



**WIR BAUEN EINE
PROGRESSIVE WEB APP**



WELCOME



HANS CHRISTIAN OTTO

- » Works at crosscan
- » DEINE-EMAIL@LALALA.DE
- » [@muhdiekuh](#)



ELMAR BURKE

- » Works at Blendle
- » `hallo@elmarburke.de`
- » `@elmarburke`

AGENDA

1. Progressive Web App
2. create-react-app
3. Struktur der Anwendung
4. Redux
5. PouchDB
6. Service Worker, Pre-Rendering und andere Dinge



PROGRESSIVE WEB APP

**“A NEW WAY TO
DELIVER AMAZING
USER EXPERIENCES ON
THE WEB.”**

Google

PROGRESSIVE WEB APPS

Progressive Web Apps (PWA) sind

- » Zuverlässig - laden augenblicklich und zeigen nie den Downasaur, auch in sogar in prekären Netzwerken.
- » Schnell - Reagieren schnell auf Aktionen von Benutzer mit butterweichen Animationen und Scrolling.
- » Fesselnd Fühlen sich wie eine natürliche App auf dem Gerät an.

WAS HEISST DAS NUN FÜR UNS?

Zuverlässig, Schnell, Fesselnd

» Offline als Standard → Offline First

» Animationen bei Übergangen

» Es aber auch nicht übertreiben, pragmatisch bleiben, Visionen wahren

WIE ERREICHEN WIR DAS TECHNISCH

- » `manifest.json`
- » Service Worker
- » Pre-Prendering oder Server Side Rendering
- » CSS Animationen



create-react-app

create-react-app

Erstellt uns eine React App mit eingebauter Build-Umgebung.

```
$ npm install -g create-react-app
```

oder

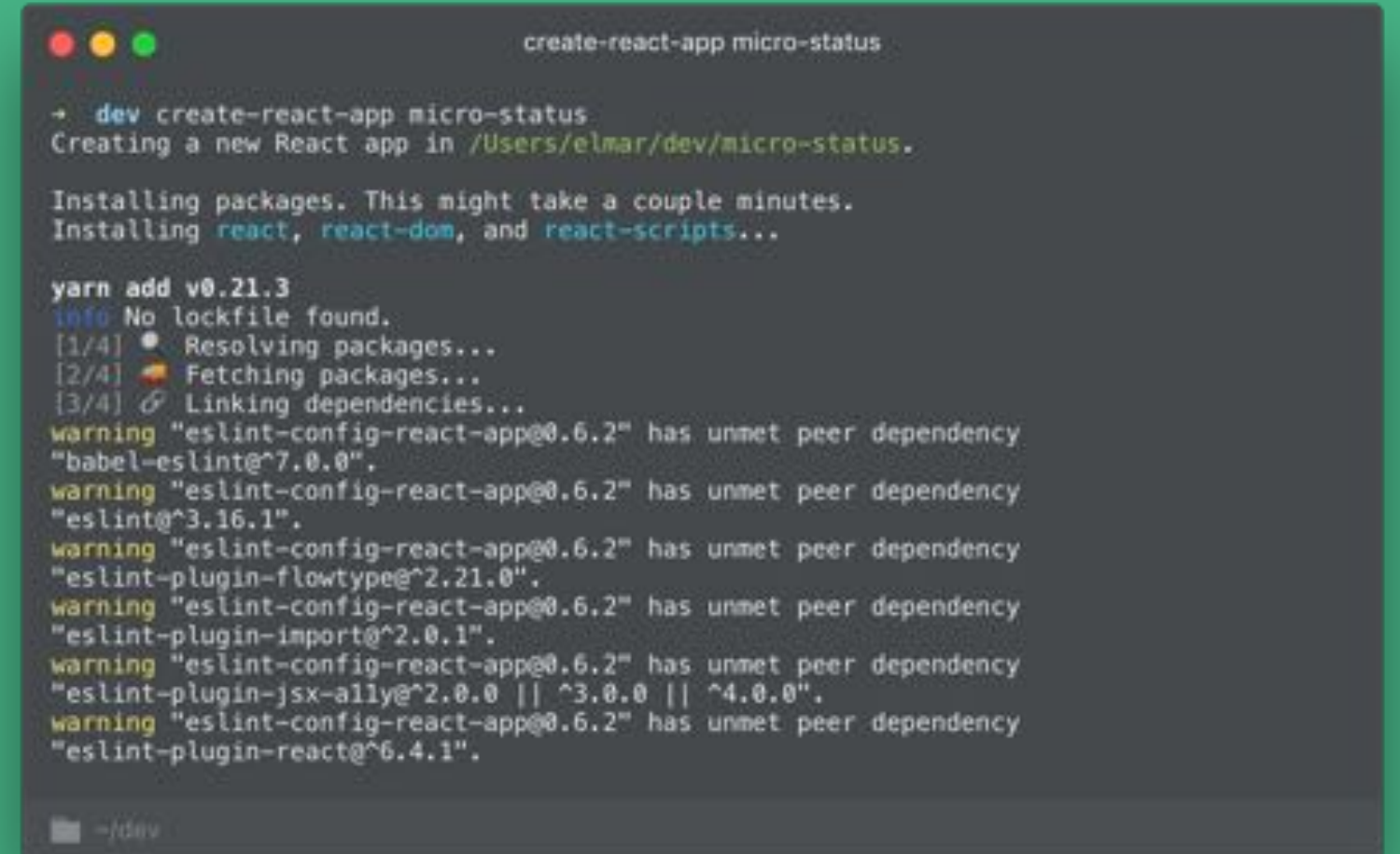
```
$ yarn global add create-react-app
```

3-2-1-



ERSTELLEN DER APP

```
$ create-react-app micro-status
```



```
create-react-app micro-status

→ dev create-react-app micro-status
Creating a new React app in /Users/elmar/dev/micro-status.

Installing packages. This might take a couple minutes.
Installing react, react-dom, and react-scripts...

yarn add v0.21.3
info No lockfile found.
[1/4] • Resolving packages...
[2/4] 📦 Fetching packages...
[3/4] 🔗 Linking dependencies...
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"babel-eslint@^7.0.0".
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"eslint@^3.16.1".
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"eslint-plugin-flowtype@^2.21.0".
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"eslint-plugin-import@^2.0.1".
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"eslint-plugin-jsx-ally@^2.0.0 || ^3.0.0 || ^4.0.0".
warning "eslint-config-react-app@0.6.2" has unmet peer dependency
"eslint-plugin-react@^6.4.1".

~/dev
```



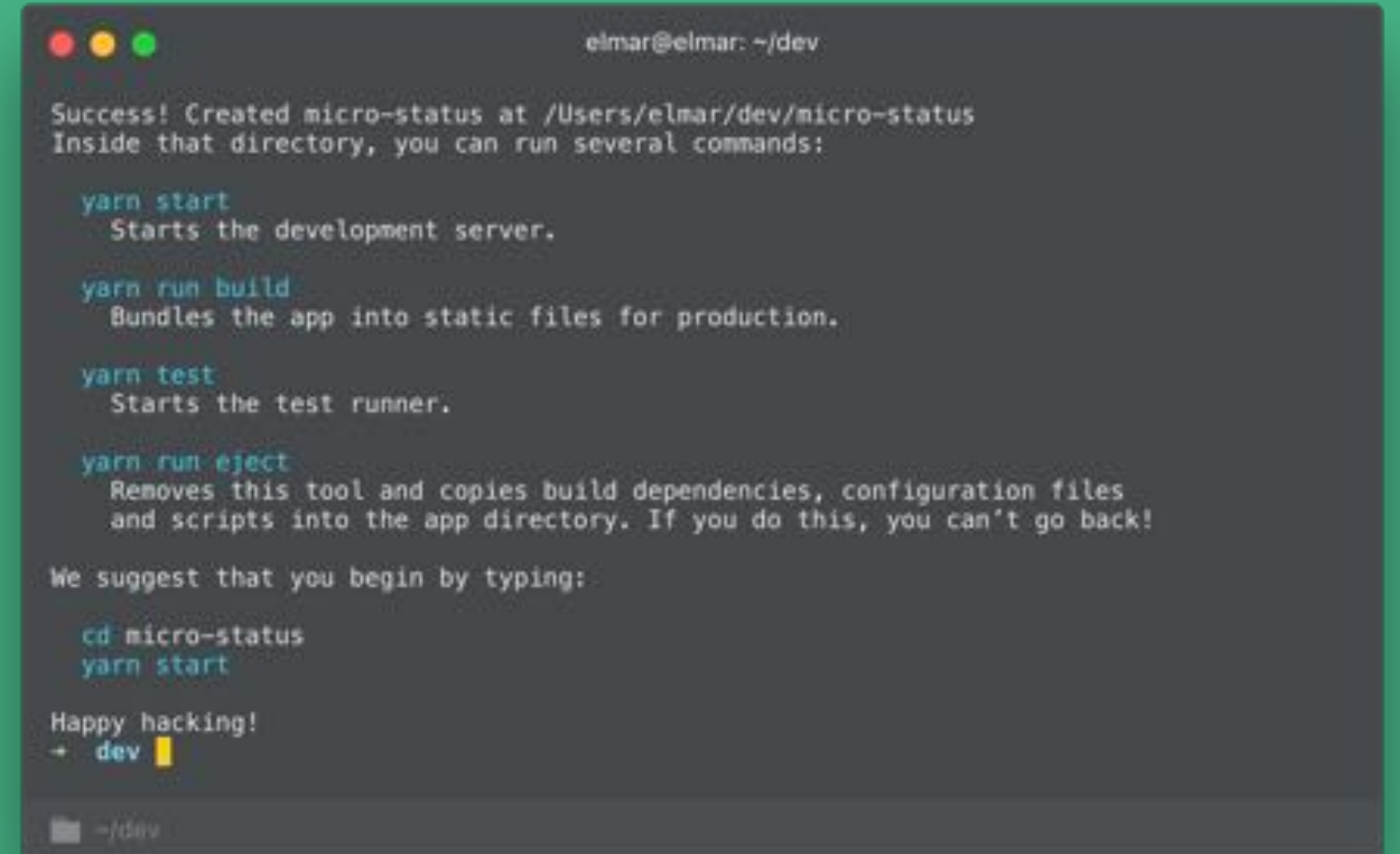
```
$ cd micro-status/
```

```
$ npm start
```

Oder

```
$ yarn start
```

» Im Browser öffnet sich die Anwendung.

A terminal window with a dark background and light green text. The window title is 'elmar@elmar: ~/dev'. The output shows a success message for creating the 'micro-status' directory, followed by a list of available commands: 'yarn start' (starts the development server), 'yarn run build' (bundles the app for production), 'yarn test' (starts the test runner), and 'yarn run eject' (removes the tool and copies dependencies). It then suggests typing 'cd micro-status' and 'yarn start'. The window ends with 'Happy hacking!' and a prompt '→ dev' with a yellow cursor. A file explorer icon and the path '~/dev' are visible at the bottom left.

```
elmar@elmar: ~/dev

Success! Created micro-status at /Users/elmar/dev/micro-status
Inside that directory, you can run several commands:

  yarn start
    Starts the development server.

  yarn run build
    Bundles the app into static files for production.

  yarn test
    Starts the test runner.

  yarn run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd micro-status
  yarn start

Happy hacking!
→ dev
```

```
yarn start

Compiled successfully!

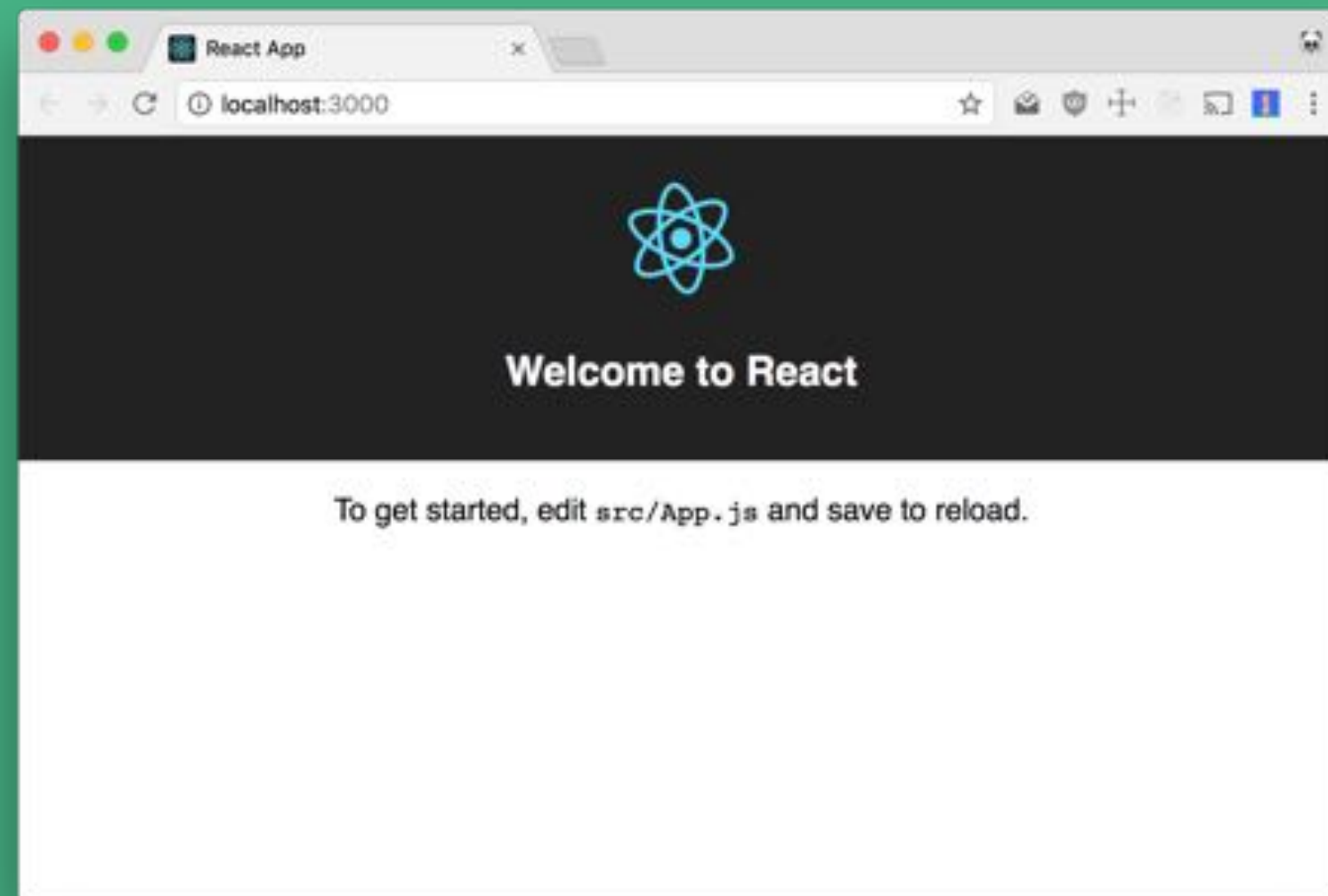
The app is running at:

  http://localhost:3000/

Note that the development build is not optimized.
To create a production build, use yarn run build.


```

~/dev/micro-status





STRUKTUR DER ANWENDUNG

GRUNDSÄTZE

- » Mit React wird HTML in JavaScript-Code geschrieben, weil es unmittelbar zusammen gehört.
- » Styles sollten auch dazu gehören, gute Möglichkeit sind Styled Components.
- » Component-based Architecture
- » Feature-based Architecture
- » "Blackbox-Components"

ORDNERSTRUKTUR

```
/build
/node_modules
/public
/src
  /components
    /TopBar
      /index.js
  /data
  /scences
    /Home
      /components
        /...
      /index.js
    /Info
    /NotFound
  /services
  /index.js
  /App.js
```



ERSTELLE EINE HOMESCENE COMPONENT UND BINDE SIE IN DER `App.js` EIN

- » Component zeigt den Namen der App an
- » Pfad: `src/scences/Home/index.js`
- » Entferne alles unnötige in `src/App.js`
- » Zeige der Komponente in `src/App.js`

GEHT DAS AUCH SCHÖN?

- » Globale Styles in `App.css`
 - » Hintergrundfarben, Font-Size etc
- » Styles auf Component-Basis mit `styled-components` 🍌

STYLED COMPONENTS

```
$ yarn add styled-components
# oder
$ npm install --save styled-components

// src/components/Content.js
import styled from 'styled-components'

const Content = styled.div`
  max-width: 600px;
`

export default Content
```



DIE HINTERGRUNDFARBE DER APP SOLLTE LILA SEIN UND DER CONTENT DUNKEL-GRAU

» Mache den Hintergrund lila

```
body { background-color: #9b58b5; }
```

» Erstelle eine StyledComponent

```
const Content = styled.div`  
  max-width: 600px;  
  background-color: #363636;  
  color: #dcdcdc;  
`
```

» Benutze die Styled Component in der HomeScene



STATUS COMPONENT

- » Zeigt einen Status an
- » Hat den Status Text und einen Zeitpunkt



COMPOSE COMPONENT

- » Formular
- » Button zum absenden
- » Bekommt ein onSubmit als Property



ERSTELLE EINE TOPBAR

- » Schwarzer Hintergrund
- » etwa 3em hoch
- » So breit wie Content
- » Später kommt hier ein Status-Indikator hinzu



STATE WITH REDUX

ORDNERSTRUKTUR (SCHON WIEDER)

```
/src
  /components
  /data
    /status
      /api.js
      /actions.js
      /reducer.js
    /reducer.js
  /scences
    /Home
      /components
      /index.js
      /reducer.js
    /reducer.js
  /index.js
  /App.js
  /reducer.js
```




WIR BINDEN REDUX EIN!

Wir installieren Redux, eine Verbindung zwischen React und Redux sowie einige Middleware für Redux um async einfacher zu machen.

Außerdem eine Library um Daten zu normalisieren.

```
$ yarn add redux react-redux redux-thunk redux-promise-middleware normalizr
```



ERSTELLEN WIR EINEN REDUCER UND EINE ACTION

- » `fetchStatus(): { type: '...', payload: {...} }`
- » Reducer speichert Daten in `{ all: [] }`
- » Weil der Reducer weiß, wo die Daten sind, liegt dort auch der Selector



ERSTELLEN WIR EINEN STORE

- » Erstellen der `src/configureStore.js`
- » Einbauen der Provider Component
- » Übergabe des Stores



VERBINDEN WIR DAS MIT REACT

```
import { connect } from 'react-redux'

...
import { getAllStatus } from '../data/status/reducer'
import { fetchStatus } from '../data/status/actions'

...

const mapStateToProps = (state) => ({
  status: getAllStatus(state)
})

export default connect(mapStateToProps, {
  fetchStatus
})(Home)
```


WIE SEHE ICH JETZT DATEN?

- » Unser Store ist leer.
- » Wir haben noch keine Möglichkeit, Daten in unseren Store zu laden oder hinzuzufügen.
- Wir brauchen ein Formular



ERSTELLE EIN FORMULAR ZUM ABSETZEN DES STATUS

- » `src/scenes/Home/components/Compose.js`
- » `const TextArea = styled.textarea`...``
- » `const Button = styled.button`...``
- » `const Compose = () => {
 <Form><TextArea /><Button /></Form>
}`
- » (noch) kein Event-Handling

UND JETZT VERBINDEN WIR DAS MIT DER REDUXWELT

- » Callback in `Compose.js` nachdem die Daten abgesendet wurden
- » Action schreiben
- » Action in der `HomeScene` verbinden
- » Reducer macht was mit der Action

I PROMISE TO DELIVER DATA

- » Bisher liefern unsere Actions immer die Daten mit.
- » Wenn wir Daten laden, starten wir eine Action, die mehrere Actions nach sich ziehen kann.
- » Promise Middleware hilft uns hier



PROMISE MIDDLEWARE EINFÜGEN UND DATEN LADEN (SIMULIEREN)

- » Action `FETCH_DATA` erstellen und im Reducer einbauen
- » Aufruf in einem Lifecycle Hook der HomeScene

? NACH DEM RELOAD IST JA ALLES WEG?

- » Unser State lebt im Arbeitsspeicher des Browsers
 - » Wird die Anwendung neu initialisiert, dann ist alles weg
- Wir müssen unsere Daten irgendwo persistieren

POUCHDB



SEPRATES DECKSET

A hand holding a pen, poised to draw on a blueprint. The entire image is overlaid with a semi-transparent green filter. The text 'SERVICE WORKER, PRE-RENDERING UND ANDERE DINGE' is written in bold, white, uppercase letters across the center of the image.

SERVICE WORKER, PRE-RENDERING UND ANDERE DINGE

SERVICE WORKER

WARUM?

- » Offline-Verfügbarkeit der App Shell*
- » Kann auch Push Messages und Background-Sync

FERTIGE TOOLS, DIE BEI DER ERSTELLUNG VON SW HELFEN:

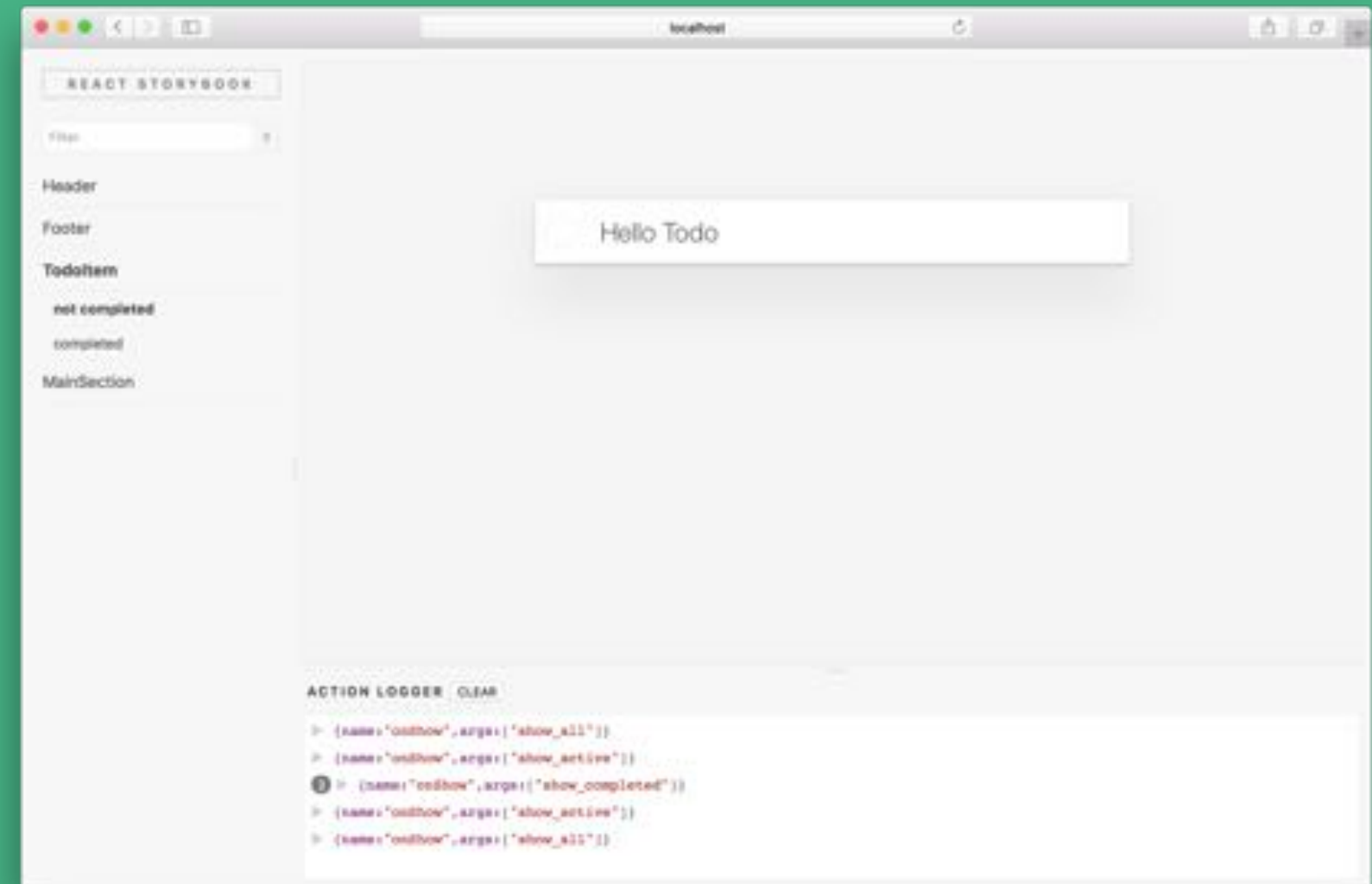
- » Für Webpack: `offline-plugin`
- » Für die Command Line: `sw-precache`

PRE-RENDERING

- » Wir können die App Shell schon als fertiges HTML rendern, dann kann der Browser schon ohne JavaScript beginnen, zu rendern.
- » Beschleunigt die Startzeit Anwendung beim ersten Aufruf. Danach haben wir Service Worker
- » Weniger Aufwendig als SSR
- » Aufwand/Nutzen-Faktor deutlich höher
- » Für die Command Line: `react-prerender`

STORYBOOK

- » Bauen von Components, die los gelöst von der Anwendung entwickelt werden
- » Für UI Elemente wie Buttons und Input-Elemente
- » Vereinfacht den Umgang mit verschiedenen Disziplinen
- » Für die Command Line:
`react-storybook`



THAT'S IT

