```
interface GreetingComposerState = {  
  name: string;  
  greeting: string;  
};
```

TYPESCRIPT UND REACT: PROPERTIES & STATE

Komponenten-Klassen - 2

- Types in Komponenten-Klasse angeben

Typ für Props

```
interface GreetingComposerProps {  
  initialName: string;  
  onSave(newGreeting: NewGreeting) => void;  
};
```

Typ für State

```
interface GreetingComposerState = {  
  name: string;  
  greeting: string;  
};
```

Typen angeben

```
class GreetingComposer extends React.Component  
  <GreetingComposerProps, GreetingComposerState> {  
  . . .  
}
```

TYPESCRIPT UND REACT: PROPERTIES & STATE

Komponenten-Klassen - 3

- State initialisieren (optional)

```
class GreetingComposer extends React.Component<GreetingComposerProps, GreetingComposerState> {  
  
    readonly state: GreetingComposerState = {  
        greeting: "",  
        name: ""  
    }  
    . . .  
}
```

TYPESCRIPT UND REACT: PROPERTIES & STATE

Typische Fehler, die durch TypeScript aufgedeckt werden

Potentielle Fehler

```
// Properties sind read-only  
this.props.initialName = null;  
  
// Nur bekannte Properties dürfen verwendet werden  
const x = this.props.not_here;  
  
// State muss vollständig initialisiert werden  
this.state = {name: ""}; // greeting fehlt  
  
// this.state darf nur im Konstruktor verwendet werden  
this.state.name = ""; // außerhalb des Cstr  
  
// Elemente im State müssen korrekten Typ haben  
this.setState({name: 7}); // 7 is not a string  
  
// Unbekannte Elemente dürfen nicht in den State  
gesetzt werden  
this.setState({notHere: 'invalid'});
```

Build Optimierung

Bundle-Größe optimieren

BUNDLE OPTIMIERUNG

Problem: Große Bundle-Datei

| Name | Größe |
|---|--------|
|  main.js | 5.8 MB |

- Browser muss viele Daten laden (**Netzwerk!**)
- Browser muss große JavaScript-Datei parsen (**CPU!**)
- Browser muss das JavaScript ausführen (**CPU!**)
- ...erst jetzt ist die Anwendung bereit!

Schritt 1: Production-Mode von Webpack

Kommandozeile

```
webpack --mode production
```

oder webpack.config.js:

```
module.exports = {
  entry: . . .,
  output: { . . . },
  mode: "production"
}
```

| Name | Änderungsdatum | Größe |
|---------|----------------|--------|
| main.js | Heute | 3.3 MB |

```
WARNING in entrypoint size limit: The following entrypoint(s) combined as
set size exceeds the recommended limit (244 KiB). This can impact web per-
formance.
Entrypoints:
  main (3.31 MiB)
    main.js
```

Schritt 2: Source Maps auslagern

- Browser lädt Source Maps erst wenn benötigt

webpack.config.js:

```
module.exports = {  
  entry: . . .,  
  output: { . . . },  
  mode: "production",  
  devtool: "source-map" // z.B. oder none  
}
```

Ergebnis...

| | | |
|---|-------|----------|
|  main.js | Heute | 567.7 KB |
|  main.js.map | Heute | 2 MB |

- In Webpack 4 ist das Standard im **Production Mode** (wenn nichts anderes konfiguriert ist)

Schritt 2: Source Maps auslagern

- Browser lädt Source Maps erst wenn benötigt

webpack.config.js:

```
module.exports = {
  entry: . . . ,
  output: { . . . },
  mode: "production",
  devtool: "source-map" // z.B. oder none
}
```

Ergebnis...

| | | |
|---|-------|----------|
|  main.js | Heute | 567.7 KB |
|  main.js.map | Heute | 2 MB |

...aber immer noch groß

```
WARNING in entrypoint size limit: The following entrypoint(s) combined as
set size exceeds the recommended limit (244 KiB). This can impact web per-
formance.
Entrypoints:
  main (568 KiB)
    main.js
```

Schritt 3: Externe Libs nicht ins Application Bundle

- Stattdessen z.B. über CDN laden
- Können sehr gut gecacht werden (vom Browser u.a.)
- Mehrere parallele Requests zum Laden möglich

webpack.config.js:

```
module.exports = {
  externals: {
    "react": "React",
    "react-dom": "ReactDOM",
    . . .
  }
}
```

index.html

```
<script src="https://.../react.prod.min.js" >
<script src="https://.../react-dom.prod.min.js" />
```

Schritt 3: Externe Libs nicht ins Application Bundle

- Stattdessen z.B. über CDN laden
- Können sehr gut gecacht werden (vom Browser u.a.)
- Mehrere parallele Requests zum Laden möglich

webpack.config.js:

```
module.exports = {
  externals: {
    "react": "React",
  }
}
```

| Name | Änderungsdatum | Größe |
|-------------|----------------|----------|
| main.js.map | Heute | 281.6 KB |
| main.js | Heute | 69.7 KB |

index.html

```
<script src="https://.../react.prod.min.js" >
<script src="https://.../react-dom.prod.min.js" />
<script src="https://.../react-dom.prod.min.js" />
```

Code Splitting

Asynchrone Importe

Asynchrones Laden von Modulen

- Erlaubt das dynamische Nachladen von Code-Teilen
- Beim Start der Anwendung nur direkt benötigte Teile laden (Minimalversion)
- Weitere Teile werden erst bei Benutzerinteraktion oder im Hintergrund geladen
- Basiert auf `dynamic import` (Nicht Standard, aber Stage 3)

<https://reactjs.org/docs/code-splitting.html>

<https://webpack.js.org/guides/code-splitting/>

CODE SPLITTING

Beispiel: Ein Modul dynamisch importieren - 1

Ein JS Modul
(calculator.js)

```
// calculator.js
export default function calculator(a, b) {
    return a+b;
}
```

CODE SPLITTING

Beispiel: Ein Modul dynamisch importieren - 2

Ein JS Modul
(calculator.js)

```
// calculator.js
export default function calculator(a, b) {
  return a+b;
}
```

Zum Vergleich: statischer
Import

```
// Verwender
import calculator from "./calculator";
console.log(calculator(7,8));
```

CODE SPLITTING

Beispiel: Ein Modul dynamisch importieren - 3

Ein JS Modul
(calculator.js)

```
// calculator.js
export default function calculator(a, b) {
  return a+b;
}
```

Zum Vergleich: statischer Import

```
// Verwender
import calculator from "./calculator";
console.log(calculator(7,8));
```

1. Modul wird asynchron geladen

```
// Verwender (asynchroner Import)
import("./calculator").
then(calculatorModule => {
  const calculator = calculatorModule.default;
  console.log(calculator(7, 8));
})
```

2. Modul steht zur Verfügung

Nachladen von React-Komponenten

- Erfordert die Darstellung von Platzhaltern, bis die eigentliche Komponente geladen ist
- Wenn die eigentliche Komponente geladen ist, muss die umschließende Komponente neu dargestellt werden
 - Geladene Komponente kann in den State gesetzt werden
 - Alternativ *forceUpdate* zum neuen rendern
- Fertige Lösung: react-loadable
(<https://github.com/jamiebuilds/react-loadable>)

CODE SPLITTING

Beispiel: Eine React-Komponente nachladen - 1

Eine Loader-
Komponente

```
class LoadableGreetingComposer extends React.Component {  
    state = {};
```

Ziel-Komponente
laden

```
    async componentDidMount() {  
        const result = await import("./GreetingComposer");  
        this.setState  
            ({ GreetingComposer: result.default });  
    }  
}
```

```
}
```

CODE SPLITTING

Beispiel: Eine React-Komponente nachladen - 2

Eine Loader-Komponente

```
class LoadableGreetingComposer extends React.Component {  
    state = {};
```

Ziel-Komponente laden

```
    async componentDidMount() {  
        const result = await import("./GreetingComposer");  
        this.setState  
            ({ GreetingComposer: result.default });  
    }  
  
    render() {  
        if (this.state.GreetingComposer) {  
            return <this.state.GreetingComposer {...this.props} />;  
        }  
  
        return <span>Loading...</span>;  
    }  
}
```

Ziel-Komponente anzeigen oder Platzhalter

CODE SPLITTING

Beispiel: Eine React-Komponente nachladen - 3

Eine Loader-Komponente

```
class LoadableGreetingComposer extends React.Component {  
  state = {};
```

Ziel-Komponente laden

```
  async componentDidMount() {  
    const result = await import("./GreetingComposer");  
    this.setState  
      ({ GreetingComposer: result.default });  
  }
```

```
  render() {  
    if (this.state.GreetingComposer) {  
      return <this.state.GreetingComposer {...this.props} />;  
    }  
  
    return <span>Loading...</span>;  
  }  
}
```

Verwendung

```
<LoadableGreetingComposer initialValue="Klaus" />
```

CODE SPLITTING

Webpack: Ausgabe-Datei benennen

```
class LoadableGreetingComposer extends React.Component {  
    state = {};  
  
    async componentDidMount() {  
        const result =  
            await import(/*GreetingComposer*/ "./GreetingComposer");  
        this.setState  
            ({ GreetingComposer: result.default });  
    }  
    . . .  
}
```

Gewünschten
Namen beim
Import als
Kommentar angeben

Fertige Lösung: React Loadable

- <https://github.com/jamiebuilds/react-loadable>
- Fehler-Handling
- Vor-Laden von Komponenten
- Unterstützung für Server-Site Rendering

Fertige Lösung: React Loadable

- <https://github.com/jamiebuilds/react-loadable>
- Fehler-Handling
- Vor-Laden von Komponenten
- Unterstützung für Server-Site Rendering

Eine Loading-Komponente

```
function LoadingComponent({error}) {  
  return error ? "Error 😥" : "Loading...";  
}
```

Fertige Lösung: React Loadable

- <https://github.com/jamiebuilds/react-loadable>
- Fehler-Handling
- Vor-Laden von Komponenten
- Unterstützung für Server-Site Rendering

Eine Loading-Komponente

```
function LoadingComponent({error}) {  
  return error ? "Error ☹" : "Loading...";  
}
```

```
import Loadable from "react-loadable";
```

```
const LoadableComposer = Loadable({  
  loader: () => import("./GreetingComposer") ,  
  loading: LoadingComponent  
});
```

Die Loader-Komponente

Fertige Lösung: React Loadable

- <https://github.com/jamiebuilds/react-loadable>
- Fehler-Handling
- Vor-Laden von Komponenten
- Unterstützung für Server-Site Rendering

Eine Loading-Komponente

```
function LoadingComponent({error}) {  
  return error ? "Error ☹" : "Loading...";  
}
```

```
import Loadable from "react-loadable";
```

Die Loader-Komponente

```
const LoadableComposer = Loadable({  
  loader: () => import("./GreetingComposer") ,  
  loading: LoadingComponent  
});
```

Verwendung

```
<LoadableComposer />
```

CODE SPLITTING

Beispiel: Code Splitting mit React Router und React Loadable

```
class LoadingComponent { . . . }

const LoadableAdminPage = Loadable({
  loader: () => import("./pages/AdminPage"),
  loading: LoadingComponent
});

const LoadableDisplayPage = Loadable({
  loader: () => import("./pages/DisplayPage"),
  loading: LoadingComponent
});

const GreetingApp = () => (
  <Router>
    <Route path="/greet/:greetingId" component={LoadableDisplayPage} />
    <Route path="/" component={LoadableAdminPage} />
  </Router>
);
```

CODE SPLITTING

Beispiel: Code Splitting mit React Router und React Loadable

```
class LoadingComponent { . . . }

const LoadableAdminPage = Loadable({
  loader: () => import("./pages/AdminPage"),
  loading: LoadingComponent
});

const LoadableDisplayPage = Loadable({
  loader: () => import("./pages/DisplayPage"),
  loading: LoadingComponent
});

const GreetingApp = () => (
  <Router>
    <Route path="/greet/:greetingId" component={LoadableDisplayPage} />
    <Route path="/" component={LoadableAdminPage} />
  </Router>
);
```

Demo

- <http://localhost:8000>

CODE SPLITTING

Beispiel: Code Splitting mit React Router und React Loadable

```
class LoadingComponent { . . . }

const LoadableAdminPage = Loadable({
  loader: () => import("./pages/AdminPage"),
  loading: () => <div>Loading</div>
});

const LoadableDisplayPage = Loadable({
  loader: () => import("./pages/DisplayPage"),
  loading: () => <div>Loading</div>
});

const GreetingApp = () => (
  <Router>
    <Route path="/greet/:greetingId" component={LoadableDisplayPage} />
    <Route path="/" component={LoadableAdminPage} />
  </Router>
);
```

| Name | Größe |
|---------------------|---------|
| AdminPage.js | 6.7 KB |
| DisplayPage.js | 3 KB |
| GreetingComposer.js | 1.4 KB |
| main.js | 65.6 KB |

Caching

Browser Request vermeiden: JavaScript im Browser cachen

- Erfordert eindeutige Dateinamen
 - Hash-Namen mit Webpack generieren

Eindeutige Namen
erzeugen:

```
module.exports = {  
  output: {  
    filename: "[name].[chunkhash].js",  
    ...  
  }  
}
```

Browser Request vermeiden: JavaScript im Browser cachen

- Erfordert eindeutige Dateinamen
 - Hash-Namen mit Webpack generieren
 - index.html mit erzeugten Files generieren

Eindeutige Namen
erzeugen:

```
module.exports = {
  output: {
    filename: "[name].[chunkhash].js",
    ...
  }
}

plugins: [
  new HtmlWebpackPlugin({
    filename: ...,
    template: ...
  })
]
```

index.html generieren:

Browser Request vermeiden: JavaScript im Browser cachen

- Cache-Header im Server setzen

Beispiel: nginx

```
map $sent_http_content_type $expires {
    text/html                 epoch;
    application/javascript     max;
    text/css                  max;
}

server {
    listen ...;
    . . .

    expires $expires;
}
```

CACHING

Browser Request vermeiden: JavaScript im Browser cachen

- Cache-Header im Server setzen

Beispiel: nginx

Demo

- <http://localhost:9000>

```
$expires {  
    epoch;  
    max;  
    max;
```

```
server {  
    listen ...;  
    ...  
  
    expires $expires;  
}
```

Vielen Dank!

<http://bit.ly/jax2018-react-enterprise>

Fragen?

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