Die optimale SPA

Lutz Stallknecht / pixelio.de

Maiborn Wolff
Mensch IT

Über mich

Sebastian Springer

- JavaScript Entwickler
- aus München
- arbeite bei MaibornWolff
- https://github.com/sspringer82
- @basti_springer
- Techstack: Node & React
- https://github.com/sspringer82/addressbook

Über euch

Wer seid ihr denn?

- Name
- Vorkenntnisse
- Erwartungen an diesen Workshop

Die Probleme von SPAs

Ressourcenverbrauch

Wechsel zwischen Sichten

Leere Komponenten

Lange Ladezeit

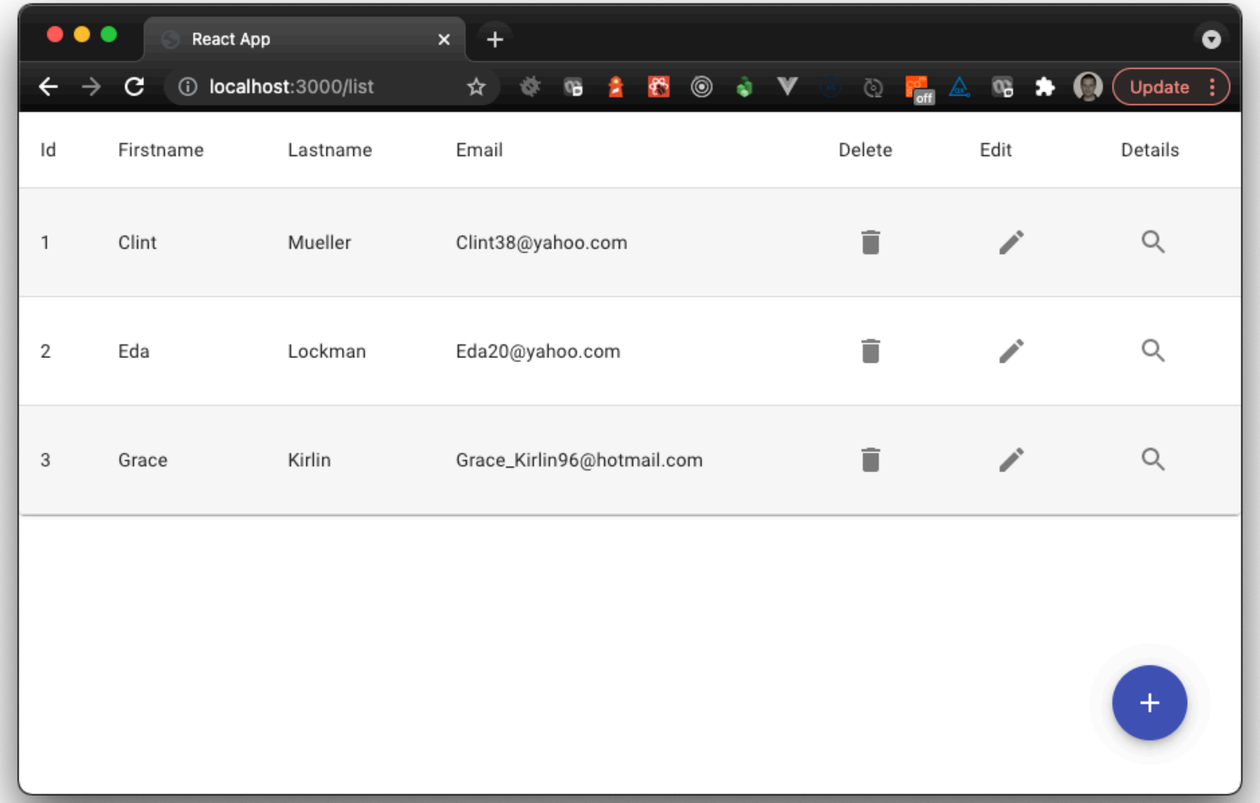
Initial weiße Seite

Rainer Sturm / pixelio.de

Aufgabe 0

Installation der Applikation

- Holt euch den Quellcode: https://github.com/sspringer82/addressbook
- Installiert alle Abhängigkeiten: npm install
- Startet das Frontend: npm start
- Startet das Backend: npm run backend


React App
localhost:3000/list
Update
Id
Firstname
Lastname
Email
Delete
Edit
Details
1
Clint
Mueller
Clint38@yahoo.com
2
Eda
Lockman
Eda20@yahoo.com
3
Grace
Kirlin
Grace_Kirlin96@hotmail.com

Laden der Applikation

Petra Dirscherl / pixelio.de

Das Laden der Applikation

Verbesserung des ersten Aufrufs

- Die größten Problemfelder sind:
  - Ladezeit
  - Paketgröße
- Beide hängen direkt miteinander zusammen

Verarbeitung

Was macht der Browser mit unserem Code?

- HTML laden, verarbeiten & rendern
- Weitere Ressourcen laden (JavaScript, CSS)
- JavaScript (React) ausführen
- React erzeugt die Komponenten und initiales Rendering
- Initialen Lifecycle durchlaufen (useEffect, componentDidMount)
- Daten laden
- Komponenten final rendern + regulären Lifecycle beginnen

Production Build

npm run build

- Sourcecode minifizieren
  - Variablen verkürzen
  - Whitespaces entfernen
  - Kommentare entfernen
- Dateien zusammenfassen
- Struktur für die Auslieferung vorbereiten

```
% yarn build
yarn run v1.22.10
$ react-scripts build
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  41.33 KB  build/static/js/2.18dbf348.chunk.js
  1.56 KB   build/static/js/3.34318162.chunk.js
  1.17 KB   build/static/js/runtime-main.fda51a50.js
  592 B     build/static/js/main.a937e557.chunk.js
  531 B     build/static/css/main.8c8b27cf.chunk.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your pa

The build folder is ready to be deployed.
You may serve it with a static server:

  yarn global add serve
  serve -s build

Find out more about deployment here:

  https://cra.link/deployment

✨  Done in 6.12s.
```

Aufgabe 1
Production Build

- Baut eure Applikation
- Liefert die Applikation aus (passiert automatisch nach dem Build über http://localhost:3001 über das Backend)
- Vergleicht die Ladezeit zwischen beiden Versionen
  - Die besten Resultate erzielt Ihr mit Traffic Shaping (z.B. Slow 3G)

Tree Shaking

Nicht verwendeten Code verwerfen

- Der Buildprozess analysiert den Quellcode
- Nicht verwendeter Code nicht in das Production Bundle aufgenommen
- Der Prozess basiert auf dem Modulsystem
- Auch nicht verwendete Methoden werden verworfen

- TIPP: kleine und unabhängige Funktionen implementieren und exportieren

angieconscious / pixelio.de

Tree Shaking

Wird im Code verwendet

```
export function add(a, b) {
  console.log('this is add');
  return a + b;
}

export function sub(a, b) {
  console.log('this is sub');
  return a - b;
}
```

Wird nicht verwendet

SEARCH

this is add

Replace

files to include

./build

files to exclude

2 results in 2 files - Open in editor

main.363981b1.chunk.js build/static/js 1

(){return 1,2,console.log("this is add"),Object(i.jsx)("div",{className...

main.363981b1.chunk.js.map build/static/js 1

add(a, b) {\n console.log('this is add');\n return a + b;\n}\n\nexport ...

SEARCH

this is sub

Replace

files to include

./build

files to exclude

1 result in 1 file - Open in editor

main.363981b1.chunk.js.map build/static/js 1

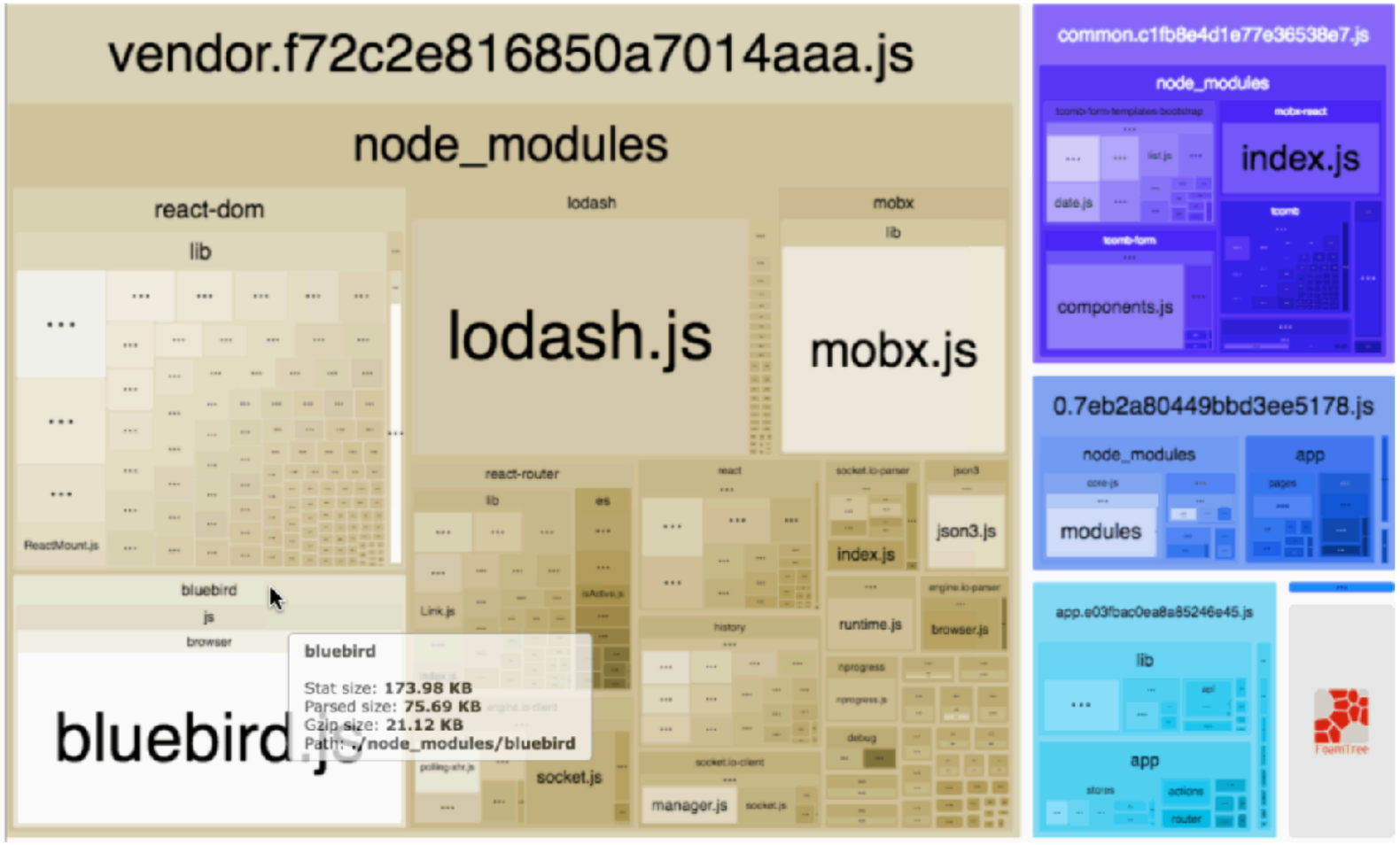
sub(a, b) {\n console.log('this is sub');\n return a - b;\n}\n","const r...

Aufgabe 2
Tree Shaking

- Implementiert je eine Funktion
  - getFullname(address: Address): string => `${address.firstname} ${address.lastname}`
  - getContact(address: Address): string => `${address.firstname} ${address.lastname} <${address.email}>`
- Implementiert das
  - als zwei separate Funktionen
  - als Utility-Klasse mit zwei statischen Methoden
- Bindet die Funktionen in die ListItem-Komponente ein und prüft die Auswirkungen auf das Tree Shaking

Webpack-Bundle-Analyzer

What’s really inside your bundle



https://www.npmjs.com/package/webpack-bundle-analyzer

Aufgabe 3

Webpack Bundle Analyzer

- Verwendet das analyze.js-Script im Repo, um eure gebaute Applikation zu analysieren
  - Ausführung: node analyze.js
- Folgende Pakete müssen zusätzlich installiert werden:
  - webpack-bundle-analyzer
  - progress-bar-webpack-plugin

HTTP Compression

- Server komprimiert alle ausgehenden Pakete
- Client entpackt sie beim Empfang
- Positiv für Netzwerk, negativ für CPU
- Netzwerk-Vorteile überwiegen in der Regel
- Der Browser sendet: 'accept-encoding': 'gzip, deflate, br'

HTTP Compression (gzip)

Leere CRA im Prod Build via Fast 3G

	Ohne Komprimierung	Mit Komprimierung	
Size transferred	670kB	578kB	-92kB
Finish	3,72s	3,22s	-0,5s
DOM Contentloaded	1,96s	1,47s	-0,49s
Load	2,55s	2,05s	-0,5s

Aufgabe 4
Komprimierung

- Schaltet die Komprimierung im Webserver an
  - Npm install compression
  - server.js
    - Const compression = require(‘compression’);
    - app.use(compresson());
- Prüft die Auswirkungen der Komprimierung (Ladezeit und DOM-Events)

Dependency Optimization

Abhängigkeiten und Tree Shaking

- Modulare Architekturen unterstützen Tree Shaking besser
- Manche Bibliotheken verhindern durch ihre Architektur Tree Shaking
  - Moment.js ist ein bekanntes Beispiel dafür
  - Alternativen: date-fns, Luxon, …

Erika Hartmann / pixelio.de

Code Splitting

1 großes Paket vs. mehrere kleine Pakete

- Moderne Bundler unterstützen Code Splitting
- Mehrere kleine Bundles, die parallel geladen werden können
- Kann in Kombination mit Lazy Loading verwendet werden

Code Splitting

Mehrere Bundles im Einsatz

Name	Method	Status	Size	Time	Waterfall
localhost	GET	200	3.3 kB	583…	
main.6dea0f05.chunk.css	GET	200	674 B	564…	
2.5d0841ec.chunk.js	GET	200	177 kB	1.61 s	
main.1a549375.chunk.js	GET	200	10.5 kB	695…	
todo?_embed=subtask	GET	200	957 B	570…	

Lazy Loading

Komponenten bei Bedarf nachladen

- Kombination aus React.lazy und dynamischem Import
- Werden mit Suspense verwendet - Möglichkeit für Fallback-Inhalt
- Komponente ist nicht im initialen Bundle enthalten
- Buildwerkzeuge bauen zusätzliche Bundles
- Routebasiertes Lazy Loading möglich

Lazy Loading

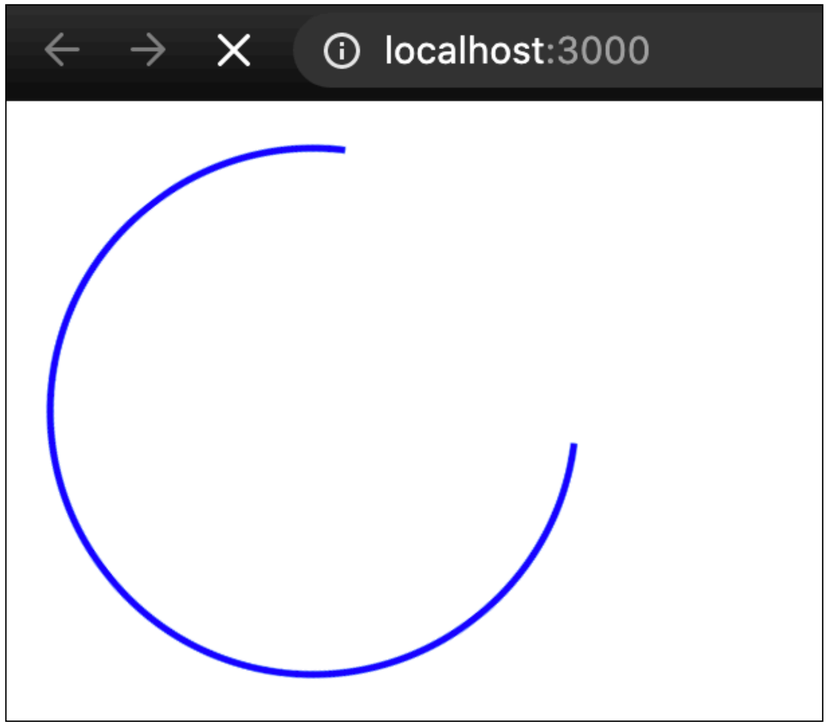
Mit Suspense for Code Splitting

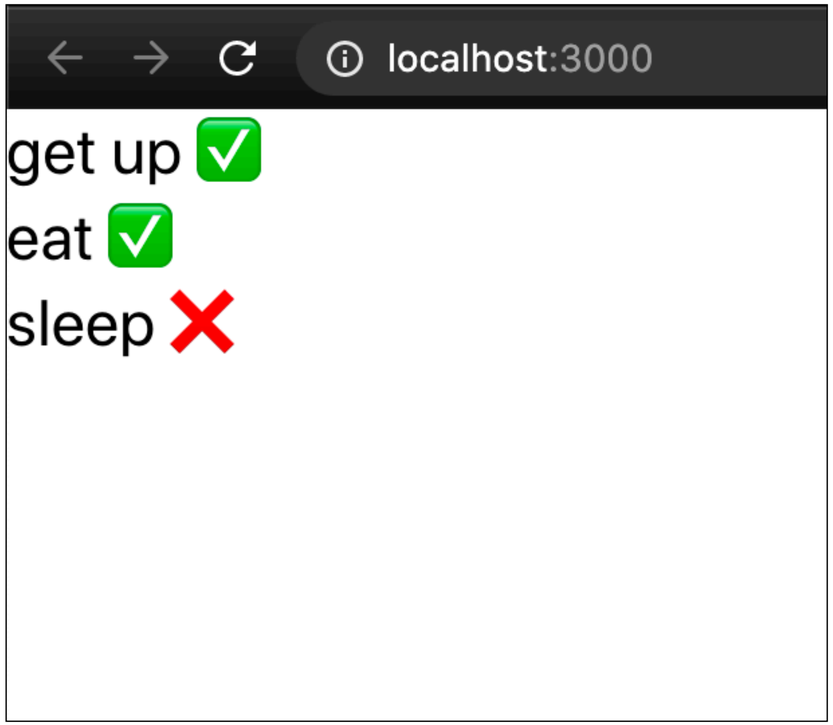
```
const List = React.lazy(() => import('./List'));

function App() {
  return (
    <Suspense
      fallback={<ClipLoader color={'#0000ff'} loading={true} size={150} />}
    >
      <List />
    </Suspense>
  );
}
```

Lazy Loading

Mit Suspense for Code Splitting





Lazy Loading

Mit Suspense for Code Splitting

Name	Method	Status	Size	Time	Waterfall
127.0.0.1	GET	200	3.4 kB	58…	
main.8c8b27cf.chunk.css	GET	200	1.1 kB	56…	
2.6222dcaa.chunk.js	GET	200	155 kB	1.4…	
main.406a65b5.chunk.js	GET	200	1.1 kB	58…	
3.5ae154c7.chunk.js	GET	200	1.8 kB	57…	
todo	GET	200	633 B	57…	

Aufgabe 5
Lazy Loading

- Ladet die ListItem-Komponente lazy
- Nutzt als Platzhalter einen Loading indicator aus dem react-spinners-Paket
  - npm i react-spinners
- Prüft die Struktur, die der Server ausliefert

Lazy Loading

Mit dem React Router

```
const List = React.lazy(() => import('./List'));
const Form = React.lazy(() => import('./Form'));

function App() {
  return (
    <Router>
      <Suspense fallback={<ClipLoader … />}>
        <Route exact path="/list" component={List} />
        …
        <Route exact path="/" exact>
          <Redirect to="/list" />
        </Route>
      </Suspense>
    </Router>
  );
}
```

Aufgabe 6

Lazy Loading mit dem React Router

- Sorgt dafür, dass alle Komponenten im Routing lazy geladen werden
  - Betrifft: List, Form, Detail
- Verwendet hier ebenfalls einen Loading Indicator
- Prüft die Ladereihenfolge und das Verhalten des Browsers

Server Side Rendering

Der Server übernimmt die Hälfte der Arbeit

- Es wird keine weiße Seite ausgeliefert
- Der Server führt React aus und bereitet die HTML Struktur vor
- Hydrate: React übernimmt die Kontrolle über die Struktur im Browser
- Vorteil: Die Benutzer sehen sofort etwas
- Nachteil: Eine Interaktion ist erst nach dem Hydrate-Prozess möglich

Server Side Rendering

Zusammenarbeit mit Node.js

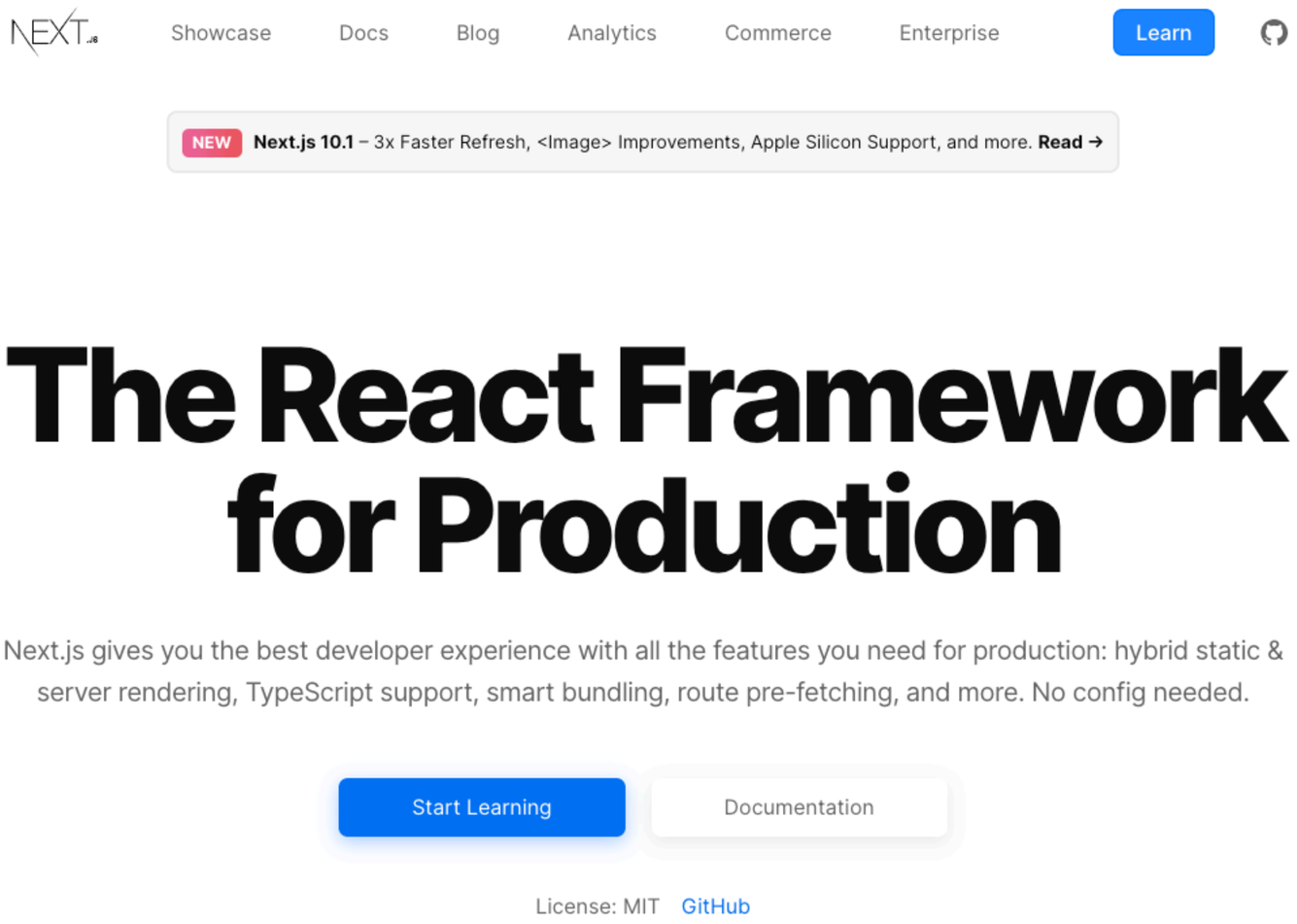
```
app.get('/', (req, res) => {
  ReactDOMServer.renderToNodeStream(
    <Html initialData={JSON.stringify(data)}>
      <App {...data} />
    </Html>,
  ).pipe(res);
});
```

```
ReactDOM.hydrate(<App {...initialData} />, document.getElementById('app'));
```

SSR - Demo

Server Side Rendering

Framework-Unterstützung - https://nextjs.org/



Server Side Rendering

Pre-rendering in Next

- Static Generation: HTML wird zum Buildzeitpunkt vorgeneriert
- Server Side Rendering: HTML wird bei jedem Request erzeugt
- Beide Varianten können mit Client Side Rendering kombiniert werden

Timo Klostermeier / pixelio.de

CDN verwenden

Statische Inhalte schneller ausliefern

- Nur für statische Inhalte (Assets)
- Geografische Nähe zu den Benutzern
- Sehr gute Anbindung
- Viele große Anbieter: Google, Akamai, CloudFront, CloudFlare

Browser Cache

- Browser speichert statische Inhalte zwischen
- Kann über HTTP-Header sehr gut gesteuert werden
  - Beispiel: `Cache-Control: public, max-age=31536000`

Browser Cache

Name	Method	Status	Size	Time	Waterfall
127.0.0.1	GET	304	263 B	572 ms	
main.8c8b27cf.chunk.css	GET	200	(memory cache)	0 ms	
2.6222dcaa.chunk.js	GET	200	(memory cache)	0 ms	
main.406a65b5.chunk.js	GET	200	(memory cache)	0 ms	
3.5ae154c7.chunk.js	GET	200	(memory cache)	0 ms	
todo	GET	200	373 B	567 ms	

Service Workers

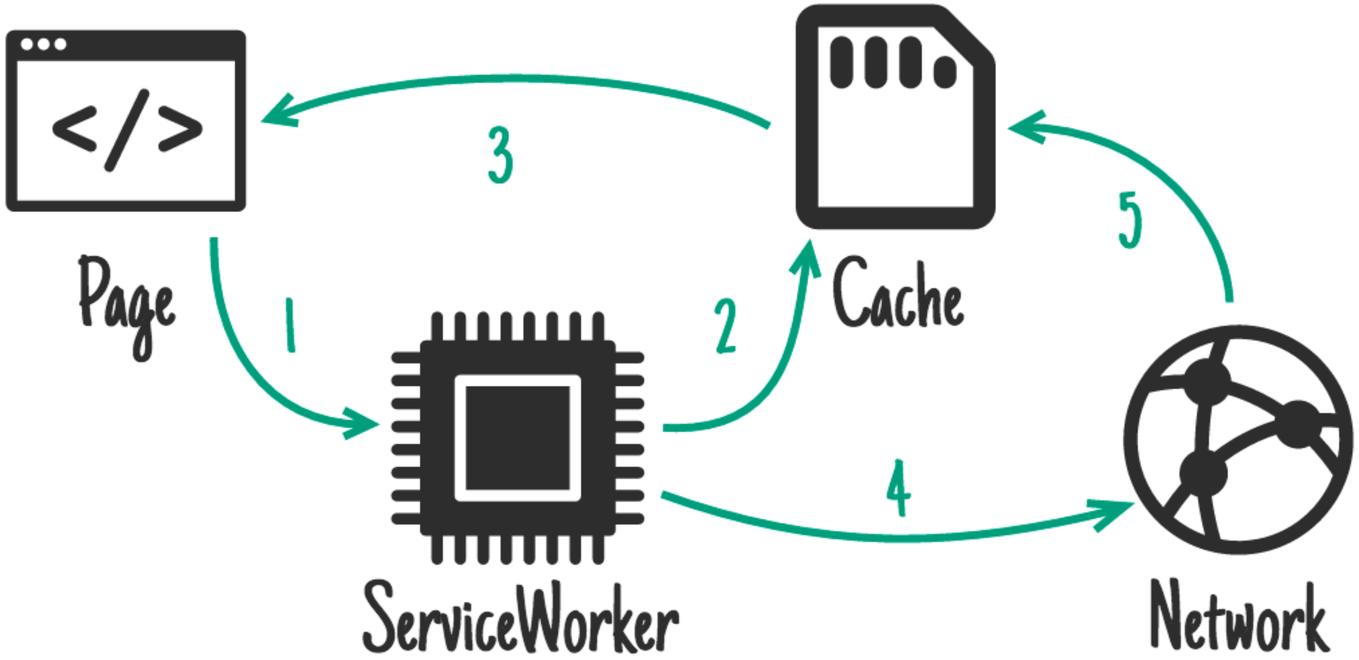
Caching und Offlinefähigkeit

- Service Workers sind weit verbreitetes Browserfeature
- Proxy-Prozess im Browser zwischen Client und Server
- Kann statische und dynamische Anfragen abfangen
- Bessere Kontrollmöglichkeiten aus der Applikation heraus
- Ermöglicht Offlinefähigkeit
- CRA Template verfügbar

Rainer Sturm / pixelio.de

Service Worker

Verschiedene Caching Strategien



https://developers.google.com/web/tools/workbox/modules/workbox-strategies

Aufgabe 8
PWA

- Aktiviert den Service Worker in der index.tsx-Datei und baut die Applikation neu.
- Prüft das Verhalten der Applikation
- BONUS: stellt die Cache-Strategie um

Optimierungen im Code

Gisela Peter / pixelio.de

Optimierungen im Code

Und was können wir jetzt genau tun?

- Generell: darauf achten dass React nicht unnötig viel rendern muss
- Ressourcenauslastung beachten:
  - Arbeitsspeicher
  - CPU

Keine Inline Functions

…außer wo sie unvermeidbar sind

- Werden bei jedem Render-Vorgang neu erzeugt
- Kostet Speicher und beschäftigt den Garbage Collector
- Unabhängige Funktionen können ausgelagert werden
- Abhängige Funktionen können mit useCallback optimiert werden

Memoisierung

Funktionen und Objekte wiederverwenden

- React baut beim Rendern Objekte und Strukturen neu auf
- Die useMemo und useCallback Hooks helfen hier
- Konstante Strukturen, die nur bei Änderungen der Abhängigkeiten neu erzeugt werden

Memoisierung

Beispiel mit useCallback

```
<div key={todo.id}>
  {todo.title} {todo.done ? '✅' : '❌'}
  <button onClick={(e) => {
    setTodos(() => ...)
  }}>delete</button>
</div>
```

ohne useCallback

mit useCallback

```
<div key={todo.id}>
  {todo.title} {todo.done ? '✅' : '❌'}
  <button onClick={useCallback((e) => {
    setTodos(() => ...)
  }, [setTodos])}>delete</button>
</div>
```

Aufgabe 9

UseCallback und useMemo

- Nutzt useCallback und useMemo, um den Eventhandler für den Klick auf den Delete-Button in der ListItem-Komponente zu memoisieren.
- Spoiler: es wird zu keinen merklichen Ergebnissen führen

Memoisierung
ACHTUNG

- Bitte erst optimieren, wenn es Probleme gibt
- Use case: große Komponentenbäume mit häufigen Änderungen
- Nicht jede inline-Funkion in useCallback oder useMemo wrappen
- Bringt nicht in jedem Fall bessere Performance
- Reduziert Lesbarkeit

Immutable Data Structures

- Vorteile:
  - Keine Seiteneffekte (Stichwort Assignment by Reference)
  - Einfachere Change detection
  - Einfacher zu verwalten
- setState führt kein re-render aus, wenn eine tiefe Referenz verändert wird.
- Achtung: Spread ({…items}) erzeugt nur eine flache Kopie

berggeist007 / pixelio.de

Immutable Data Structures

Bibliotheken (eine Auswahl)

- Immutability Helper (https://github.com/kolodny/immutability-helper)
- Immer (https://immerjs.github.io/immer/)
- Immutable.js (https://immutable-js.github.io/immutable-js/)

Immutable Data Structures

Ein Beispiel

```
import produce from 'immer';

...

function handleDelete(id) {
  setTodos((oldTodos) => {
    produce(oldTodos, (draftTodos) => {
      draftTodos = draftTodos.filter(t => t.id !== id);
    })
  })
}
```

Aufgabe 10

ImmerJS

- Installiert das Paket immer (npm install immer)
- Nutzt immer, um beim Löschen das Element aus der Liste zu entfernen

React.Fragment

Das DOM klein halten

- React benötigt genau ein Wurzelelement pro Komponente
- Häufig werden Elemente in ein Container-Element zusammengefasst
- Jedes Element wird im DOM eingefügt
- Ein großes DOM bedeutet viel Arbeit für den Browser
- React.Fragment oder <> wirkt wie ein Element, wird aber nicht gerendert

React.Fragment

Ein Beispiel

```
const List = () => {
  …
  return (
    <>
      <Filter />
      {todos.map((todo) => (
        <div>
          {todo.title} {todo.done ? '✅' : '❌'}
        </div>
       )}
    </>
  );
};
```

Output:

```
<form className="filter">...</form>
<div>get up ✅</div>
<div>eat ✅</div>
<div>sleep ❌</div>
```

Debouncing/Throttling

- Throttling: Funktionsausführung verzögern
  - z.B. serverbasiertes Suchfeld, nicht bei jedem Tastendruck einen Request senden.
- Debouncing: Verhindern, dass eine Funktion zu oft aufgerufen wird
  - https://lodash.com/docs/4.17.15#debounce
  - Wartet eine bestimmte Anzahl von Millisekunden bis die Funktion erneut ausgeführt wird.

Aufgabe 11
Debouncing

- Implementiert eine Filtermöglichkeit für die Liste
  - Fügt ein Eingabefeld ein
  - Speichert den Wert des Eingabefelds in einen lokalen State
  - Implementiert einen Effect-Hook, der auf die Änderung des Suchfelds reagiert
  - Setzt einen Timeout und kommuniziert mit dem Server
  - Die Cleanup Routine des Effect hooks cancelt den Timeout

Eindeutige key Props

Warning: Each child in a list should have a unique "key" prop.

- Jeder Listeneintrag erhält eine eindeutige “key” Prop
- React verwendet diese, um festzustellen, ob Elemente wiederverwendet werden können
- Werte sollten zwischen Render-Aufrufen gleich bleiben
  - Gut: Unique Id
  - Schlecht: Math.random
- “key”-Wert muss nur in der aktuellen Liste eindeutig sein

Eindeutige key Props

Ein Beispiel

```
<>
  {todos.map((todo) => (
    <div key={todo.id}>
      {todo.title} {todo.done ? '✅' : '❌'}
    </div>
  ))}
</>
```

Die key-Prop wird nicht im HTML ausgegeben

Aufgabe 11,5

Unique key Prop

- Falls ihr es nicht schon getan habt: behebt die “Each child in a list should have a unique “key” prop.” Warnung

React.memo

Komponenten memoisieren

- Voraussetzung: Komponente rendert bei gleichen Props immer das gleiche
- React.memo prüft nur die Props, Context und State funktionieren trotzdem
- Als zweites Argument kann eine Vergleichsfunktion übergeben werden
  - Erhält prevProps und nextProps
- Nur zur Performanceoptimierung, nicht zur Steuerung des Renderings

React.memo

Ein Beispiel

```
const Details = ({ todo }) => {
  return (
    <div>
      <h2>{todo.title}</h2>
      <p>{todo.description}</p>
    </div>
  );
};

function areEqual(prevProps, nextProps) {
  return prevProps.title === nextProps.title &&
    prevProps.description === nextProps.description;
}

export React.memo(Details, areEqual);
```

Aufgabe 12

React.memo

- Nutzt React.memo in Verbindung mit der ListItem-Komponente
- Die Vergleichsfunktion soll nur true zurückliefen wenn sich eine der folgenden Eigenschaften geändert hat:
  - Id, firstname, lastname, email
- Gebt in der Vergleichsfunktion die old und new Props aus
- Entfernt die key-Prop aus der Liste und seht auch das Ergebnis an

Animationen

CSS Animationen vs. JavaScript Animationen

- CSS Animationen werden unabhängig von React ausgeführt
- CSS Animationen werden vom Browser optimiert
- JavaScript-gesteuerte Animationen sorgen für re-Renders
- CSS Animationen können über die dynamische Zuweisung von Klassennamen gesteuert werden

Animationen

CSS Animation

```
@keyframes h1ColorChange {
  from {
    color: green;
  }
  to {
    color: blue;
  }
}

h1 {
  animation-name: h1ColorChange;
  animation-duration: 10s;
}
```

WebWorker

Die schwere Arbeit auslagern

- Rechenintensive Aufgaben können die GUI blockieren
- WebWorker sind unabhängig vom Hauptprozess
- Kommunikation über Nachrichten
- WebWorker kann nicht auf das DOM zugreifen
- WebWorker kann viele Browser APIs verwenden (z.B. IndexedDB, fetch)

WebWorker

Beispiel

```
const Compute = () => {
  const [state, setState] = useState(0);
  useEffect(() => {
    const worker = new WebWorker('./worker.js');
    worker.postMessage('calculate');
    worker.onMessage = (data) => {
      setState(data);
    };
    return () => {
      worker.terminate();
    };
  }, []);

  return <div>Result is: {state}</div>;
};
```

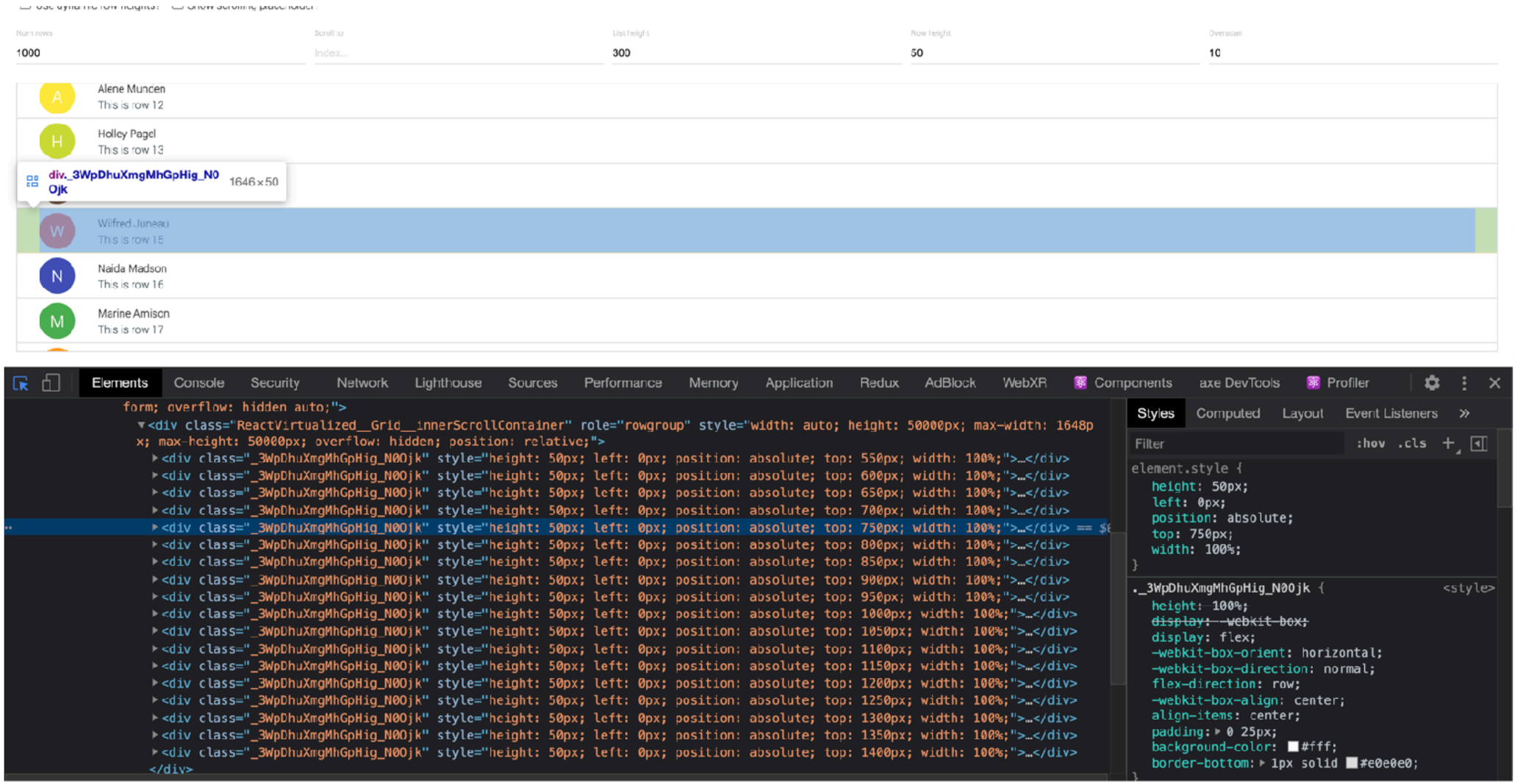
Lange Listen virtualisieren

https://github.com/bvaughn/react-virtualized

- Listen mit mehreren 1000 Elementen können Performanceprobleme verursachen (großes DOM)
- react-virtualized implementiert virtuelles Scrolling
- react-virtualized rendert nur die sichtbaren Komponenten/Elemente
- Erzeugt und löscht Elemente nach Bedarf

Lange Listen virtualisieren

Demo



Aufgabe 13

React virtualised

- Installiert react-virtualised (npm install react-virtualized —legacy-peer-deps)
- Ersetzt die Liste durch eine React Virtualized Tabelle
- Nutzt den folgenden Endpunkt: http://localhost:3001/fakeUsers/100
  - Die Zahl gibt die Anzahl der Datensätze an, die zurückgeliefert werden

Was solltet ihr mitnehmen?

Lessons learned

- Nicht vorschnell optimieren (YAGNI)
- Sourcecode immer möglichst auf Lesbarkeit und nicht auf Performance optimieren
- Allgemeinen Best Practices folgen
- Performancetuning erst wenn sich Probleme abzeichnen
  - Analysieren
  - Verbessern
  - Messen

Fragen?

…dann ist jetzt genau der richtige Zeitpunkt!

Rainer Sturm / pixelio.de

Kontakt

Maiborn Wolff
Mensch IT

Sebastian Springer
sebastian.springer@maibornwolff.de

MaibornWolff GmbH
Theresienhöhe 13
80339 München

@basti_springer

https://github.com/sspringer82